МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра кібербезпеки

ПРОГРАМУВАННЯ ДЛЯ НАУКОВИХ ДОСЛІДЖЕНЬ

ОПОРНИЙ КОНСПЕКТ ЛЕКЦІЙ

для здобувачів освітньо-професійної програми
«Кібербезпека»
другого (магістерського) рівня вищої освіти
спеціальності «Кібербезпека та захист інформації»

Тернопіль – ЗУНУ
2025

Опорний конспект лекцій з курсу «Програмування для наукових досліджень» для здобувачів освітньо-професійної програми «Кібербезпека» другого (магістерського) рівня вищої освіти спеціальності «Кібербезпека та захист інформації» / Укл.: Касянчук М.М., Івасьєв С.В. – Тернопіль: ЗУНУ, 2025. – 106 с.

Відповідальний за випуск: Івасьєв С.В., к.т.н., доцент кафедри кібербезпеки

Рецензенти:

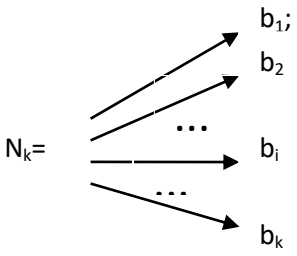
Загородна Н.В. к.т.н., доцент, завідувач кафедри кібербезпеки Тернопільського національного технічного університету імені Івана Пулюя

Батько Ю.М. к.т.н., доцент кафедри комп'ютерної інженерії факультету комп'ютерних інформаційних технологій Західноукраїнського національного університету

Методичні вказівки розглянуті та схвалені на засіданні кафедри кібербезпеки, протокол № 10 від 24 квітня 2025 р.

ТЕМА 1: ОПРАЦЮВАННЯ БАГАТОРОЗРЯДНИХ ЧИСЕЛ.

В основу цілочисельного перетворення СЗК покладена Китайська теорема про залишки [79]. Суть прямого перетворення цілочисельної форми СЗК полягає в тому, що, згідно з теоремами про залишки, будь-яке ціле число можна однозначно перетворити набором найменших невід'ємних залишків в системі взаємно простих модулів



$$b_i = resN_k (\mathrm{mod}\ p_i), \qquad (1.1)$$

що відповідає рішенню діофантового рівняння

$$N_k = b_i (\mathrm{mod}\ p_i), \qquad (1.2)$$

яке відповідає цілочисельному рішенню лінійного рівняння:

$$N_k = a_i p_i + b_i, \qquad (1.3)$$

де a_i – ранг; b_i – найменший невід'ємний залишок.

При цьому діапазон кодування чисел N_k дорівнює:

$$P = \prod_{i=1}^{k} p_i \ ; \ 0 \le N_k \le P-1. \qquad (1.4)$$

Таким чином, ціле число N_k однозначно представляється набором залишків b_i.

Зворотне перетворення цілочисельної форми СЗК виконується згідно аналітичного виразу [25]

$$N_k = res \sum_{i=1}^{k} b_i \cdot B_i (\mathrm{mod}\ P), \qquad (1.5)$$

де B_i – базисні числа СЗК, які обчислюються згідно діофантового рівняння:

$$B_i = \frac{P}{p_i} \cdot m_i \equiv 1 (\mathrm{mod}\ P_i). \qquad (1.6)$$

Тобто, для виконання зворотнього перетворення цілочисельної форми СЗК необхідно задати параметри:

1. Набір взаємно простих модулів $p_1, p_2, ..., p_i, ..., p_k$;

2. Діапазон кодування чисел P;

3. Набір базисних чисел B_1, B_2, ..., B_i, ..., $B_к$;

4. Набір коефіцієнтів, які забезпечують ортогональність перетворень m_1, m_2, …, m_i, …, m_k;

5. Аналітичні вирази прямого і зворотнього перетворення:

$$b_i = res\, N_k (\mathrm{mod}\ p_i),$$

$$N_k = res \sum_{i=1}^{k} b_i * B_i (\mathrm{mod}\ P). \qquad (1.7)$$

У таблиці 2.1 показаний приклад кодування чисел в цілочисельній формі СЗК, для наступної системи модулів:

$p_1=2$, $p_2=5$; $P=2*5=10$; $0 \le N_k \le 9$.

$$B_1 = \frac{10}{2} \cdot m_1 = 1(\mathrm{mod}\ 2), m_1 = 1; B_1 = 5;$$

$$B_2 = \frac{10}{5} \cdot m_2 = 1(\mathrm{mod}\ 5), m_2 = 3; B_2 = 6.$$

Таблиця 2.1 - Кодування чисел в цілочисельній формі СЗК.

N_k	b_1	b_2
0	0	0
1	1	1
2	0	2
3	1	3
4	0	4
5	1	0
6	0	1
7	1	2
8	0	3
9	1	4

Теорія виконання алгоритму операцій додавання, віднімання та множення в цілочисельній формі СЗК детально викладена в роботі [68]. В основу арифметики залишкових класів покладено глибоке розпаралелення обробки даних, яке виконується по кожному модулю p_i окремо з виключенням міжрозрядних переносів.

Алгоритми визначення операції додавання, віднімання і множення в СЗК виконується на основі наступних математичних виразів:

1) Сумування $x_k+y_k \le P-1$:

$$x_k=(a_1, a_2, \dots a_i, \dots a_k) \qquad x_k+y_k=(a_1, a_2, \dots a_i, \dots a_k) \qquad (1.8)$$

$$y_k=(b_1, b_2, \dots b_i, \dots b_k) \qquad \underline{+\ (b_1, b_2, \dots b_i, \dots b_k)},$$

$$(c_1, c_2, \dots c_i, \dots c_k)$$

де $c_i=res(a_i+b_i) \bmod p_i$;

2) Сумування $x_k+y_k \ge P$:

$$x_k+y_k=(a_1, a_2, \dots a_i, \dots a_k); \qquad (1.9)$$

$$\underline{+\ (b_1, b_2, \dots b_i, \dots b_k)},$$

$$(d_1, d_2, \dots d_i, \dots d_k)$$

де $d_i=res(a_i+b_i) \bmod P$;

3) Віднімання ($x_k \ge y_k$):

$$x_k-y_k=(a_1, a_2, \dots a_i, \dots a_k) \qquad (1.10)$$

$$\underline{-\ (b_1, b_2, \dots b_i, \dots b_k)},$$

$$(e_1, e_2, \dots e_i, \dots e_k)$$

де $e_i=res(a_i-b_i) \bmod P_i$;

4) Віднімання ($x_k < y_k$) $x_k-y_k=(x_k-y_k)=P+x_k-y_k$:

$$x_k-y_k=(a_1, a_2, \dots a_i, \dots a_k) \qquad (1.11)$$

$$\underline{-\ (b_1, b_2, \dots b_i, \dots b_k)},$$

$$(f_1, f_2, \dots f, \dots f_k);$$

де $f_i=res(a_i-b_i) \bmod P_i$;

5) Множення ($x_k \cdot y_k$):

$$x_k*y_k=(a_1, a_2, \dots a_i, \dots a_k) \qquad (1.12)$$

Nik

$$\underline{(b_1, b_2, \dots b_i, \dots b_k)},$$

$$(j_1, j_2, \dots j, \dots j_k)$$

де $j_i=res(a_i \cdot b_i) \bmod P_i$.

5) Ділення (x_k/y_k).

Створення повнофункціональних процесорів в базисі Крестенсона потребує вдосконалення методів реалізації операції ділення над числами з заданими наборами залишків.

Запропонований алгоритм виконання операції ділення [119] на основі доповнення в класах діофантових рівнянь, над великорозрядними числами, заданими в базисі Радемахера, що має вигляд:

$$Z = X \diagup Y,$$

в якому $X_{(2)} = (x_{n-1},...,x_i,...,x_0); x_i \in \overline{0,1}; Y_{(2)} = (y_{n-1},...,y_i,...,y_0); y_i \in \overline{0,1}$,

$$Z_{(2)} = \underbrace{(z_{r-1},...,z_i,...,z_0)}_{Z_0} + \underbrace{(b_{r-1},...,b_i,...,y_0)}_{b_0}; y_i \in \overline{0,1};$$

де Z_0 - ціла частина (ранг);

b_0 - дробова частина (залишок).

Рішення цієї задачі відповідає рівнянню в цілих числах:

$$X = Z_0 \cdot Y + b_0;\ X^* = Z_0 \cdot Y,$$

яке відповідає порівнянню:

$$X \equiv b_0 (\mathrm{mod}\, Y). \qquad (1.13)$$

Застосувавши принцип доповнення відносно рівняння (2.13) в класі діофантових рівнянь, отримаємо:

$$X^* + b_0 \equiv 0 (\mathrm{mod} Y).$$

Тобто, якщо Y можна представити p_j, яке входить в склад системи взаємно простих модулів базису Крестенсона, метод ділення реалізується згідно наступного алгоритму:

1) До ділимого X додаємо доповнення b_0 по модулю Y.

2) Спрощуємо систему взаємопростих модулів СЗК базису Крестенсона, які представлені простими числами $p_1, p_2,..., p_j,..., p_k$, вилучивши, при цьому, з нього модуль Y.

3) Визначаємо діапазон зміни ділимого у границях: $0 \le X \le P_j$, де $P = \prod_{j=1}^{k} p_j$.

Розглянемо рішення задачі коли $Y = p_j$.

4) На основі представлення чисел в розмежованій СЗК [89] перетворюємо двійкові числа X та Y у базис Крестенсона з набором модулів $\{p_j, Y, j \in \overline{1,k}\}$ на основі матричних алгоритмів (рис.2.1).

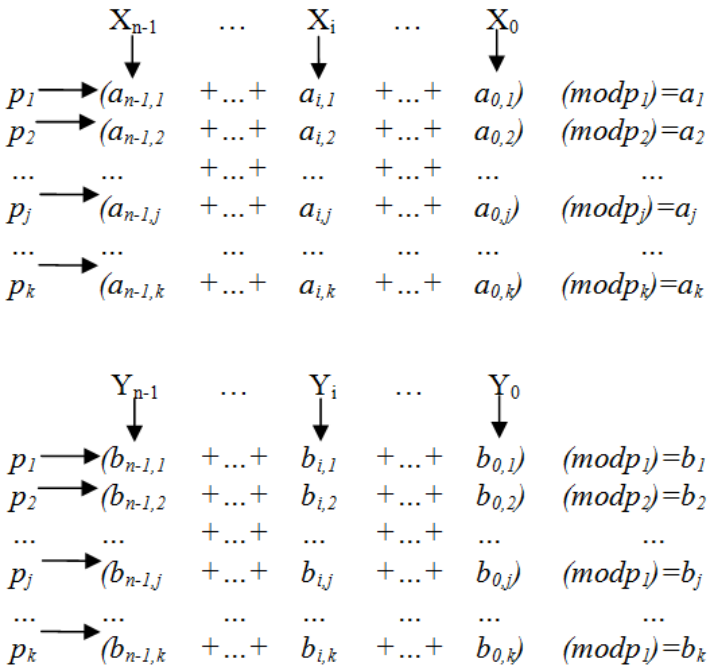


Рисунок 1 - Математичні алгоритми міжбазисного перетворення Радемахера-Крестенсона на основі розмежованої СЗК, відповідно числа X та Y.

5) До числа X, представленого в СЗК, додаємо числове значення доповнюючого залишку b_j^* по модулю p_j, тобто

$$\begin{array}{r} X = (b_1,...,b_j,...,b_k) \\ + \quad b_j^* = (b_1^*,...,b_j^*,...,b_k^*) \\ \hline X^* = (x_1^*,...,x_j^*,...,x_k^*) \end{array}.$$

6) На основі алгоритму модульного множення чисел в базисі Крестенсона вирішуємо задачу знаходження інформативних залишків ділимого, згідно наступного матричного алгоритму:

$$\begin{array}{r} Y = (y_1,...,y_j,...,y_k) \\ \times \quad Z^* = (z_1^*,...,z_j^*,...,z_k^*) \\ \hline X^* = (x_1^*,...,x_j^*,...,x_k^*) \end{array}.$$

Тобто $(y_{i\neq j} \cdot z_{i\neq j}^*) \bmod p_{i\neq j} = x_{i\neq j}^*$, що означає видалення неінформативного залишку по модулю p_j.

7) З представлення СЗК вилучаються неінформативні модулі із залишками y_i, що дорівнюють нулю. В результаті формуємо нову СЗК на основі інформативних $y_i \neq y_j$

, тобто $Z^* = (z_1^*,...,z_i^*,...,z_l^*); i \in \overline{1,l}$, у системі модулів $p_1,..,p_i,...,p_l$. При цьому, $\prod_{i=1}^{l} p_i \geq X_{\max}$.

8) Для отримання правильного цілого результату ділення виконується відніманням від інформативних залишків СЗК згідно виразу:

$$\begin{array}{r} Z^* = (z_1^*,...,z_i^*,...,z_l^*) \\ - \\ \hline Z = (z_1,...,z_i,...,z_l) \end{array} \quad \begin{array}{c} \\ (1,...,1,...,1) \\ \\ \end{array}.$$

Таким чином, результат операції ділення отримаємо у вигляді:

$$X/Y = \{Z_i\} + \{b_i\}; i \in \overline{1,l};\ j \in \overline{1,k}.$$

Проведені дослідження розробленого методу виконання операції ділення в СЗК дозволяють оцінити часову складність процесора ділення в базисі Крестенсона на основі виразу

$$\tau_{div} = (l+1) \cdot \nu,$$

де l - число модулів, добуток яких представляє дільник Y.

Однією із переваг цього методу є відсутність операції порівняння в алгоритмі ділення, що зменшує часову складність операції ділення в СЗК та дозволяє розпаралелити її виконання в бінарно-розмежованій СЗК.

На рис.2.2 показані порівняльні характеристики швидкодії виконання арифметико-логічних операцій в базисі Радемахера та Крестенсона.

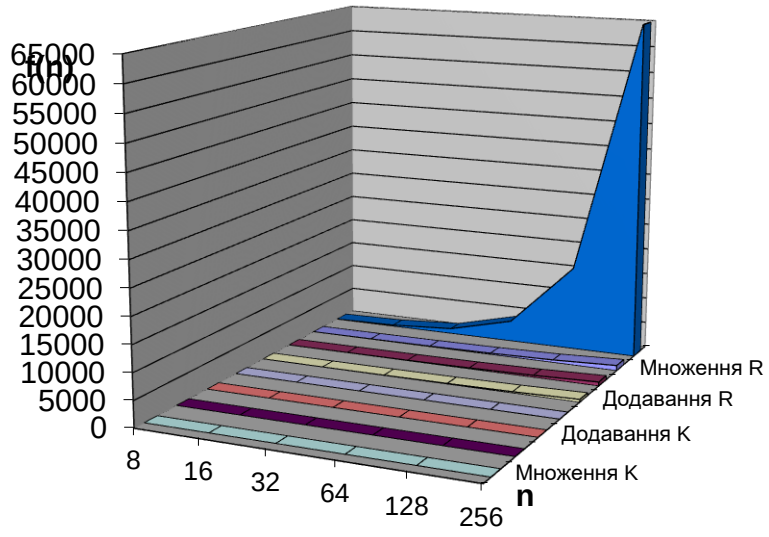


Рисунок 2 - Порівняльні характеристики швидкодії виконання арифметико-логічних операцій у базисах Радемахера та цілочисельного перетворення Крестенсона

Недоліком цілочисельної форми перетворення СЗК є практична відсутність простої операції порівняння чисел, що суттєво ускладнює реалізації алгоритмів та відповідних процесів ділення. В той же час, переваги однотактного матричного виконання інших арифметичних операцій забезпечує широкі перспективи застосування теоретичних основ цілочисельного перетворення СЗК для створення та широкомасштабного впровадження супершвидкісних процесорів в комп'ютерних мережах. Особливу перспективу має застосування цілочисельної форми СЗК при створенні спецпроцесорів, в яких базовими операціями є додавання та множення. Прикладом таких процесорів є цифрові фільтри, обчислювачі кореляційних та спектральних характеристик випадкових процесів, а також операції над матрицями та алгебраїчними поліномами.

ТЕМА 2: МОЖЛИВОСТІ PYTHON В НАУКОВИХ ДОСЛІДЖЕННЯХ.

Застосування програмування на Python для вирішення наукових задач. **Disclaimer:** для багатьох програмістів, як і для багатьох людей, пов'язаних таким чи іншим чином з науковою роботою, дані тези можуть бути очевидними.

Виявлення прихованих періодичностей, трендів тощо. В даний момент нас цікавить розклад часових рядів технологічних параметрів (допустимо, температури вузла, або вібраційної швидкості підшипника) в нетривіальні базиси. Зазвичай розклад відбувається по функціях Фур'є, але є інші базиси, простіші для обробки комп'ютерами: функції Радемахера, Хаара, Крестенсона, Уолша. І дають вони доволі цікаві результати.

Функція (або вейвлет) Хаара застосовується для стистення зображень. Сама функція виглядає дуже просто:

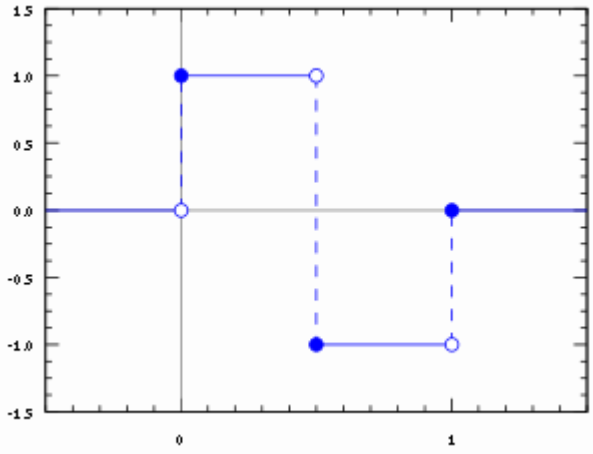


Рисунок 1

Як же стискалося зображення? Просто: для чорно-білого фото бралися значення сусідніх пікселів і в один масив записувалася їх півсума, а в другий їх піврізниця. І так для кожної пари пікселів. А потім масив піврізниць просто відсікався. Таким чином, ми отримували масив удвічі меншого розміру з усередненими значеннями для кожної пари пікселів.

А тепер найцікавіше: а як же втрати? Втрати для кожного конкретного випадку будуть дуже різними. Допустимо у нас є зображення, подібне до шахової дошки - білий піксель-чорний піксель і т.д. Після першого такого стискання у нас вийде повністю сіре зображення. Зміст повністю втратиться, чого нам дуже не хотілося б.

На щастя, реальні фотографії характерні тим, що сусідні пікселі мають дуже близькі значення. І до них цей метод стиснення застосовується доволі ефективно.

Ми зациклилися на зображеннях. В даному випадку чорно-білі зображення - це всього лиш масиви значень відтінку сірого для кожного пікселя. Довгий час працюючи з технологічними параметрами мого об'єкта дослідження, я приблизно вивчив їх характер змін у часі. Специфіка така - з кожного давача значення знімається кожних 5 хвилин. І, так вже сталося, що дані технологічні об'єкти майже не мають змін у параметрах функціонування з коротким періодом - сусідні значення у більшості випадків рівні, або майже рівні. Тому я вирішив просто для себе провести

Стискання масиву часового ряду одного параметра з допомогою вейвлета Хаара. Підключимо для цього дві бібліотеки - scipy.io (для завантаження даних з файлу .mat, в якому вони зберігалися по замовчуванню) та pylab (для перегляду проміжних резутьтатів). Функція load (для завантаження даних):

```
def load():
    data = scipy.io.loadmat('data0.mat')
    mdata=data['data0']
```

```
    return mdata
```

Функція compress (для стискання цих даних):

```
def compress(mdata):
    a=[]
    b=[]
    isEven=[]
    for e in mdata:
        b.append(e)
    if len(mdata)%2==0:
        isEven.append(True)
    else:
        b.append(mdata[len(mdata)-1])
        isEven.append(False)
    for e in range(int(len(b)/2)):
        a.append((b[e*2]+b[e*2+1])/2)
    return a
```

В принципі, пояснювати тут нічого. Єдине - масив isEven створений щоб застосовувати в перспективі, поки що обходиться без нього. Відповідно, приведу тут же функцію decompress:

```
def decompress(a):
    rdata=[]
    for e in a:
        rdata.append(e)
        rdata.append(e)
    return rdata
```

Все просто - вона дублює значення зі стиснутого масиву. Що ж у нас вийшло в резутьтаті? Ось вихідні дані:

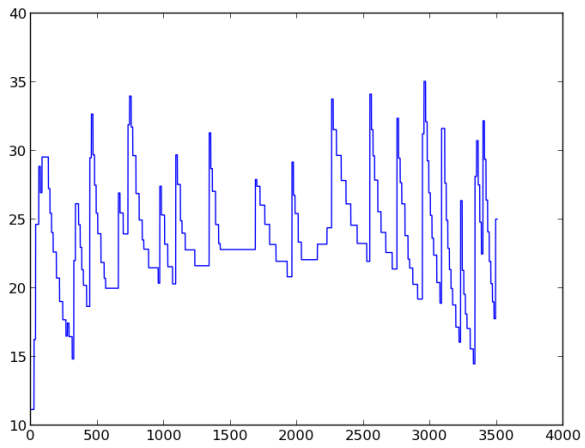


Рисунок 2

А ось відновлені:

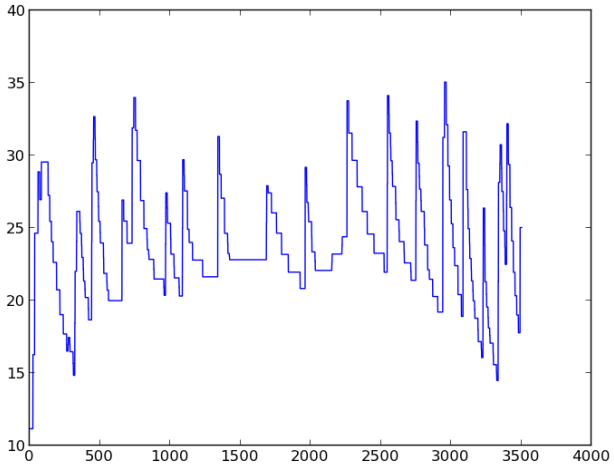


Рисунок 3

Ви бачите різницю? І це при стисненні в 2 рази! Однак, людське око - штука непевна. Давайте подивимося, що скаже нам про ці дані комп'ютер. Для порівняння вихідних та відновлених даних була написана функція compare:

```
def compare(mdata,rdata):
    i=0
    mmean=sum(mdata)/len(mdata)
    rmean=sum(rdata)/len(rdata)
    s1=0
    s2=0
    s3=0
    for e in range(len(mdata)):
        s1+=(mdata[e]-mmean)*(mdata[e]-mmean)
        s2+=(rdata[e]-rmean)*(rdata[e]-rmean)
        s3+=(rdata[e]-rmean)*(mdata[e]-mmean)
    r=s3/((s1*s2)**0.5)
    print r
    for e in range(len(mdata)):
        if mdata[e]==rdata[e]:
            i=i+1
    print float(i)/len(mdata)
```

Функція ця виводить два значення. Спочатку про друге - цілком логічна штука - відношення співпадінь значень у відновлених і вихідних даних до загальної кількості даних. Однак математики користуються іншим критерієм - коефіцієнтом лінійної кореляції (студенти, які вчили теорію ймовірності і матстатистику, мали б бути знайомі з цим поняттям). Його формула дуже проста, вона є у Вікіпедії, не буду переписувати інформацію звідти. Власне змінна r якраз і є цим коефіцієнтом.

Які ж вони результати дають нам при стисненні в два рази?

Коефіцієнт лінійної кореляції: 0.99426435

Кількість значень, що співпали: 0.959474885845

Погодьтеся, не найбільші втрати при такому стисненні. Кореляція в 99,4% - це взагалі прекрасний результат.

Ви можете сказати: окей, чувак, ти взяв такий відрізок з однієї з функцій і воно так красиво працює. Що з іншими? Проведемо ці самі операції стиснення з 5 різними параметрами - серед них є і температура, і тиск, і віброшвидкість. Більше того, додамо результати для стиснення в 4 та 8 раз, провівши стиснення та відновлення 2 та 3 рази відповідно. Що ми отримуємо? Ідентичність значень:

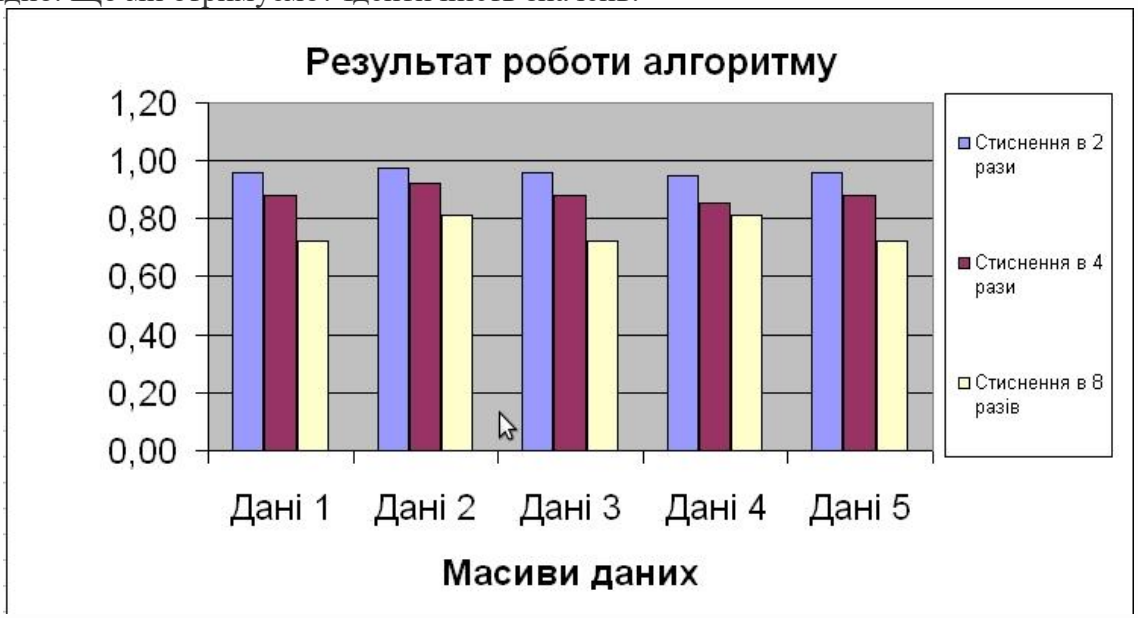


Рисунок 4

Коефіцієнт кореляції для цих значень:

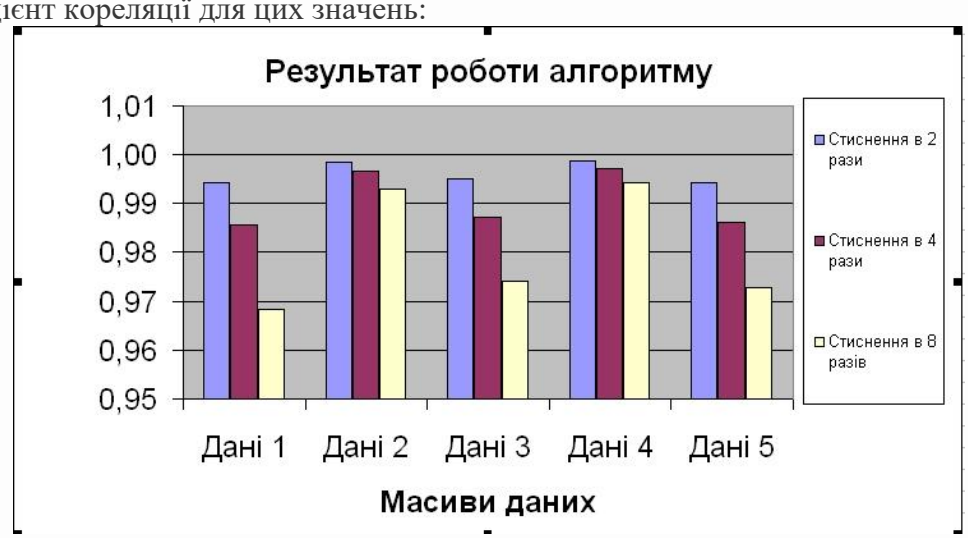


Рисунок 5

Можете побачити, як відновлення залежить від характеру даних. Тому, якщо оформляти цей алгоритм, хорошим рішенням буде його проектування з деякою мірою адаптивності відносно вихідних даних. Тим не менше, відновлена функція після стиснення у 8 разів корелює з вихідними даними на 96-99%.

ТЕМА 3: МОДУЛЬ SHUTIL.

Модуль shutil містить набір функцій високого рівня для обробки файлів, груп файлів, і папок. Зокрема, доступні тут функції дозволяють копіювати, переміщати і видаляти файли і папки. Часто використовується разом з модулем os .

Операції над файлами і директоріями

shutil.copyfileobj (fsrc, fdst [, length]) - скопіювати вміст одного файлового об'єкта (fsrc) в інший (fdst). Необов'язковий параметр length - розмір буфера при копіюванні (щоб весь, можливо величезний, файл не читався цілком в пам'ять).

При цьому, якщо позиція покажчика в fsrc ні тел.0 (тобто до цього було зроблено щось на зразок fsrc.read (47)), то буде копіюватися вміст починаючи з поточної позиції, а не з початку файлу.

shutil.copyfile (src, dst, follow_symlinks = True) - копіює вміст (але не метадані) файлу src в файл dst. Повертає dst (тобто куди файл був скопійований). src і dst це рядки - шляхи до файлів. dst повинен бути повним ім'ям файлу.

Якщо src і dst представляють собою один і той же файл, виняток **shutil.SameFileError.**

Якщо dst існує, то він буде перезаписаний.

Якщо follow_symlinks = False і src є посиланням на файл, то буде створена нова символічна посилання замість копіювання файлу, на який ця символічна посилання вказує.

shutil.copymode (src, dst, follow_symlinks = True) - копіює права доступу з src в dst. Вміст файлу, власник, і група не змінюються.

shutil.copystat (src, dst, follow_symlinks = True) - копіює права доступу, час останнього доступу, останньої зміни, і прапори src в dst. Вміст файлу, власник, і група не змінюються.

shutil.copy (src, dst, follow_symlinks = True) - копіює вміст файлу src в файл **або папку** dst. Якщо dst є каталогом, файл буде скопійований з тією ж назвою, що було в src. Функція повертає шлях до місцезнаходження нового скопійованого файлу.

Якщо follow_symlinks = False, і src це посилання, dst буде посиланням.

Якщо follow_symlinks = True, і src це посилання, dst буде копією файлу, на який посилається src

copy () копіює вміст файлу, і права доступу.

shutil.copy2 (src, dst, follow_symlinks = True) - як copy (), але намагається копіювати всі метадані.

shutil.copytree (src, dst, symlinks = False, ignore = None, copy_function = copy2, ignore_dangling_symlinks = False) - рекурсивно копіює все дерево директорій з коренем в src, повертає директорію призначення.

Директорія dst не повинна існувати. Вона буде створена, разом з пропущеними батьківськими директоріями.

Права і часи у директорій копіюються copystat (), файли копіюються за допомогою функції copy_function (за замовчуванням shutil.copy2 ()).

Якщо symlinks = True, посилання в дереві src будуть посиланнями в dst, і метадані будуть скопійовані настільки, наскільки це можливо.

Якщо False (за замовчуванням), будуть скопійовані вміст і метадані файлів, на які вказували посилання.

Якщо symlinks = False, якщо файл, на який вказує посилання, не існує, буде додано виняток в список помилок, у виключенні shutil.Error в кінці копіювання.

Можна встановити прапор ignore_dangling_symlinks = True, щоб приховати цю помилку.

Якщо ignore НЕ None, то це повинна бути функція, що приймає в якості аргументів ім'я директорії, в якій зараз copytree (), і список вмісту, що повертається os.listdir

(). Оскільки copytree () викликається рекурсивно, ignore викликається 1 раз для кожної піддиректорії. Вона повинна повертати список об'єктів щодо поточного імені директорії (тобто підмножина елементів у другому аргументі). Ці об'єкти не будуть скопійовані.

shutil.ignore_patterns (* patterns) - функція, яка створює функцію, яка може бути використана в якості ignore для copytree (), ігноруючи файли і директорії, які відповідають glob -style шаблонами.

наприклад:

```
copytree (source, destination, ignore = ignore_patterns ( '* .pyc', 'tmp *'))
# Скопіюйте всі файли, крім закінчуються на .pyc або починаються з tmp
```

shutil.rmtree (path, ignore_errors = False, onerror = None) - Видаляє поточну директорію і все піддиректорії; path повинен вказувати на директорію, а не на символічне посилання.

Якщо ignore_errors = True, то помилки, що виникають в результаті невдалого видалення, будуть проігноровані. Якщо False (за замовчуванням), ці помилки будуть передаватися оброблювачу onerror, або, якщо його немає, то виняток.

На ОС, які підтримують функції на основі файлових дескрипторів, за замовчуванням використовується версія rmtree (), що не вразлива до атак на символічні посилання.

На інших платформах це не так: при підібраному часу і обставин "хакер" може, маніпулюючи посиланнями, видалити файли, які недоступні йому в інших обставинах.

Щоб перевірити, вразлива система до подібних атак, можна використовувати атрибут rmtree.avoids_symlink_attacks.

Якщо заданий onerror, це повинна бути функція з 3 параметрами: function, path, excinfo.

Перший параметр, function, це функція, яка створила виключення; вона залежить від платформи і інтерпретатора. Другий параметр, path, це шлях, який передається функції. Третій параметр, excinfo - це інформація про виключення, яка повертається sys.exc_info (). Винятки, викликані onerror, не обробляються.

shutil.move (src, dst, copy_function = copy2) - рекурсивно переміщує файл або директорію (src) в інше місце (dst), і повертає місце призначення.

Якщо dst - існуюча директорія, то src переміщається всередину директорії. Якщо dst існує, але не директорія, то воно може бути перезаписано.

shutil.disk_usage (path) - повертає статистику використання дискового простору як namedtuple з атрибутами total, used і free, в байтах.

shutil.chown (path, user = None, group = None) - змінює власника і / або групу у файлу або директорії.

shutil.which (cmd, mode = os.F_OK | os.X_OK, path = None) - повертає шлях до виконуваного файлу по заданій команді. Якщо немає відповідності ні з одним файлом, то None. mode це права доступу, що вимагаються від файлу, за замовчуванням шукає тільки виконувані.

архівація

Високорівневі функції для створення і читання архівуються і стислих файлів. Засновані на функціях з модулів zipfile і tarfile.

shutil.make_archive (base_name, format [, root_dir [, base_dir [, verbose [, dry_run [, owner [, group [, logger]]]]]]]) - створює архів і повертає його ім'я.

base_name це ім'я файлу для створення, включаючи шлях, але не включаючи розширення (не потрібно писати ".zip" і т.д.).

format - формат архіву.

root_dir - директорія (щодо поточної), яку ми архівуємо.

base_dir - директорія, в яку буде архівувати (тобто всі файли в архіві будуть в цій папці).

Якщо dry_run = True, архів не буде створено, але операції, які повинні були бути виконані, запишуться в logger.

owner і group використовуються при створенні tar-архіву.

shutil.get_archive_formats () - список доступних форматів для архівування.

```
>>> shutil.  get_archive_formats ()
[( 'bztar', "bzip2'ed tar-file"),
 ( 'gztar', "gzip'ed tar-file"),
 ( 'tar', 'uncompressed tar file'),
 ( 'xztar', "xz'ed tar-file"),
 ( 'zip', 'ZIP file')]
```

shutil.unpack_archive (filename [, extract_dir [, format]]) - розпаковує архів. filename - повний шлях до архіву.

extract_dir - то, куди буде вилучатись вміст (за замовчуванням в поточну).

format - формат архіву (за замовчуванням намагається вгадати по розширенню файлу).

shutil.get_unpack_formats () - список доступних форматів для розархівування.

Запит розміру терміналу виведення

shutil.get_terminal_size (fallback = (columns, lines)) - повертає розмір вікна терміналу.

fallback повернеться, якщо не вдалося дізнатися розмір терміналу (термінал не підтримує такі запити, або програма працює без терміналу). За замовчуванням (80, 24).

```
>>> shutil.  get_terminal_size ()
os.terminal_size (columns = 102, lines = 29)
>>> shutil.  get_terminal_size () # Зменшили вікно
os.terminal_size (columns = 67, lines = 17)
```

ТЕМА 4: МОДУЛЬ FRACTIONS, SUBPROCESS, CMATH.

Модуль subprocess відповідає за виконання наступних дій: породження нових процесів, з'єднання с потоками стандартного введення, стандартного виводу, стандартного виводу повідомлень про помилки і отримання кодів повернення від цих процесів.

Рекомендованим підходом до роботи з підпроцесами є використання наступних допоміжних функцій для всіх випадків, де вони можуть впоратися. Для більш складних випадків може бути використаний безпосередньо інтерфейс Popen.

subprocess.call *(args, *, stdin = None, stdout = None, stderr = None, shell = False, timeout = None)* - виконує команду, описану args. Чекає завершення команди, а потім повертає код повернення.

Аргументи, наведені вище, є лише найбільш поширеними з них. Повна сигнатура функція в значній мірі така ж, як конструктор Popen. Аргумент timeout передається Popen.wait (). Якщо тайм-аут закінчується, дочірній процес буде убитий, а потім буде піднято виняток TimeoutExpired.

```
>>> subprocess.  call ([ "ls", "-l"])
0
>>> subprocess.  call ( "exit 1", shell = True)
1
```

subprocess.check_call *(args, *, stdin = None, stdout = None, stderr = None, shell = False, timeout = None)* - виконує команду, описану args. Чекає завершення команди, а потім завершується, якщо код повернення 0, або піднімає виняток CalledProcessError, об'єкт якого повертає код завершення атрибутом returncode.

```
>>> subprocess.  check_call ([ "ls", "-l"])
0
>>> subprocess.  check_call ( "exit 1", shell = True)
Traceback (most recent call last):
...
subprocess.CalledProcessError: Command 'exit 1' returned non-zero exit status 1
```

subprocess.check_output *(args, *, input = None, stdin = None, stderr = None, shell = False, universal_newlines = False, timeout = None)* - виконує команду і повертає її висновок. Піднімає виняток CalledProcessError, якщо код повернення ненульовий.

```
>>> subprocess.  check_output ([ "echo", "Hello World!"])
b'Hello World! \ n '
>>> subprocess.  check_output ([ "echo", "Hello World!"], universal_newlines = True)
'Hello World! \ N'
>>> subprocess.  check_output ([ "sed", "-e", "s / foo / bar /"],
... input = b "when in the course of fooman events \ n")
b'when in the course of barman events \ n '
>>> subprocess.  check_output ( "exit 1", shell = True)
Traceback (most recent call last):
...
subprocess.CalledProcessError: Command 'exit 1' returned non-zero exit status 1
>>> subprocess.  check_output (
... "ls non_existent_file; exit 0",
... stderr = subprocess.  STDOUT,
... shell = True)
...
'ls: non_existent_file: No such file or directory \ n'
```

Створення нових процесів і управління ними в даному модулі обробляється класом Popen. Він пропонує велику гнучкість, так що розробники можуть впоратися з менш поширеними випадками, не охопленими зручними функціями.

subprocess.DEVNULL - значення, яке може використовуватися як аргумент stdin, stdout або stderr. Чи означає, що буде використаний спеціальний файл devnull.

subprocess.PIPE - значення, яке може використовуватися як аргумент stdin, stdout або stderr. Чи означає, що для дочірнього процесу буде створено пайп.

subprocess.STDOUT - значення, яке може використовуватися як аргумент stderr. Чи означає, що потік помилок буде перенаправлений в потік виводу.

Клас **subprocess.Popen** *(args, bufsize = -1, executable = None, stdin = None, stdout = None, stderr = None, preexec_fn = None, close_fds = True, shell = False, cwd = None, env = None, universal_newlines = False, startupinfo = None, creationflags = 0, restore_signals = True, start_new_session = False, pass_fds = ())* - Виконує програму в новому процесі. args - рядок або послідовність аргументів програми. Зазвичай першим вказують виконувану програму, а потім аргументи, але також її можна вказати в параметрі executable.

Також Popen підтримує менеджери контексту :

```
with Popen ([ "ifconfig"], stdout = PIPE) as proc:
  log.  write (proc. stdout. read ())
```

Методи класу Popen:

Popen.poll () - якщо процес завершив роботу - поверне код повернення, в іншому випадку None.

Popen.wait *(timeout = None)* - очікує завершення роботи процесу і повертає код повернення. Якщо протягом timeout процес не завершився, підніметься виняток TimeoutExpired (яке можна перехопити, після чого зробити ще раз wait).

Цей метод може викликати блокування (зависання), якщо встановлено stdout = PIPE або stderr = PIPE, і дочірній процес генерує велику кількість даних в stdout і stderr. Використання communicate () дозволить уникнути цього.

Popen.communicate *(input = None, timeout = None)* - взаімодействовует з процесом: посилає дані, що містяться в input в stdin процесу, очікує завершення роботи процесу, повертає кортеж даних потоку виведення і помилок. При цьому в Popen необхідно задати значення PIPE для stdin (якщо ви хочете посилати в stdin), stdout, stderr (якщо ви хочете прочитати висновок дочірнього процесу).

Якщо протягом timeout процес не завершився, підніметься виняток TimeoutExpired (яке можна перехопити, після чого зробити ще раз communicate, або вбити дочірній процес).

Прочитані дані буферизується в пам'ять, тому не варто застосовувати цей метод у разі величезних вихідних даних.

Popen.send_signal *(signal)* - посилає сигнал *signal.*

Popen.terminate () - зупиняє дочірній процес.

Popen.kill () - вбиває дочірній процес.

Модуль fractions надає підтримку раціональних чисел.

class **fractions.Fraction** (numerator = 0, denominator = 1)

class **fractions.Fraction** (other_fraction)

class **fractions.Fraction** (float)

class **fractions.Fraction** (decimal)

class **fractions.Fraction** (string)

Клас, який представляє собою раціональні числа. Примірник класу можна створити з пари чисел (чисельник, знаменник), з іншого раціонального числа, числа з плаваючою точкою, числа типу decimal.Decimal, і з рядка, що представляє собою число.

```
>>> from fractions import Fraction
```

```
>>> Fraction (1, 3)
Fraction (1, 3)
 >>> Fraction (2, 6)
Fraction (1, 3)
 >>> Fraction (100)
Fraction (100, 1)
 >>> Fraction ()
Fraction (0, 1)
 >>> Fraction ( '3/7')
Fraction (3, 7)
 >>> Fraction ( '3/7')
Fraction (3, 7)
 >>> Fraction ( '3.1415')
Fraction (6283, 2000)
 >>> Fraction (3.1415)
Fraction (7074029114692207, 2251799813685248)
```

Необхідно зауважити, що, оскільки числа з плаваючою точкою не зовсім точні, що виходить раціональне число може відрізнятися від того, що ми хочемо отримати. Можете поділити стовпчиком 7074029114692207 на 2251799813685248 і переконатися :)

Раціональні числа можна, як int і float, складати, множити, ділити ...

```
 >>> from fractions import Fraction
>>> a = Fraction (1, 7)
>>> b = Fraction (1, 3)
>>> a + b
Fraction (10, 21)
 >>> a - b
Fraction (-4, 21)
 >>> a * b
Fraction (1, 21)
 >>> a / b
Fraction (3, 7)
 >>> a% b
Fraction (1, 7)
 >>> b% a
Fraction (1, 21)
 >>> a ** b
0.5227579585747102
 >>> abs (a - b)
Fraction (4, 21)
```

Fraction.limit_denominator (max_denominator = 1000000) - найближчим раціональне число зі знаменником не більш даного.

```
 >>> from fractions import Fraction
>>> a = Fraction (3.1415)
>>> a
Fraction (7074029114692207, 2251799813685248)
 >>> a.  limit_denominator ()
Fraction (6283, 2000)
```

Також, крім класу раціональних чисел, модуль fractions надає функцію для знаходження найбільшого загального дільника.

fractions.gcd (a, b) - найбільший спільний дільник чисел a і b.

```
 >>> from fractions import gcd
```

```
>>> gcd (1, 5)
1
 >>> gcd (1000, 3)
1
 >>> gcd (4, 6)
2
 >>> gcd (0, 2)
2
 >>> gcd (0, 0)
0
```

Модуль cmath - надає функції для роботи з комплексними числами.

cmath.phase (x) - повертає фазу комплексного числа (її ще називають аргументом). Еквівалентно math.atan2 (x.imag, x.real). Результат лежить в проміжку [-π, π].

Отримати модуль комплексного числа можна за допомогою вбудованої функції abs ().

cmath.polar (x) - перетворення до полярних координат. Повертає пару (r, phi).

cmath.rect (r, phi) - перетворення з полярних координат.

cmath.exp (x) - e^{x}.

cmath.log (x [, base]) - логарифм x за основою base. Якщо base не вказано, повертається натуральний логарифм.

cmath.log10 (x) - десятковий логарифм.

cmath.sqrt (x) - квадратний корінь з x.

cmath.acos (x) - арккосинус x.

cmath.asin (x) - арксинус x.

cmath.atan (x) - арктангенс x.

cmath.cos (x) - косинус x.

cmath.sin (x) - синус x.

cmath.tan (x) - тангенс x.

cmath.acosh (x) - гіперболічний арккосинус x.

cmath.asinh (x) - гіперболічний арксинус x.

cmath.atanh (x) - гіперболічний арктангенс x.

cmath.cosh (x) - гіперболічний косинус x.

cmath.sinh (x) - гіперболічний синус x.

cmath.tanh (x) - гіперболічний тангенс x.

cmath.isfinite (x) - True, якщо дійсна і уявна частини кінцеві.

cmath.isinf (x) - True, якщо або дійсна, або уявна частина нескінченна.

cmath.isnan (x) - True, якщо або дійсна, або уявна частина NaN.

cmath.pi - π.

cmath.e - e.

ТЕМА 5: МОДУЛІ GLOB, COPY, FUNCTOOLS, OS.PATH, JSON.

Модуль glob знаходить все шляху, що збігаються з заданим шаблоном відповідно до правил, що використовуються оболонкою Unix. Обробляються символи "*" (довільну кількість символів), "?" (Один символ), і діапазони символів за допомогою []. Для використання тильди "~" і змінних оточення необхідно використовувати os.path .expanduser () і os.path.expandvars ().

Для пошуку спецсимволов, укладайте їх в квадратні дужки. Наприклад, [?] Відповідає символу "?".

glob.glob (pathname) повернення список (можливо, порожній) шляхів, відповідних шаблоном pathname. Шлях може бути як абсолютним (наприклад, /usr/src/Python-1.5/Makefile) або відносний (як ../../Tools/*/*.gif).

glob.iglob (pathname) - повертає ітератор, що дає ті ж значення, що і glob.glob.

glob.escape (pathname) - екранує всі спеціальні символи для glob ("?", "*" і "["). Спеціальні символи в імені диску не екрануються (так як вони там не враховуються), тобто в Windows escape ("//? / C: / Quo vadis? .Txt") повертає "//? / C: / Quo vadis [?] .txt ". (нове в python 3.4).

Розглянемо, наприклад, каталог, який містить лише такі файли: 1.gif, 2.txt і card.gif. glob.glob () поверне наступні результати. Зверніть увагу, що будь-які провідні компоненти шляху зберігаються.

```
 >>> import glob
>>> glob.  glob ( './[0-9].*')
[ './1.gif', './2.txt']
 >>> glob.  glob ( '* .gif')
[ '1.gif', 'card.gif']
 >>> glob.  glob ( '? .gif')
[ '1.gif']
```

Якщо каталог містить файли, що починаються з ".", Вони не будуть включатися за замовчуванням. Розглянемо, наприклад, каталог, що містить card.gif і .card.gif:

```
 >>> import glob
>>> glob.  glob ( '* .gif')
[ 'card.gif']
 >>> glob.  glob ( '.c *')
[ '.card.gif']
```

Операція присвоювання не копіює об'єкт, він лише створює посилання на об'єкт. Для змінюваних колекцій, або для колекцій, що містять змінні елементи, часто необхідна така копія, щоб її можна було змінити, не змінюючи оригінал. Даний модуль надає загальні (поверхнева і глибока) операції копіювання.

copy.copy (x) - повертає поверхневу копію x.

copy.deepcopy (x) - повертає повну копію x.

Виняток **copy.error** - виникає, якщо об'єкт неможливо скопіювати.

Різниця між поверхневим і глибоким копіюванням істотна тільки для складових об'єктів, що містять змінні об'єкти (наприклад, список списків, або словник, як значення якого - списки або словники):

- **Поверхнева копія** створює новий складений об'єкт, і потім (у міру можливості) вставляє в нього посилання на об'єкти, що знаходяться в оригіналі.
- **Глибока копія** створює новий складений об'єкт, і потім рекурсивно вставляє в нього копії об'єктів, що знаходяться в оригіналі.
-

```
>>> import copy
>>> test_1 = [1, 2, 3, [1, 2, 3]]
>>> test_copy = copy.  copy (test_1)
>>> print (test_1, test_copy)
[1, 2, 3, [1, 2, 3]] [1, 2, 3, [1, 2, 3]]
  >>> test_copy [3].  append (4)
>>> print (test_1, test_copy)
[1, 2, 3, [1, 2, 3, 4]] [1, 2, 3, [1, 2, 3, 4]]
  >>> test_1 = [1, 2, 3, [1, 2, 3]]
>>> test_deepcopy = copy.  deepcopy (test_1)
>>> test_deepcopy [3].  append (4)
>>> print (test_1, test_deepcopy)
[1, 2, 3, [1, 2, 3]] [1, 2, 3, [1, 2, 3, 4]]
```

Для операції глибокого копіювання часто виникають дві проблеми, яких немає у операції поверхневого копіювання:

- Рекурсивні об'єкти (складові об'єкти, які явно або неявно містять посилання на себе) можуть стати причиною рекурсивного циклу;
- Оскільки глибока копія копіює все, вона може скопіювати занадто багато, наприклад, адміністративні структури даних, які повинні бути разделяемості навіть між копіями.

Функція deepcopy вирішує ці проблеми шляхом:

- Зберігання "memo" словника об'єктів, скопійованих під час поточного проходу копіювання;
- Дозволу класів, певним користувачем, перевизначати операцію копіювання або набір копіюються компонентів.

```
>>> r = [1, 2, 3]
>>> r.  append (r)
>>> print (r)
[1, 2, 3, [...]]
  >>> p = copy.  deepcopy (r)
>>> print (p)
[1, 2, 3, [...]]
```

Цей модуль не копіює типи на кшталт модулів , класів , функцій , методів, сліду в стеці, стекових кадрів, файлів , сокетов, вікон, і подібних типів.

Поверхнева копія змінюваних об'єктів також може бути створена шляхом .copy () у списків (починаючи з Python 3.3), привласненням зрізу (copied_list = original_list [:]), методом .copy () словників і множин . Створювати копію незмінних об'єктів (таких, як, наприклад, рядків) необов'язково (вони ж незмінні).

Для того, щоб визначити власну реалізацію копіювання, клас може визначити спеціальні методи __copy __ () і __deepcopy __ (). Перший викликається для реалізації операції поверхневого копіювання; додаткових аргументів не передається. Другий викликається для реалізації операції глибокого копіювання; йому передається один аргумент, словник memo. Якщо реалізація __deepcopy __ () потребує створення глибокої копії компонента, то він повинен викликати функцію deepcopy () з компонентом в якості першого аргументу і словником memo в якості другого аргументу.

Модуль functools - збірник функцій високого рівня: взаємодіючих з іншими функціями або повертають інші функції.

Модуль functools визначає наступні функції:

functools.cmp_to_key (func) - перетворює функцію порівняння в key-функцію. Використовується з інструментами, які беруть key-функції (sorted (), min (), max (), heapq.nlargest (), heapq.nsmallest (), itertools.groupby ()). Ця функція в основному використовується в якості перехідного інструменту для програм, перетворених з Python 2, які підтримували використання функцій порівняння.

Функція порівняння - функція, що приймає два аргументи, що порівнює їх і повертає негативне число, якщо перший аргумент менше, нуль, якщо дорівнює і позитивне число, якщо більше. Key-функція - функція, що приймає один аргумент і повертає інше значення, що визначає положення аргументу при сортуванні.

@ **Functools.lru_cache** (maxsize = 128, typed = False) - декоратор, який зберігає результати maxsize останніх викликів. Це може заощадити час при дорогих обчисленнях, якщо функція періодично викликається з тими ж аргументами.

Оскільки в якості кеша використовується словник, всі аргументи повинні бути хешіруемимі.

Якщо maxsize встановлений в None, кеш може зростати нескінченно. Також функція найбільш ефективна, якщо maxsize це ступінь двійки.

Якщо typed - True, аргументи функції з різними типами будуть кешуватися окремо. Наприклад, f (3) і f (3.0) будуть вважатися різними викликами, які повертають, можливо, різний результат.

Щоб допомогти виміряти ефективність кешування і відрегулювати розмір кешу, загорнута функція доповнюється функцією cache_info (), яка повертає namedtuple, що показує попадання в кеш, промахи, максимальний розмір і поточний розмір. У багатопотоковому оточенні, кількість влучень і промахів приблизно.

Також є функція cache_clear () для очищення кеша.

Оригінальна функція доступна через атрибут __wrapped__.

```
 >>> from functools import lru_cache
>>> # Числа Фібоначчі (спробуйте прибрати lru_cache) :)
...
>>> @lru_cache (maxsize = None)
... def fib (n):
... if n <2:
... return n
... return fib (n - 1) + fib (n - 2)
...
>>>
 >>> print ([fib (n) for n in range (100)])
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, ...]
 >>> print (fib. Cache_info ())
CacheInfo (hits = 196, misses = 100, maxsize = None, currsize = 100)
```

@ **Functools.total_ordering** - декоратор класу, в якому поставлено одне або більше методів порівняння. Цей декоратор автоматично додає всі інші методи. Клас повинен визначати один з методів __lt __ (), __le __ (), __gt __ (), або __ge __ (). Крім того, він повинен визначати метод __eq __ ().

наприклад:

```
@total_ordering
class Student:
   def __eq__ (self, other):
      return ((self. lastname. lower (), self. firstname. lower ()) ==
            (Other. Lastname. Lower (), other. Firstname. Lower ()))
   def __lt__ (self, other):
      return ((self. lastname. lower (), self. firstname. lower ()) <
```

```
(Other. Lastname. Lower (), other. Firstname. Lower ()))
```

functools.partial (func, * args, ** keywords) - повертає partial-об'єкт (по суті, функцію), який при виклику викликається як функція func, але додатково передають туди позиційні аргументи args, і іменовані аргументи kwargs. Якщо інші аргументи передаються при виклику функції, то позиційні додаються в кінець, а іменовані розширюють і перезаписують.

наприклад:

```
>>> from functools import partial
>>> basetwo = partial (int, base = 2)
>>> basetwo.  __doc__ = 'Convert base 2 string to an int.'
>>> print (basetwo ( '10010'))
18
```

functools.reduce (function, iterable [, initializer]) - бере два перших елемента, застосовує до них функцію, бере значення і третій елемент, і таким чином згортає iterable в одне значення. Наприклад, reduce (lambda x, y: x + y, [1, 2, 3, 4, 5]) еквівалентно ((((1 + 2) +3) +4) +5). Якщо заданий initializer, він поміщається на початку послідовності.

functools.update_wrapper (wrapper, wrapped, assigned = WRAPPER_ASSIGNMENTS, updated = WRAPPER_UPDATES) - оновлює функцію-оболонку, щоб вона стала схожою на обгорнуті функцію. assigned - кортеж, який вказує, які атрибути вихідної функції копіюються в функцію-оболонку (за замовчуванням це WRAPPER_ASSIGNMENTS (__name__, __module__, __annotations__ і __doc__)). updated - кортеж, який вказує, які атрибути оновлюються (за замовчуванням це WRAPPER_UPDATES (оновлюється __dict__ функції-оболонки)).

@ **Functools.wraps** (wrapped, assigned = WRAPPER_ASSIGNMENTS, updated = WRAPPER_UPDATES) - зручна функція для виклику partial (update_wrapper, wrapped = wrapped, assigned = assigned, updated = updated) як декоратора при визначенні функції-оболонки. наприклад:

```
>>> from functools import wraps
>>> def my_decorator (f):
... @wraps (f)
... def wrapper (* args, ** kwds):
... print ( 'Calling decorated function')
... return f (* args, ** kwds)
... return wrapper
...
>>> @my_decorator
... def example ():
... "" "Docstring" ""
... print ( 'Called example function')
...
>>> example ()
Calling decorated function
 Called example function
  >>> print (example. __Name__)
example
  >>> print (example. __Doc__)
Docstring
```

os.path є вкладеним модулем в модуль os , і реалізує деякі корисні функції для роботи з шляхами.

os.path.abspath (path) - повертає нормалізоване абсолютний шлях.

os.path.basename (path) - базове ім'я шляху (еквівалентно os.path.split (path) [1]).

os.path.commonprefix (list) - повертає найдовший префікс всіх шляхів в списку.

os.path.dirname (path) - повертає ім'я директорії шляху path.

os.path.exists (path) - повертає True, якщо path вказує на існуючий шлях або дескриптор відкритого файлу.

os.path.expanduser (path) - замінює ~ або ~ user на домашню директорію користувача.

os.path.expandvars (path) - повертає аргумент підперті змінними оточення ($ name або $ {name} замінюються змінної оточення name). Неіснуючі імена не замінює. На Windows також замінює% name%.

os.path.getatime (path) - час останнього доступу до файлу, в секундах.

os.path.getmtime (path) - час останньої зміни файлу, в секундах.

os.path.getctime (path) - час створення файлу (Windows), час останньої зміни файлу (Unix).

os.path.getsize (path) - розмір файлу в байтах.

os.path.isabs (path) - чи є шлях абсолютним.

os.path.isfile (path) - чи є шлях файлом.

os.path.isdir (path) - чи є шлях Директорією.

os.path.islink (path) - чи є шлях символічним посиланням.

os.path.ismount (path) - чи є шлях точкою монтування.

os.path.join (path1 [, path2 [...]]) - з'єднує шляху з урахуванням особливостей операційної системи.

os.path.normcase (path) - нормалізує регістр шляху (на файлових системах, які не враховують регістр, призводить шлях до нижнього регістру).

os.path.normpath (path) - нормалізує шлях, прибираючи надлишкові роздільники і посилання на попередні директорії. На Windows перетворює прямі слеші в зворотні.

os.path.realpath (path) - повертає канонічний шлях, прибираючи все символічні посилання (якщо вони підтримуються).

os.path.relpath (path, start = None) - обчислює шлях щодо директорії start (за замовчуванням - щодо поточної директорії).

os.path.samefile (path1, path2) - вказують чи path1 і path2 на один і той же файл або директорію.

os.path.sameopenfile (fp1, fp2) - вказують чи дескриптори fp1 і fp2 на один і той же відкритий файл.

os.path.split (path) - розбиває шлях на кортеж (голова, хвіст), де хвіст - останній компонент шляху, а голова - все інше. Хвіст ніколи не починається зі слеша (якщо шлях закінчується слешем, то хвіст порожній). Якщо слешів в шляху немає, то порожній буде голова.

os.path.splitdrive (path) - розбиває шлях на пару (привід, хвіст).

os.path.splitext (path) - розбиває шлях на пару (root, ext), де ext починається з точки і містить не більше однієї точки.

os.path.supports_unicode_filenames - чи підтримує файлову систему Unicode.

JSON (JavaScript Object Notation) - простий формат обміну даними, заснований на підмножині синтаксису JavaScript. Модуль json дозволяє кодувати і декодувати дані в зручному форматі.

Кодування основних об'єктів Python:

```
>>> import json
```

```
>>> json.  dumps ([ 'foo', { 'bar': ( 'baz', None, 1.0, 2)}])
'[ "foo", { "bar": [ "baz", null, 1.0, 2]}]'
  >>> print (json. Dumps ( "\" foo \ b ar "))
"\" foo \ bar "
  >>> print (json. Dumps ( '\ u1234'))
"\ u1234"
  >>> print (json. Dumps ( '\\'))
"\\"
  >>> print (json. Dumps ({ "c": 0, "b": 0, "a": 0}, sort_keys = True))
{ "a": 0, "b": 0, "c": 0}
```

Компактне кодування:

```
 >>> import json
>>> json.  dumps ([1, 2, 3, { '4': 5, '6': 7}], separators = ( ',', ':'))
'[1,2,3, { "4": 5, "6": 7}]'
```

Гарний висновок:

```
 >>> import json
>>> print (json. Dumps ({ '4': 5, '6': 7}, sort_keys = True, indent = 4))
{
    "4": 5,
    "6": 7
 }
```

Декодування (парсинг) JSON:

```
 >>> import json
>>> json.  loads ( '[ "foo", { "bar": [ "baz", null, 1.0, 2]}]')
[ 'foo', { 'bar': [ 'baz', None, 1.0, 2]}]
  >>> json.  loads ( ' "\\" foo \\ bar "')
' "foo \ x08ar'
```

Основи:

json.dump (obj, fp, skipkeys = False, ensure_ascii = True, check_circular = True, allow_nan = True, cls = None, indent = None, separators = None, default = None, sort_keys = False, ** kw) - серіалізуются obj як форматований JSON потік в fp.

Якщо skipkeys = True, то ключі Словник не базового типу (str, unicode, int, long, float, bool, None) будуть проігноровані, замість того, щоб викликати виключення TypeError.

Якщо ensure_ascii = True, все не-ASCII символи у висновку будуть екрановані послідовностями \ uXXXX, і результатом буде рядок, що містить тільки ASCII символи. Якщо ensure_ascii = False, рядки запишуться як є.

Якщо check_circular = False, то перевірка циклічних посилань буде пропущена, а такі посилання будуть викликати OverflowError.

Якщо allow_nan = False, при спробі серіалізовать значення з коми, що виходить за допустимі межі, буде викликатися ValueError (nan, inf, -inf) в суворій відповідності зі специфікацією JSON, замість того, щоб використовувати еквіваленти з JavaScript (NaN, Infinity, -Infinity).

Якщо indent є невід'ємним числом, то масиви і об'єкти в JSON будуть виводитися з цим рівнем відступу. Якщо рівень відступу 0, негативний або "", то замість цього просто використовуватимуться нові рядки. Значення за замовчуванням None відображає найбільш компактне уявлення. Якщо indent - рядок, то вона і буде використовуватися в якості відступу.

Якщо sort_keys = True, то ключі виведеного словника будуть відсортовані.

json.dumps (obj, skipkeys = False, ensure_ascii = True, check_circular = True, allow_nan = True, cls = None, indent = None, separators = None, default = None, sort_keys = False, ** kw) - серіалізуются obj в рядок JSON-формату.

Аргументи мають те ж значення, що і для dump ().

Ключі в парах ключ / значення в JSON завжди є рядками. Коли словник конвертується в JSON, все ключі словника перетворюються в рядки. В результаті цього, якщо словник спочатку перетворити в JSON, а потім назад в словник, то можна не отримати словник, ідентичний вихідному. Іншими словами, loads (dumps (x))! = X, якщо x має нестроковие ключі.

json.load (fp, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None, ** kw) - десеріалізует JSON з fp.

object_hook - опціональна функція, яка застосовується до результату декодування об'єкта (dict). Використовуватися буде значення, яке цієї функцією, а не отриманий словник.

object_pairs_hook - опціональна функція, яка застосовується до результату декодування об'єкта з певною послідовністю пар ключ / значення. Буде використаний результат, що повертається функцією, замість справжнього словника. Якщо заданий так само object_hook, то пріоритет віддається object_pairs_hook.

parse_float, якщо визначений, буде викликаний для кожного значення JSON з плаваючою точкою. За замовчуванням, це еквівалентно float (num_str).

parse_int, якщо визначений, буде викликаний для рядка JSON з числовим значенням. За замовчуванням еквівалентно int (num_str).

parse_constant, якщо визначений, буде викликаний для наступних рядків: "-Infinity", "Infinity", "NaN". Може бути використано для порушення винятків при виявленні помилкових чисел JSON.

Якщо не вдасться десеріалізовать JSON, буде порушено виняток ValueError.

json.loads (s, encoding = None, cls = None, object_hook = None, parse_float = None, parse_int = None, parse_constant = None, object_pairs_hook = None, ** kw) - десеріалізует s (екземпляр str, що містить документ JSON) в об'єкт Python.

Інші аргументи аналогічні аргументам в load ().

Кодувальники і декодувальник

Клас **json.JSONDecoder** (object_hook = None, parse_float = None, parse_int = None, parse_constant = None, strict = True, object_pairs_hook = None) - простий декодер JSON.

Виконує наступні перетворення при декодуванні:

JSON	Python
object	dict
array	list
string	str
number (int)	int
number (real)	float
true	True
false	False
null	None

Він також розуміє NaN, Infinity, і -Infinity як відповідні значення float, які знаходяться за межами специфікації JSON.

Клас **json.JSONEncoder** (skipkeys = False, ensure_ascii = True, check_circular = True, allow_nan = True, sort_keys = False, indent = None, separators = None, default = None)

Розширюваний кодировщик JSON для структур даних Python. Підтримує наступні об'єкти і типи даних за замовчуванням:

Python	**JSON**
dict	object
list, tuple	array
str	string
int, float	number
True	true
False	false
None	null

ТЕМА 6: МОДУЛІ CALENDAR, OS, PICKLE, DATETIME, BISECT.

Модуль calendar дозволяє надрукувати собі календарик (а також містить деякі інші корисні функції для роботи з календарями).

calendar.Calendar (firstweekday = 0) - клас календаря. firstweekday - перший день тижня (0 - понеділок, 6 - неділя).

методи:

iterweekdays () - итератор днів тижня, починаючи з firstweekday.

itermonthdates (year, month) - итератор для місяця month року year. Повертає всі дні цього місяця (як об'єкти datetime.date), а також дні до і після цього місяця до повного тижня.

itermonthdays2 (year, month) - як itermonthdates, тільки дні повертаються не як datetime.date об'єкти, а кортежі (номер дня, номер дня тижня).

itermonthdays (year, month) - як itermonthdates, тільки дні повертаються не як datetime.date об'єкти, а номери днів.

monthdatescalendar (year, month) - список тижнів у місяці. Тиждень - список з 7 об'єктів datetime.date.

monthdays2calendar (year, month) - як monthdatescalendar, але об'єкти - кортежі (номер дня, номер дня тижня).

monthdayscalendar (year, month) - як monthdatescalendar, але об'єкти - номери днів.

---

calendar.TextCalendar (firstweekday = 0) - клас для генерації текстового календаря.

методи:

formatmonth (theyear, themonth, w = 0, l = 0) - повертає календар на місяць у вигляді рядка, з шириною колонки w і висотою l.

prmonth (theyear, themonth, w = 0, l = 0) - друкує календар на місяць.

formatyear (theyear, w = 2, l = 1, c = 6, m = 3) - повертає календар на рік; з m колонок, шириною дати w, висотою тижні l і кількістю прогалин між місяцями c.

pryear (theyear, w = 2, l = 1, c = 6, m = 3) - друкує календар на рік.

---

calendar.HTMLCalendar (firstweekday = 0) - клас для генерації HTML календаря.

методи:

formatmonth (theyear, themonth, withyear = True) - календар на місяць у вигляді HTML таблиці. Якщо withyear True, номер року буде включений в заголовок.

formatyear (theyear, width = 3) - календар на рік у вигляді HTML таблиці. width - кількість місяців в ряду.

formatyearpage (theyear, width = 3, css = "calendar.css", encoding = None) - календар на рік у вигляді повноцінної HTML сторінки, з підключенням файлу css (який ви можете створити самі), і в кодуванні encoding.

---

calendar.LocaleTextCalendar (firstweekday = 0, locale = None) - дозволяє створити текстовий календар з назвами на рідній мові.

calendar.LocaleHTMLCalendar (firstweekday = 0, locale = None) - дозволяє створити HTML календар з назвами на рідній мові.

Наприклад, ось такий календарик вийшов у мене:

```
 import calendar
a = calendar.  LocaleHTMLCalendar (locale = 'Russian_Russia')
with open ( 'calendar.html', 'w') as g:
   print (a. formatyear (2014 року, width = 4), file = g)
```

Також модуль calendar надає кілька корисних функцій:

calendar.setfirstweekday (weekday) - встановлює перший день тижня (0 - понеділок, 6 - неділя). Також надані значення calendar.MONDAY, calendar.TUESDAY, calendar.WEDNESDAY, calendar.THURSDAY, calendar.FRIDAY, calendar.SATURDAY і calendar.SUNDAY.

calendar.firstweekday () - повертає перший день тижня.

calendar.isleap (year) - є чи рік високосним.

calendar.leapdays (y1, y2) - кількість високосних років в послідовності від y1 до y2.

calendar.weekday (year, month, day) - день тижня для цієї дати.

calendar.monthrange (year, month) - день тижня першого дня місяця і кількість днів у цьому місяці.

Модуль os надає безліч функцій для роботи з операційною системою, причому їх поведінку, як правило, не залежить від ОС, тому програми залишаються переносяться. Тут будуть приведені найбільш часто використовувані з них.

Будьте уважні: деякі функції з цього модуля підтримуються не всіма ОС.

os.name - ім'я операційної системи. Доступні варіанти: 'posix', 'nt', 'mac', 'os2', 'ce', 'java'.

os.environ - словник змінних оточення. Змінний (можна додавати і видаляти змінні оточення).

os.getlogin () - ім'я користувача, який перебуває в термінал (Unix).

os.getpid () - поточний id процесу.

os.uname () - інформація про ОС. повертає об'єкт з атрибутами: sysname - ім'я операційної системи, nodename - ім'я машини в мережі (визначається реалізацією), release - реліз, version - версія, machine - ідентифікатор машини.

os.access (path, mode, *, dir_fd = None, effective_ids = False, follow_symlinks = True) - перевірка доступу до об'єкта у поточного користувача. Прапори: **os.F_OK** - об'єкт існує, **os.R_OK** - доступний на читання, **os.W_OK** - доступний на запис, **os.X_OK** - доступний на виконання.

os.chdir (path) - зміна поточної директорії.

os.chmod (path, mode, *, dir_fd = None, follow_symlinks = True) - зміна прав доступу до об'єкта (mode - вісімкове число).

os.chown (path, uid, gid, *, dir_fd = None, follow_symlinks = True) - змінює id власника і групи (Unix).

os.getcwd () - поточна робоча директорія.

os.link (src, dst, *, src_dir_fd = None, dst_dir_fd = None, follow_symlinks = True) - створює жорстку посилання.

os.listdir (path = ".") - список файлів і директорій в папці.

os.mkdir (path, mode = 0o777, *, dir_fd = None) - створює директорію. OSError, якщо директорія існує.

os.makedirs (path, mode = 0o777, exist_ok = False) - створює директорію, створюючи при цьому проміжні директорії.

os.remove (path, *, dir_fd = None) - видаляє шлях до файлу.

os.rename (src, dst, *, src_dir_fd = None, dst_dir_fd = None) - перейменовує файл або директорію з src в dst.

os.renames (old, new) - перейменовує old в new, створюючи проміжні директорії.

os.replace (src, dst, *, src_dir_fd = None, dst_dir_fd = None) - перейменовує з src в dst з примусовою заміною.

os.rmdir (path, *, dir_fd = None) - видаляє порожню директорію.

os.removedirs (path) - видаляє директорію, потім намагається видалити батьківські директорії, і видаляє їх рекурсивно, поки вони порожні.

os.symlink (source, link_name, target_is_directory = False, *, dir_fd = None) - створює символічне посилання на об'єкт.

os.sync () - записує всі дані на диск (Unix).

os.truncate (path, length) - обрізає файл до довжини length.

os.utime (path, times = None, *, ns = None, dir_fd = None, follow_symlinks = True) - модифікація часу останнього доступу і зміни файлу. Або times - кортеж (час доступу в секундах, час зміни в секундах), або ns - кортеж (час доступу в наносекундах, час зміни в наносекундах).

os.walk (top, topdown = True, onerror = None, followlinks = False) - генерація імен файлів в дереві каталогів, зверху вниз (якщо topdown дорівнює True), або від низу до верху (якщо False). Для кожного каталогу функція walk повертає кортеж (шлях до каталогу, список каталогів, список файлів).

os.system (command) - виконує системну команду, повертає код її завершення (в разі успіху 0).

os.urandom (n) - n випадкових байт. Можливе використання цієї функції в криптографічних цілях.

os.path - модуль, який реалізує деякі корисні функції на роботи з шляхами.

Модуль pickle реалізує потужний алгоритм сериализации і десеріалізациі об'єктів Python. "Pickling" - процес перетворення об'єкта Python в потік байтів , а "unpickling" - зворотна операція, в результаті якої потік байтів перетворюється назад в Python-об'єкт. Так як потік байтів легко можна записати в файл , модуль pickle широко застосовується для збереження і завантаження складних об'єктів в Python.

Не завантажуйте за допомогою модуля pickle файли з ненадійних джерел. Це може призвести до незворотних наслідків.

Модуль pickle надає наступні функції для зручності збереження / завантаження об'єктів:

pickle.dump (obj, file, protocol = None, *, fix_imports = True) - записує серіалізовані об'єкт в файл. Додатковий аргумент protocol вказує використовуваний протокол. За замовчуванням дорівнює 3 і саме він рекомендований для використання в Python 3 (незважаючи на те, що в Python 3.4 додали протокол версії 4 з деякими оптимізаціями). У будь-якому випадку, записувати і завантажувати треба з одним і тим же протоколом.

pickle.dumps (obj, protocol = None, *, fix_imports = True) - повертає серіалізовані об'єкт. Згодом ви його можете використовувати як завгодно.

pickle.load (file, *, fix_imports = True, encoding = "ASCII", errors = "strict") - завантажує об'єкт з файлу.

pickle.loads (bytes_object, *, fix_imports = True, encoding = "ASCII", errors = "strict") - завантажує об'єкт з потоку байт.

Модуль pickle також визначає кілька винятків :

- **pickle.PickleError**
  - **pickle.PicklingError** - трапилися проблеми з сериализацией об'єкта.
  - **pickle.UnpicklingError** - трапилися проблеми з десеріалізацію об'єкта.

Цих функцій цілком достатньо для збереження і завантаження вбудованих типів даних.

>>>

```
>>> import pickle
>>> data = {
... 'a': [1, 2.0, 3, 4 + 6j],
... 'b': ( "character string", b "byte string"),
... 'c': {None, True, False}
...}
>>>
>>> with open ( 'data.pickle', 'wb') as f:
```

```
... pickle.  dump (data, f)
...
>>> with open ( 'data.pickle', 'rb') as f:
... data_new = pickle.  load (f)
...
>>> print (data_new)
{ 'c': {False, True, None}, 'a': [1, 2.0, 3, (4 + 6j)], 'b': ( 'character str
```

Модуль datetime надає класи для обробки часу і дати різними способами. Підтримується і стандартний спосіб представлення часу, проте більший акцент зроблено на простоту маніпулювання датою, часом і їх частинами.

Класи, що надаються модулем datetime:

Клас **datetime.date** (year, month, day) - стандартна дата. Атрибути: year, month, day. Незмінний об'єкт.

Клас **datetime.time** (hour = 0, minute = 0, second = 0, microsecond = 0, tzinfo = None) - стандартний час, не залежить від дати. Атрибути: hour, minute, second, microsecond, tzinfo.

Клас **datetime.timedelta** - різниця між двома моментами часу, з точністю до мікросекунд.

Клас **datetime.tzinfo** - абстрактний базовий клас для інформації про тимчасову зоні (наприклад, для обліку часового поясу і / або літнього часу).

Клас **datetime.datetime** (year, month, day, hour = 0, minute = 0, second = 0, microsecond = 0, tzinfo = None) - комбінація дати і часу.

Обов'язкові аргументи:

- datetime.MINYEAR (1) ≤ year ≤ datetime.MAXYEAR (9999)
- 1 ≤ month ≤ 12
- 1 ≤ day ≤ кількість днів у цьому місяці і році

необов'язкові:

- 0 ≤ minute <60
- 0 ≤ second <60
- 0 ≤ microsecond <1000000

Методи класу datetime:

datetime.today () - об'єкт datetime з поточної дати і часу. Працює також, як і datetime.now () зі значенням tz = None.

datetime.fromtimestamp (timestamp) - дата з стандартного уявлення часу.

datetime.fromordinal (ordinal) - дата з числа, що представляє собою кількість днів, що минули з 01.01.1970.

datetime.now (tz = None) - об'єкт datetime з поточної дати і часу.

datetime.combine (date, time) - об'єкт datetime з комбінації об'єктів date і time.

datetime.strptime (date_string, format) - перетворює рядок в datetime (так само, як і функція strptime з модуля time).

datetime.strftime (format) - див. функцію strftime з модуля time.

datetime.date () - об'єкт дати (з відсіканням часу).

datetime.time () - об'єкт часу (з відсіканням дати).

datetime.replace ([year [, month [, day [, hour [, minute [, second [, microsecond [, tzinfo]]]]]]]]) - повертає новий об'єкт datetime зі зміненими атрибутами.

datetime.timetuple () - повертає struct_time з datetime.

datetime.toordinal () - кількість днів, що минули з 01.01.1970.

datetime.timestamp () - повертає час в секундах з початку епохи.

datetime.weekday () - день тижня у вигляді числа, понеділок - 0, неділя - 6.

datetime.isoweekday () - день тижня у вигляді числа, понеділок - 1, неділя - 7.

datetime.isocalendar () - кортеж (рік в форматі ISO, ISO номер тижні, ISO день тижня).

datetime.isoformat (sep = 'T') - красива рядок виду "YYYY-MM-DDTHH: MM: SS.mmmmmm" або, якщо microsecond == 0, "YYYY-MM-DDTHH: MM: SS"

datetime.ctime () - см. ctime () з модуля time .

Приклад роботи з класом datetime:

```
>>>
```

```
 >>> from datetime import datetime, date, time
>>> # Using datetime.combine ()
>>> d = date (2005, 7, 14)
>>> t = time (12, 30)
>>> datetime.  combine (d, t)
datetime.datetime (2005, 7, 14, 12, 30)
 >>> # Using datetime.now () or datetime.utcnow ()
>>> datetime.  now ()
datetime.datetime (2007, 12, 6, 16, 29, 43, 79043) # GMT +1
 >>> datetime.  utcnow ()
datetime.datetime (2007, 12, 6, 15, 29, 43, 79060)
 >>> # Using datetime.strptime ()
>>> dt = datetime.  strptime ( "21/11/06 16:30", "% d /% m /% y% H:% M")
>>> dt
datetime.datetime (2006, 11, 21, 16, 30)
 >>> # Using datetime.timetuple () to get tuple of all attributes
>>> tt = dt.  timetuple ()
>>> for it in tt:
... print (it)
...
2006 Повідомлення: # year
 11 # month
 21 # day
 16 # hour
 30 # minute
 0 # second
 1 # weekday (0 = Monday)
 325 # number of days since 1st January
 -1 # dst - method tzinfo.dst () returned None
 >>> # Date in ISO format
>>> ic = dt.  isocalendar ()
>>> for it in ic:
... print (it)
...
2006 Повідомлення: # ISO year
 47 # ISO week
 2 # ISO weekday
 >>> # Formatting datetime
>>> dt.  strftime ( "% A,% d.% B% Y% I:% M% p")
'Tuesday, 21. November 2006 4:30 PM'
```

Модуль **bisect** - забезпечує підтримку списку в відсортованому порядку за допомогою алгоритму розподілу навпіл.

Набір функцій:

bisect.insort (list, elem), він же **bisect.insort_right (list, elem)** - вставка елемента в відсортоване список , при цьому elem розташовується якомога правіше (всі елементи, рівні йому, залишаються зліва).

bisect.insort_left (list, elem) - вставка елемента в відсортований список, при цьому elem розташовується якомога лівіше (всі елементи, рівні йому, залишаються праворуч).

bisect.bisect (list, elem), він же **bisect.bisect_right (list, elem)** - пошук місця для вставки елемента в відсортований список, таким чином, щоб elem розташовувався якомога правіше.

bisect.bisect_left (list, elem) - пошук місця для вставки елемента в відсортований список, таким чином, щоб elem розташовувався якомога лівіше.

Для повного щастя не вистачає тільки функції для перевірки наявності елемента в відсортованому списку. На щастя, це легко вирішується.

```
 >>> from bisect import bisect_left
>>> def contains (l, elem):
... index = bisect_left (l, elem)
... if index <len (l):
... return l [index] == elem
... return False
...
>>> contains (list (range (1000)), - 10)
False
 >>> testlist = (1, 2, 3, 6, 8, 10, 15)
>>> contains (testlist, 10)
True
 >>> contains (testlist, 0)
False
 >>> contains (testlist, 20)
False
```

ТЕМА 7: МОДУЛІ COLLECTIONS, ARRAY, ITERTOOLS, TIME.

Модуль **collections** - надає спеціалізовані типи даних, на основі словників , кортежів , множин , списків .

Першим розглянутим типом даних буде Counter.

collections.Counter

collections.Counter - вид словника, який дозволяє нам вважати кількість незмінних об'єктів (в більшості випадків, рядків). приклад:

```
>>> import collections
>>> c = collections.  Counter ()
>>> for word in [ 'spam', 'egg', 'spam', 'counter', 'counter', 'counter']:
... c [word] + = 1
...
>>> print (c)
Counter ({ 'counter': 3, 'spam': 2, 'egg': 1})
  >>> print (c [ 'counter'])
3
  >>> print (c [ 'collections'])
0
```

Але можливості Counter на цьому не закінчуються. У нього є кілька спеціальних методів:

elements () - повертає список елементів в лексикографічному порядку.

```
  >>> c = Counter (a = 4, b = 2, c = 0, d = - 2)
>>> list (c. Elements ())
[ 'a', 'a', 'a', 'a', 'b', 'b']
```

most_common ([n]) - повертає n найбільш часто зустрічаються елементів, в порядку убування зустрічальності. Якщо n не вказано, повертаються всі елементи.

```
  >>> Counter ( 'abracadabra').  most_common (3)
[( 'a', 5), ( 'r', 2), ( 'b', 2)]
```

subtract ([iterable-or-mapping]) - віднімання

```
  >>> c = Counter (a = 4, b = 2, c = 0, d = - 2)
>>> d = Counter (a = 1, b = 2, c = 3, d = 4)
>>> c.  subtract (d)
Counter ({ 'a': 3, 'b': 0, 'c': -3, 'd': -6})
```

Найбільш часто вживані шаблони для роботи з Counter:

- sum (c.values ()) - загальна кількість.
- c.clear () - очистити лічильник.
- list (c) - список унікальних елементів.
- set (c) - перетворити в безліч.
- dict (c) - перетворити в словник.
- c.most_common () [: - n: -1] - n найменш часто зустрічаються елементів.
- c + = Counter () - видалити елементи, що зустрічаються менш ніж один раз.

Counter також підтримує додавання, віднімання, перетин і об'єднання:

```
  >>> c = Counter (a = 3, b = 1)
>>> d = Counter (a = 1, b = 2)
>>> c + d
Counter ({ 'a': 4, 'b': 3})
  >>> c - d
```

```
Counter ({ 'a': 2})
  >>> c & d
Counter ({ 'a': 1, 'b': 1})
  >>> c |  d
Counter ({ 'a': 3, 'b': 2})
```

Наступними на черзі у нас черги (deque)

collections.deque

collections.deque (iterable, [maxlen]) - створює чергу з ітеріруємого об'єкта з максимальною довжиною maxlen. Черги дуже схожі на списки, за винятком того, що додавати і видаляти елементи можна або праворуч, або ліворуч.

Методи, певні в deque:

append (x) - додає x в кінець.

appendleft (x) - додає x в початок.

clear () - очищає чергу.

count (x) - кількість елементів, рівних x.

extend (iterable) - додає в кінець все елементи iterable.

extendleft (iterable) - додає в початок всі елементи iterable (починаючи з останнього елемента iterable).

pop () - видаляє і повертає останній елемент черги.

popleft () - видаляє і повертає перший елемент черги.

remove (value) - видаляє перше входження value.

reverse () - розгортає чергу.

rotate (n) - послідовно переносить n елементів з початку в кінець (якщо n негативно, то з кінця в початок).

collections.defaultdict

collections.defaultdict нічим не відрізняється від звичайного словника за винятком того, що за замовчуванням завжди викликається функція, яка повертає значення:

```
 >>> import collections
>>> defdict = collections.  defaultdict (list)
>>> print (defdict)
defaultdict (<class 'list'>, {})
  >>> for i in range (5):
... defdict [i].  append (i)
...
>>> print (defdict)
defaultdict (<class 'list'>, {0: [0], 1: [1], 2: [2], 3: [3], 4: [4]})
```

collections.OrderedDict

collections.OrderedDict - ще один схожий на словник об'єкт, але він пам'ятає порядок, в якому йому були дані ключі. методи:

popitem (last = True) - видаляє останній елемент якщо last = True, і перший, якщо last = False.

move_to_end (key, last = True) - додає ключ в кінець якщо last = True, і в початок, якщо last = False.

```
 >>> d = { 'banana': 3, 'apple': 4, 'pear': 1, 'orange': 2}
>>> OrderedDict (sorted (d. Items (), key = lambda t: t [0]))
OrderedDict ([( 'apple', 4), ( 'banana', 3), ( 'orange', 2), ( 'pear', 1)])
  >>> OrderedDict (sorted (d. Items (), key = lambda t: t [1]))
OrderedDict ([( 'pear', 1), ( 'orange', 2), ( 'banana', 3), ( 'apple', 4)])
  >>> OrderedDict (sorted (d. Items (), key = lambda t: len (t [0])))
```

```
OrderedDict ([( 'pear', 1), ( 'apple', 4), ( 'orange', 2), ( 'banana', 3)])
```

collections.namedtuple ()

Клас **collections.namedtuple** дозволяє створити тип даних, провідний себе як кортеж, з тим доповненням, що кожному елементу присвоюється ім'я, за яким можна в подальшому отримувати доступ:

```
 >>> Point = namedtuple ( 'Point', [ 'x', 'y'])
>>> p = Point (x = 1, y = 2)
>>> p
Point (x = 1, y = 2)
 >>> p.  x
1
 >>> p [0]
1
```

Модуль array визначає масиви в python. Масиви дуже схожі на списки , але з обмеженням на тип даних і розмір кожного елемента.

Розмір і тип елемента в масиві визначається при його створенні і може набувати таких значень:

код типу	Тип в C	Тип в python	Мінімальний розмір в байтах
'B'	signed char	int	1
'B'	unsigned char	int	1
'H'	signed short	int	2
'H'	unsigned short	int	2
'I'	signed int	int	2
'I'	unsigned int	int	2
'L'	signed long	int	4
'L'	unsigned long	int	4
'Q'	signed long long	int	8
'Q'	unsigned long long	int	8
'F'	float	float	4
'D'	double	float	8

Клас **array.array** (TypeCode [, ініціалізатор]) - новий масив, елементи якого обмежені TypeCode, і ініціалізатор, який повинен бути списком, об'єктом, який підтримує інтерфейс буфера, або ітеріруемий об'єкт.

array.typecodes - рядок, що містить всі можливі типи в масиві.

Масиви змінюються. Масиви підтримують всі спискові методи (індексація, зрізи, множення, ітерації), і інші методи.

Методи масивів (array) в python

array.typecode - TypeCode символ, використаний при створенні масиву.

array.itemsize - розмір в байтах одного елемента в масиві.

array.append (x) - додавання елемента в кінець масиву.

array.buffer_info () - кортеж (осередок пам'яті, довжина). Корисно для низькорівневих операцій.

array.byteswap () - змінити порядок проходження байтів в кожному елементі масиву. Корисно при читанні даних з файлу, написаного на машині з іншим порядком байтів.

array.count (x) - повертає кількість входжень x в масив.

array.extend (iter) - додавання елементів з об'єкта в масив.

array.frombytes (b) - робить масив array з масиву байт. Кількість байт повинна бути кратна розміру одного елемента в масиві.

array.fromfile (F, N) - читає N елементів з файлу і додає їх в кінець масиву. Файл повинен бути відкритий на бінарне читання. Якщо є менше N елементів, генерується виняток EOFError, але елементи, які були доступні, додаються в масив.

array.fromlist (список) - додавання елементів зі списку.

array.index (x) - номер першого входження x в масив.

array.insert (n, x) - включити новий пункт зі значенням x в масиві перед номером n. Негативні значення розглядаються щодо кінця масиву.

array.pop (i) - видаляє i-ий елемент з масиву і повертає його. За замовчуванням видаляється останній елемент.

array.remove (x) - видалити перше входження x з масиву.

array.reverse () - зворотний порядок елементів в масиві.

array.tobytes () - перетворення до байтам.

array.tofile (f) - запис масиву у відкритий файл.

array.tolist () - перетворення масиву в список.

Ось і все, що можна було розповісти про масиви. Вони використовуються рідко, коли потрібно досягти високої швидкості роботи. В інших випадках масиви можна замінити іншими типами даних: списками , кортежами , рядками .

Модуль itertools - збірник корисних ітераторів.

itertools.count (start = 0, step = 1) - нескінченна арифметична прогресія з першим членом start і кроком step.

itertools.cycle (iterable) - повертає по одному значенню з послідовності, повторену нескінченне число разів.

itertools.repeat (elem, n = Inf) - повторює elem n раз.

itertools.accumulate (iterable) - акумулює суми.

```
accumulate ([1, 2, 3, 4, 5]) -> 1 3 6 10 15
```

itertools.chain (* iterables) - повертає по одному елементу з першого ітератора, потім з другого, до тих пір, поки ітератори не закінчаться.

itertools.combinations (iterable, [r]) - комбінації довжиною r з iterable без повторюваних елементів.

```
combinations ( 'ABCD', 2) -> AB AC AD BC BD CD
```

itertools.combinations_with_replacement (iterable, r) - комбінації довжиною r з iterable з повторюваними елементами.

```
combinations_with_replacement ( 'ABCD', 2) -> AA AB AC AD BB BC BD CC CD DD
```

itertools.compress (data, selectors) - (d [0] if s [0]), (d [1] if s [1]), ...

```
compress ( 'ABCDEF', [1, 0, 1, 0, 1, 1]) -> A C E F
```

itertools.dropwhile (func, iterable) - елементи iterable, починаючи з першого, для якого func повернула брехня.

```
dropwhile (lambda x: x <5, [1, 4, 6, 4, 1]) -> 6 4 1
```

itertools.filterfalse (func, iterable) - все елементи, для яких func повертає брехня.

itertools.groupby (iterable, key = None) - групує елементи за значенням. Значення виходить застосуванням функції key до елементу (якщо аргумент key не вказано, то значенням є сам елемент).

```
>>> from itertools import groupby
>>> things = [( "animal", "bear"), ( "animal", "duck"), ( "plant", "cactus"),
... ( "vehicle", "speed boat"), ( "vehicle", "school bus")]
>>> for key, group in groupby (things, lambda x: x [0]):
... for thing in group:
... print ( "A% s is a% s."% (Thing [1], key))
... print ()
A bear is a animal.
A duck is a animal.

A cactus is a plant.

A speed boat is a vehicle.
A school bus is a vehicle.
```

itertools.islice (iterable [, start], stop [, step]) - итератор, що складається з зрізу.

itertools.permutations (iterable, r = None) - перестановки довжиною r з iterable.

itertools.product (* iterables, repeat = 1) - аналог вкладених циклів.

```
product ( 'ABCD', 'xy') -> Ax Ay Bx By Cx Cy Dx Dy
```

itertools.starmap (function, iterable) - застосовує функцію до кожного елементу послідовності (кожен елемент розпаковується).

```
starmap (pow, [(2, 5), (3, 2), (10, 3)]) -> 32 9 1000
```

itertools.takewhile (func, iterable) - елементи до тих пір, поки func повертає істину.

```
takewhile (lambda x: x <5, [1, 4, 6, 4, 1]) -> 1 4
```

itertools.tee (iterable, n = 2) - кортеж з n ітераторів.

itertools.zip_longest (* iterables, fillvalue = None) - як вбудована функція zip, але бере найдовший итератор, а більш короткі доповнює fillvalue.

```
zip_longest ( 'ABCD', 'xy', fillvalue = '-') -> Ax By C - D -
```

Time - модуль для роботи з часом в Python.

time.altzone - зміщення DST часового поясу в секундах на захід від нульового меридіана. Якщо часовий пояс знаходиться на схід від, зміщення негативно.

time.asctime ([t]) - перетворює кортеж або struct_time в рядок виду "Thu Sep 27 16:42:37 2012". Якщо аргумент не вказано, використовується поточний час.

time.clock () - в Unix, повертає поточний час. У Windows, повертає час, що минув з моменту першого виклику даної функції.

time.ctime ([сек]) - перетворює час, виражене в секундах з початку епохи в рядок виду "Thu Sep 27 16:42:37 2012".

time.daylight - НЕ 0, якщо визначено, зимовий час або літній (DST).

time.gmtime ([сек]) - перетворює час, виражене в секундах з початку епохи в struct_time, де DST прапор завжди дорівнює нулю.

time.localtime ([сек]) - як gmtime, але з DST прапором.

time.mktime (t) - перетворює кортеж або struct_time в число секунд з початку епохи. Протилежна функції time.localtime.

time.sleep (сек) - призупинити виконання програми на задану кількість секунд.

time.strftime (формат, [t]) - перетворює кортеж або struct_time в рядок за форматом:

формат	**значення**
% a	Скорочена назва дня тижня
% A	Повна назва дня тижня
% b	Скорочена назва місяця
% B	Повна назва місяця
% c	дата та час
% d	День місяця [01,31]
% H	Час (24-годинний формат) [00,23]
% I	Час (12-годинний формат) [01,12]
% j	День року [001,366]
% m	Номер місяця [01,12]
% M	Число хвилин [00,59]
% p	До полудня або після (при 12-годинному форматі)
% S	Число секунд [00,61]
% U	Номер тижні в році (нульова тиждень починається з неділі) [00,53]
% w	Номер дня тижня [0 (Sunday), 6]
% W	Номер тижні в році (нульова тиждень починається з понеділка) [00,53]
% x	Дата
% X	час
% y	Рік без століття [00,99]
% Y	Рік до віку
% Z	Часовий пояс
%%	Знак '%'

time.strptime (рядок [, формат]) - розбір рядка, що представляє час відповідно до формату. Значення, що повертається struct_time. Формат за замовчуванням: "% a% b% d% H:% M:% S% Y".

Клас **time.struct_time** - тип послідовності значення часу. Має інтерфейс кортежу. Можна звертатися за індексом або по імені.

1. tm_year
2. tm_mon
3. tm_mday
4. tm_hour
5. tm_min
6. tm_sec
7. tm_wday
8. tm_yday
9. tm_isdst

time.time () - час, виражене в секундах з початку епохи.

time.timezone - зміщення місцевого часового поясу в секундах на захід від нульового меридіана. Якщо часовий пояс знаходиться на схід від, зміщення негативно.

time.tzname - кортеж з двох рядків: перша - ім'я DST часового поясу, другий - ім'я місцевого часового поясу.

ТЕМА 8: ПАКЕТИ МОДУЛІВ. NUMPY: ПАКЕТ ДЛЯ РОБОТИ З ЧИСЛОВИМИ МАСИВАМИ.

NumPy - це бібліотека мови Python, що додає підтримку великих багатовимірних масивів і матриць, разом з великою бібліотекою високорівневих (і дуже швидких) математичних функцій для операцій з цими масивами.

установка NumPy

На linux - пакет python3-numpy (або аналогічний для вашої системи), або через pip . Ну або ж збирати з вихідних https://sourceforge.net/projects/numpy/files/NumPy/ .

На Windows на тому ж сайті є exe установники. Або, якщо виникають проблеми, рекомендую ще хороший збірник бібліотек https://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy .

починаємо роботу

Основним об'єктом NumPy є однорідний багатовимірний масив (в numpy називається numpy.ndarray). Це багатовимірний масив елементів (зазвичай чисел), одного типу.

Найбільш важливі атрибути об'єктів ndarray:

ndarray.ndim - число вимірювань (частіше їх називають "осі") масиву.

ndarray.shape - розміри масиву, його форма. Це кортеж натуральних чисел, що показує довжину масиву по кожній осі. Для матриці з n рядків і m стовпів, shape буде (n, m). Число елементів кортежу shape одно ndim.

ndarray.size - кількість елементів масиву. Очевидно, дорівнює добутку всіх елементів атрибута shape.

ndarray.dtype - об'єкт, що описує тип елементів масиву. Можна визначити dtype, використовуючи стандартні типи даних Python. NumPy тут надає цілий букет можливостей, як вбудованих, наприклад: bool_, character, int8, int16, int32, int64, float8, float16, float32, float64, complex64, object_, так і можливість визначити власні типи даних, в тому числі і складові.

ndarray.itemsize - розмір кожного елемента масиву в байтах.

ndarray.data - буфер, який містить фактичні елементи масиву. Зазвичай не потрібно використовувати цей атрибут, так як звертатися до елементів масиву найпростіше за допомогою індексів.

створення масивів

У NumPy існує багато способів створити масив. Один з найбільш простих - створити масив із звичайних списків або кортежів Python, використовуючи функцію numpy.array () (запам'ятайте: array - функція, що створює об'єкт типу ndarray):

```
>>> import numpy as np
>>> a = np.  array ([1, 2, 3])
>>> a
array ([1, 2, 3])
  >>> type (a)
<class 'numpy.ndarray'>
```

Функція array () трансформує вкладені послідовності в багатовимірні масиви. Тип елементів масиву залежить від типу елементів вихідної послідовності (але можна і перевизначити його в момент створення).

```
>>> b = np.  array ([[1.5, 2, 3], [4, 5, 6]])
>>> b
array ([[1.5, 2., 3.],
       [4., 5., 6.]])
```

Можна також перевизначити тип в момент створення:

```
>>> b = np.  array ([[1.5, 2, 3], [4, 5, 6]], dtype = np. complex)
>>> b
array ([[1.5 + 0.j, 2.0 + 0.j, 3.0 + 0.j],
       [4.0 + 0.j, 5.0 + 0.j, 6.0 + 0.j]])
```

Функція array () не єдина функція для створення масивів. Зазвичай елементи масиву спочатку невідомі, а масив, в якому вони будуть зберігатися, вже потрібен. Тому є кілька функцій для того, щоб створювати масиви з якимось вихідним вмістом (за замовчуванням тип створюваного масиву - float64).

Функція zeros () створює масив з нулів, а функція ones () - масив з одиниць. Обидві функції приймають кортеж з розмірами, і аргумент dtype:

```
>>> np.  zeros ((3, 5))
array ([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
 >>> np.  ones ((2, 2, 2))
array ([[[1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.]]])
```

Функція eye () створює одиничну матрицю (двовимірний масив)

```
>>> np.  eye (5)
array ([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

Функція empty () створює масив без його заповнення. Початковий вміст випадково і залежить від стану пам'яті на момент створення масиву (тобто від того сміття, що в ній зберігається):

```
>>> np.  empty ((3, 3))
array ([[6.93920488e-310, 6.93920488e-310, 6.93920149e-310],
       [6.93920058e-310, 6.93920058e-310, 6.93920058e-310],
       [6.93920359e-310, 0.00000000e +000, 6.93920501e-310]])
 >>> np.  empty ((3, 3))
array ([[6.93920488e-310, 6.93920488e-310, 6.93920147e-310],
       [6.93920149e-310, 6.93920146e-310, 6.93920359e-310],
       [6.93920359e-310, 0.00000000e +000, 3.95252517e-322]])
```

Для створення послідовностей чисел, в NumPy є функція arange (), аналогічна вбудованої в Python range (), тільки замість списків вона повертає масиви, і приймає не тільки цілі значення:

```
>>> np.  arange (10, 30, 5)
array ([10, 15, 20, 25])
 >>> np.  arange (0, 1, 0.1)
array ([0., 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

Взагалі, при використанні arange () з аргументами типу float, складно бути впевненим в тому, скільки елементів буде отримано (через обмеження точності чисел з

плаваючою комою). Тому, в таких випадках зазвичай краще використовувати функцію linspace (), яка замість кроку в якості одного з аргументів приймає число, що дорівнює кількості потрібних елементів:

```
>>> np.  linspace (0, 2, 9) # 9 чисел від 0 до 2 включно
array ([0., 0.25, 0.5, 0.75, 1., 1.25, 1.5, 1.75, 2.])
```

fromfunction (): застосовує функцію до всіх комбінацій індексів

```
>>> def f1 (i, j):
... return 3 * i + j
...
>>> np.  fromfunction (f1, (3, 4))
array ([[0., 1., 2., 3.],
       [3., 4., 5., 6.],
       [6., 7., 8., 9.]])
 >>> np.  fromfunction (f1, (3, 3))
array ([[0., 1., 2.],
       [3., 4., 5.],
       [6., 7., 8.]])
```

Друк масивів

Якщо масив занадто великий, щоб його друкувати, NumPy автоматично приховує центральну частину масиву і виводить тільки його куточки.

```
>>> print (np. Arange (0, 3000, 1))
[0 1 2 ..., 2997 2998 2999]
```

Якщо вам дійсно потрібно побачити весь масив, використовуйте функцію numpy.set_printoptions:

```
np.  set_printoptions (threshold = np. nan)
```

І взагалі, за допомогою цієї функції можна налаштувати друк масивів "під себе". Функція numpy.set_printoptions приймає кілька аргументів:

precision: кількість відображуваних цифр після коми (за замовчуванням 8).

threshold: кількість елементів в масиві, що викликає обрізання елементів (за замовчуванням 1000).

edgeitems: кількість елементів на початку і в кінці кожної розмірності масиву (за замовчуванням 3).

linewidth: кількість символів в рядку, після яких здійснюється перенесення (за замовчуванням 75).

suppress: якщо True, що не друкує маленькі значення в scientific notation (за замовчуванням False).

nanstr: строкове представлення NaN (за замовчуванням 'nan').

infstr: строкове представлення inf (за замовчуванням 'inf').

formatter: дозволяє більш тонко управляти печаткою масивів.

У минулій частині ми навчилися створювати масиви і їх друкувати. Однак це не має сенсу, якщо з ними нічого не можна робити.

Сьогодні ми познайомимося з операціями над масивами.

базові операції

Математичні операції над масивами виконуються поелементно. Створюється новий масив, який заповнюється результатами дії оператора.

```
>>> import numpy as np
>>> a = np.  array ([20, 30, 40, 50])
```

```
>>> b = np.  arange (4)
>>> a + b
array ([20, 31, 42, 53])
  >>> a - b
array ([20, 29, 38, 47])
  >>> a * b
array ([0, 30, 80, 150])
  >>> a / b # При розподілі на 0 повертається inf (нескінченність)
array ([inf, 30., 20., 16.66666667])
 <string>: 1: RuntimeWarning: divide by zero encountered in true_divide
  >>> a ** b
array ([1, 30, 1600, 125000])
  >>> a% b # При взятті залишку від ділення на 0 повертається 0
<string>: 1: RuntimeWarning: divide by zero encountered in remainder
 array ([0, 0, 0, 2])
```

Для цього, природно, масиви повинні бути однакових розмірів.

```
 >>> c = np.  array ([[1, 2, 3], [4, 5, 6]])
>>> d = np.  array ([[1, 2], [3, 4], [5, 6]])
>>> c + d
Traceback (most recent call last):
 File "<input>", line 1, in <module>
ValueError: operands could not be broadcast together with shapes (2,3) (3,2)
```

Також можна робити математичні операції між масивом і числом. В цьому випадку до кожного елементу додається (або що ви там робите) це число.

```
 >>> a + 1
array ([21, 31, 41, 51])
  >>> a ** 3
array ([8000, 27000, 64000, 125000])
  >>> a <35 # І фільтрацію можна проводити
array ([True, True, False, False], dtype = bool)
```

NumPy також надає безліч математичних операцій для обробки масивів:

```
 >>> np.  cos (a)
array ([0.40808206, 0.15425145, -0.66693806, 0.96496603])
  >>> np.  arctan (a)
array ([1.52083793, 1.53747533, 1.54580153, 1.55079899])
  >>> np.  sinh (a)
array ([2.42582598e + 08, 5.34323729e + 12, 1.17692633e + 17,
        2.59235276e + 21])
```

Повний список можна подивитися тут .

Багато унарні операції, такі як, наприклад, обчислення суми всіх елементів масиву, представлені також і у вигляді методів класу ndarray.

```
 >>> a = np.  array ([[1, 2, 3], [4, 5, 6]])
>>> np.  sum (a)
21
  >>> a.  sum ()
21
  >>> a.  min ()
1
```

```
>>> a.  max ()
6
```

За замовчуванням, ці операції застосовуються до масиву, як якщо б він був списком чисел, незалежно від його форми. Однак, вказавши параметр axis, можна застосувати операцію для зазначеної осі масиву:

```
>>> a.  min (axis = 0) # Найменше число в кожному стовпці
array ([1, 2, 3])
>>> a.  min (axis = 1) # Найменше число в кожному рядку
array ([1, 4])
```

Індекси, зрізи, ітерації

Одномірні масиви здійснюють операції індексування, зрізів і ітерацій дуже схожим чином з звичайними списками і іншими послідовностями Python (хіба що видаляти за допомогою зрізів можна).

```
>>> a = np.  arange (10) ** 3
>>> a
array ([0, 1, 8, 27, 64, 125, 216, 343, 512, 729])
>>> a [1]
1
>>> a [3: 7]
array ([27, 64, 125, 216])
>>> a [3: 7] = 8
>>> a
array ([0, 1, 8, 8, 8, 8, 8, 343, 512, 729])
>>> a [:: - 1]
array ([729, 512, 343, 8, 8, 8, 8, 8, 1, 0])
>>> del a [4: 6]
Traceback (most recent call last):
 File "<input>", line 1, in <module>
ValueError: can not delete array elements
>>> for i in a:
... print (i ** (1/3))
...
0.0
 1.0
 2.0
 2.0
 2.0
 2.0
 2.0
 7.0
 8.0
 9.0
```

У багатовимірних масивів на кожну вісь припадає один індекс. Індекси передаються у вигляді послідовності чисел, розділених комами (чи то пак, кортежами):

```
>>> b = np.  array ([[0, 1, 2, 3],
... [10, 11, 12, 13],
... [20, 21, 22, 23],
... [30, 31, 32, 33],
... [40, 41, 42, 43]])
```

```
...
>>> b [2, 3] # Другий рядок, третій стовпець
23
 >>> b [(2, 3)]
23
 >>> b [2] [3] # Можна і так
23
 >>> b [:, 2] # Третій стовпець
array ([2, 12, 22, 32, 42])
 >>> b [: 2] # Перші два рядки
array ([[0, 1, 2, 3],
       [10, 11, 12, 13]])
 >>> b [1: 3,::] # Друга і третя рядки
array ([[10, 11, 12, 13],
       [20, 21, 22, 23]])
```

Коли індексів менше, ніж осей, відсутні індекси передбачаються доповненими за допомогою зрізів:

```
 >>> b [- 1] # Останній рядок.  Еквівалентно b [-1 ,:]
array ([40, 41, 42, 43])
```

b [i] можна читати як b [i, <стільки символів ':', скільки потрібно>]. У NumPy це також може бути записано за допомогою точок, як b [i, ...].

Наприклад, якщо x має ранг 5 (тобто у нього 5 осей), тоді

- x [1, 2, ...] еквівалентно x [1, 2,:,:,:],
- x [..., 3] те ж саме, що x [:,:,:,:, 3] і
- x [4, ..., 5,:] це x [4,:,:, 5,:].

```
 >>> a = np.  array (([[0, 1, 2], [10, 12, 13]], [[100, 101, 102], [110, 112, 113]]))
>>> a.  shape
(2, 2, 3)
 >>> a [1, ...] # те саме, що a [1,:,:] або a [1]
array ([[100, 101, 102],
       [110, 112, 113]])
 >>> c [..., 2] # те саме, що a [:,:, 2]
array ([[2, 13],
       [102, 113]])
```

Ітерірованіе багатовимірних масивів починається з першої осі:

```
 >>> for row in a:
... print (row)
...
[[0 1 2]
  [10 12 13]]
 [[100 101 102]
  [110 112 113]]
```

Однак, якщо потрібно перебрати поелементно весь масив, як якщо б він був одновимірним, для цього можна використовувати атрибут flat:

```
 >>> for el in a.  flat:
... print (el)
...
0
```

```
1
2
10
12
13
100
101
102
110
112
113
```

Маніпуляції з формою

Як вже говорилося, у масиву є форма (shape), яка визначається числом елементів вздовж кожної осі:

```
>>> a
array ([[[0, 1, 2],
        [10, 12, 13]],

       [[100, 101, 102],
        [110, 112, 113]]])
 >>> a.  shape
(2, 2, 3)
```

Форма масиву може бути змінена за допомогою різних команд:

```
 >>> a.  ravel () # Робить масив плоским
array ([0, 1, 2, 10, 12, 13, 100, 101, 102, 110, 112, 113])
 >>> a.  shape = (6, 2) # Зміна форми
>>> a
array ([[0, 1],
       [2, 10],
       [12, 13],
       [100, 101],
       [102, 110],
       [112, 113]])
 >>> a.  transpose () # Транспонування
array ([[0, 2, 12, 100, 102, 112],
       [1, 10, 13, 101, 110, 113]])
 >>> a.  reshape ((3, 4)) # Зміна форми
array ([[0, 1, 2, 10],
       [12, 13, 100, 101],
       [102, 110, 112, 113]])
```

Порядок елементів у масиві в результаті функції ravel () відповідає звичайному "C-стилю", тобто, чим правіше індекс, тим він "швидше змінюється": за елементом a [0,0] слід a [0,1]. Якщо одна форма масиву була змінена на іншу, масив переформовувалися також в "C-стилі". Функції ravel () і reshape () також можуть працювати (при використанні додаткового аргументу) в FORTRAN-стилі, в якому швидше змінюється більш лівий індекс.

```
>>> a
array ([[0, 1],
       [2, 10],
```

```
        [12, 13],
        [100, 101],
        [102, 110],
        [112, 113]])
 >>> a.  reshape ((3, 4), order = 'F')
array ([[0, 100, 1, 101],
        [2, 102, 10, 110],
        [12, 112, 13, 113]])
```

Метод reshape () повертає її аргумент зі зміненою формою, в той час як метод resize () змінює сам масив:

```
 >>> a.  resize ((2, 6))
>>> a
array ([[0, 1, 2, 10, 12, 13],
        [100, 101, 102, 110, 112, 113]])
```

Якщо при операції такої перебудови один з аргументів задається як -1, то він автоматично розраховується відповідно до іншими заданими:

```
 >>> a.  reshape ((3, - 1))
array ([[0, 1, 2, 10],
        [12, 13, 100, 101],
        [102, 110, 112, 113]])
```

об'єднання масивів

Кілька масивів можуть бути об'єднані разом уздовж різних осей за допомогою функцій hstack і vstack.

hstack () об'єднує масиви за першими осях, vstack () - за останніми:

```
 >>> a = np.  array ([[1, 2], [3, 4]])
>>> b = np.  array ([[5, 6], [7, 8]])
>>> np.  vstack ((a, b))
array ([[1, 2],
        [3, 4],
        [5, 6],
        [7, 8]])
 >>> np.  hstack ((a, b))
array ([[1, 2, 5, 6],
        [3, 4, 7, 8]])
```

Функція column_stack () об'єднує одномірні масиви як стовпців двовимірного масиву:

```
 >>> np.  column_stack ((a, b))
array ([[1, 2, 5, 6],
        [3, 4, 7, 8]])
```

Аналогічно для рядків є функція row_stack ().

```
 >>> np.  row_stack ((a, b)) array ([[1, 2], [3, 4], [5, 6], [7, 8]])
```

розбиття масиву

Використовуючи hsplit () ви можете розбити масив вздовж горизонтальної осі, вказавши або число повертаються масивів однакової форми, або номера стовпців, після яких масив розрізається "ножицями":

```
 >>> a = np.  arange (12).  reshape ((2, 6))
```

```
>>> a
array ([[0, 1, 2, 3, 4, 5],
       [6, 7, 8, 9, 10, 11]])
 >>> np.  hsplit (a, 3) # Розбити на 3 частини
[array ([[0, 1], [6, 7]]),
 array ([[2, 3], [8, 9]]),
 array ([[4, 5], [10, 11]])]
 >>> np.  hsplit (a, (3, 4)) # Розрізати а після третього і четвертого стовпця
[array ([[0, 1, 2], [6, 7, 8]]),
 array ([[3], [9]]),
 array ([[4, 5], [10, 11]])]
```

Функція vsplit () розбиває масив вздовж вертикальної осі, а array_split () дозволяє вказати осі, уздовж яких відбудеться розбиття.

Копії та подання

При роботі з масивами, їх дані іноді необхідно копіювати в інший масив, а іноді немає. Це часто є джерелом плутанини. Можливо 3 випадки:

Взагалі ніяких копій

Просте присвоювання не створює ні копії масиву, ні копії його даних:

```
 >>> a = np.  arange (12)
>>> b = a # Нового об'єкта створено не було
>>> b is a # a і b це два імені для одного і того ж об'єкта ndarray
True
 >>> b.  shape = (3, 4) # змінить форму а
>>> a.  shape
(3, 4)
```

Python передає змінювані об'єкти як посилання, тому виклики функцій також не створюють копій.

Подання або поверхнева копія

Різні об'єкти масивів можуть використовувати одні і ті ж дані. Метод view () створює новий об'єкт масиву, що є поданням тих же даних.

```
 >>> c = a.  view ()
>>> c is a
False
 >>> c.  base is a # с це представлення даних, що належать а
True
 >>> c.  flags.  owndata
False
 >>>
 >>> c.  shape = (2, 6) # форма а не зміниться
>>> a.  shape
(3, 4)
 >>> c [0, 4] = 1234 # дані а зміняться
>>> a
array ([[0, 1, 2, 3],
       [1234, 5, 6, 7],
       [8, 9, 10, 11]])
```

Зріз масиву це уявлення:

```
 >>> s = a [:, 1: 3]
>>> s [:] = 10
```

```
>>> a
array ([[0, 10, 10, 3],
        [1234, 10, 10, 7],
        [8, 10, 10, 11]])
```

глибока копія

Метод copy () створить справжню копію масиву і його даних:

```
 >>> d = a.  copy () # створюється новий об'єкт масиву з новими даними
>>> d is a
False
  >>> d.  base is a # d не має нічого спільного з a
False
  >>> d [0, 0] = 9999
>>> a
array ([[0, 10, 10, 3],
        [1234, 10, 10, 7],
        [8, 10, 10, 11]])
```

ТЕМА 9: АНАЛІЗ ДАНИХ ЗА ДОПОМОГОЮ PANDAS. РОБОТА З ДАНИМИ З БАЗИ ДАНИХ SQL.

`pandas` - це Python бібліотека для аналізу і обробки даних. Вона дійсно швидка і дозволяє вам легко досліджувати дані.

Мета цього циклу статей - дати конкретні приклади використання pandas.

Швидкий тур в IPython Notebook

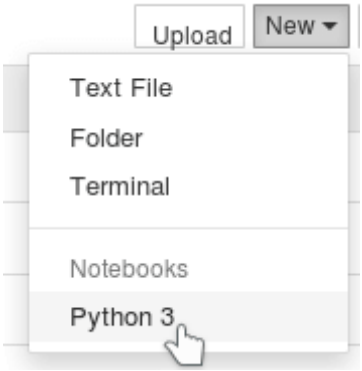
Буде корисним повторити це в інтерактивному режимі, тому краще встановити ipython за допомогою pip:

```
sudo pip3 install jupyter pandas matplotlib
```

Після чого можна починати розмови ipython просто написавши:

```
jupyter notebook
```

Після цього в браузері відкриється сервер jupyter. Створимо Python 3 notebook.



По-перше, запустимо код з осередку.

In [1]:

```
import pandas as pd

print ( "Hi! This is a cell. Press the ▸ button above to run it")
```

Hi! This is a cell. Press the ▸ button above to run it

Крім цієї кнопки, можна запускати код з комірки за допомогою Ctrl + Enter.

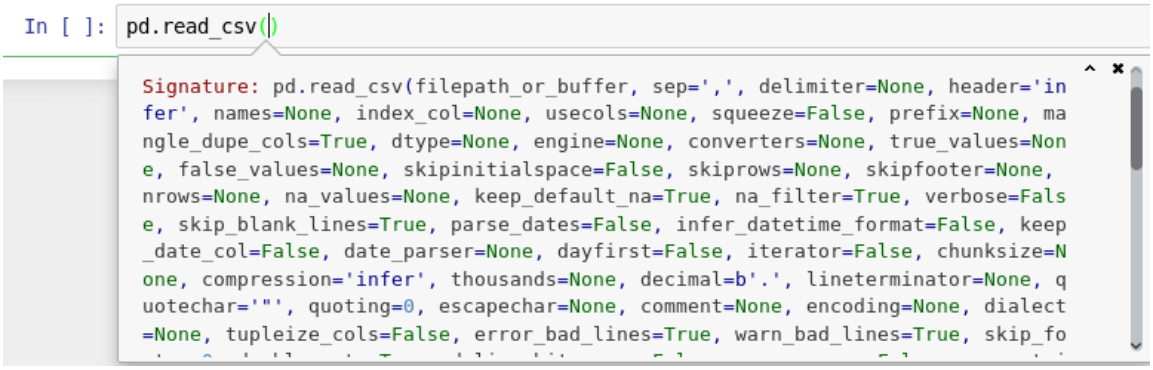
Одна з найбільш корисних речей в IPython notebook це автодоповнення.

Спробуйте наступне: натисніть в осередку відразу після read_csv (і натисніть Shift + Tab 4 рази, повільно. Подивіться, що вийде.

In []:

```
pd.  read_csv (
```

Ось що виходить після 2 разів:

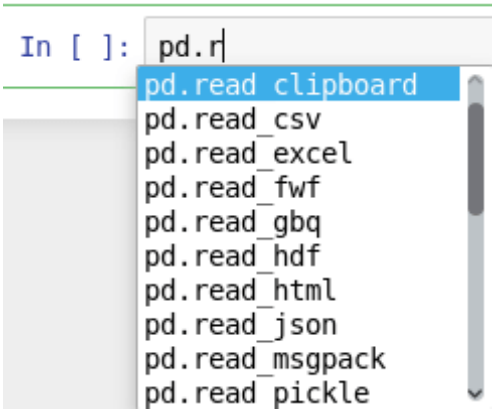


Добре, тепер спробуємо автодоповнення. Введіть pd.r (перша буква функції) і подивіться, які варіанти пропонуються.

In []:

```
pd.  r
```

Ви повинні побачити наступне:



написання коду

Написання коду в осередках абсолютно природно.

In [2]:

```
def print_10_nums ():
  for i in range (10):
    print (i, end = '')
```

In [3]:

```
print_10_nums ()
0 1 2 3 4 5 6 7 8 9
```

магічні функції

IPython має безліч магічних функцій . Далі йде приклад порівняння `sum()` з генератором списку і за допомогою ітератора, використовуючи магічну функцію `%time` .

In [4]:

```
% Time sum ([x for x in range (100000)])
CPU times: user 20 ms, sys: 0 ns, total: 20 ms
Wall time: 18.5 ms
```

Out [4]:

```
4999950000
```

In [5]:

```
% Time sum (x for x in range (100000))
CPU times: user 8 ms, sys: 0 ns, total: 8 ms
Wall time: 8.34 ms
```

Out [5]:

```
4999950000
```

Ця частина показує спосіб обробки даних, що зберігаються в форматі csv, а також побудова найпростіших графіків.

Необхідні обсяги імпорту та налаштування

In [1]:

```
# Малювати графіки відразу ж
% Matplotlib inline
```

```
import pandas as pd
import matplotlib.pyplot as plt

plt.  style.  use ( 'ggplot') # Красиві графіки
plt.  rcParams [ 'figure.figsize'] = (15, 5) # Розмір картинок
```

Читання з csv файлу

Можна читати дані з CSV файлу за допомогою функції read_csv . За умовчанням передбачається, що поля розділені комами.

Ми розглянемо деякі дані про велосипедистів Монреаля. Завантажити дані звідси .

Цей набір даних описує, скільки людей знаходилося на 7 різних велосипедних доріжках Монреаля, кожен день.

Просто взяти і прочитати за допомогою read_csv не вийде, потрібно задати аргументи, які зроблять наступне:

- Змінять роздільник на ;
- Змінять кодування на 'latin1' (за замовчуванням вважається 'utf8')
- Оброблять дати в колонці 'Date'
- Скажуть, що спочатку йде день, а потім місяць (формат YYYY-DD-MM)
- Змінять індекс на значення в колонці 'Date'

In [2]:

```
fixed_df = pd.  read_csv ( 'data / bikes.csv', # Це те, куди ви завантажили файл
                sep = ';'  , Encoding = 'latin1',
                parse_dates = [ 'Date'], dayfirst = True,
                index_col = 'Date')
fixed_df [: 3]
```

Out [2]:

	Berri 1	Brébeuf (données non disponibles)	Côte-Sainte-Catherine	Maisonneuve 1	Maisonneuve 2	du Parc	Pierre-Dupuy	Rachel1	St-Urbain (données non disponibles)
Date									
2012-01-01	35	NaN	0	38	51	26	10	16	NaN
2012-01-02	83	NaN	1	68	153	53	6	43	NaN
2012-01-03	135	NaN	2	104	248	89	3	58	NaN

вибір колонок

Коли ви обробляєте CSV за допомогою pandas, ви отримуєте об'єкт під назвою DataFrame , який складається з рядків і стовпців. Ви можете отримувати стовпці таким же чином, яким отримуєте елементи словника.

наприклад:

In [3]:

```
fixed_df [ 'Berri 1'] [: 10]
```

Out [3]:

```
Date
2012-01-01 35
2012-01-02 83
2012-01-03 135
2012-01-04 144
2012-01-05 197
2012-01-06 146
2012-01-07 98
2012-01-08 95
2012-01-09 244
2012-01-10 397
Name: Berri 1, dtype: int64
```

графіки

Просто додайте .plot() в кінець! Що може бути простіше цього? =)

Ми можемо бачити (сюрприз!), Що не так багато людей катаються на велосипеді в січні, лютому і в березні.

In [4]:

```
fixed_df [ 'Berri 1'].  plot ()
```

Out [4]:

Ми також можемо побудувати графік для всіх колонок. Також ми зробимо картинку трохи більше.

Ми бачимо, що все велосипедні доріжки ведуть себе в основному однаково - якщо це поганий день для велосипедистів, то це поганий день всюди.

In [5]:

```
fixed_df.  plot (figsize = (15, 10))
```

Out [5]:

До цього моменту, ми отримували дані тільки з csv файлів. Це досить поширений спосіб збереження даних, але далеко не єдиний! Pandas може працювати з даними з HTML, JSON, SQL, Excel (!!!), HDF5, Stata, і деяких інших речей. У цій частині ми поговоримо про роботу з даними з баз даних SQL.

In [1]:

```
import pandas as pd
import sqlite3
```

Читання з SQL баз даних

Завантажити дані з SQL бази можна за допомогою функції `pd.read_sql` . `read_sql` автоматично перетворює стовпці SQL в стовпці DataFrame.

`read_sql` приймає 2 аргументи: запит `SELECT` , і connection. Це здорово, тому що це означає, що можна читати з *будь-якого* виду бази даних - неважливо, MySQL, SQLite, PostgreSQL, або інша.

У цьому прикладі ми читаємо з бази SQLite, але інші читаються так само. [Файл, з яким ми будемо працювати] .

In [2]:

```
con = sqlite3.  connect ( "data / weather_2012.sqlite")
df = pd.  read_sql ( "SELECT * from weather_2012 LIMIT 3", con)
df
```

Out [2]:

	id	date_time	temp
0	1	2012-01-01 00:00:00	-1.8
1	2	2012-01-01 1:00:00	-1.8
2	3	2012-01-01 2:00:00	-1.8

`read_sql` не встановлює первинний ключ (`id`) в якості індексу. Можна це зробити вручну, додавши аргумент `index_col` до `read_sql` .

Якщо ви багато використовували `read_csv` , ви могли помітити, що у нього також є аргумент `index_col` . І веде він себе точно так само.

In [3]:

```
df = pd.  read_sql ( "SELECT * from weather_2012 LIMIT 3", con, index_col = 'id')
df
```

Out [3]:

	date_time	temp
id		
1	2012-01-01 00:00:00	-1.8
2	2012-01-01 1:00:00	-1.8
3	2012-01-01 2:00:00	-1.8

Якщо ви хочете, щоб dataframe був індексований декількома стовпцями, в `index_col` можна вказати їх список:

In [4]:

```
df = pd.  read_sql ( "SELECT * from weather_2012 LIMIT 3", con,
            index_col = [ 'id', 'date_time'])
df
```

Out [4]:

		temp
id	**date_time**	
1	**2012-01-01 00:00:00**	-1.8

		temp
id	**date_time**	
2	**2012-01-01 1:00:00**	-1.8
3	**2012-01-01 2:00:00**	-1.8

Запис в базу

Запис проводиться за допомогою методу `to_sql` (по аналогії з CSV):

In [5]:

```
weather_df = pd.  read_csv ( 'data / weather_2012.csv')
con = sqlite3.  connect ( "data / test_db.sqlite")
con.  execute ( "DROP TABLE IF EXISTS weather_2012")
weather_df.  to_sql ( "weather_2012", con)
/usr/local/lib/python3.5/dist-packages/pandas/core/generic.py:1345: UserWarning: The spaces in these column names will not be changed.  In pandas versions <0.14, spaces were converted to underscores.
chunksize = chunksize, dtype = dtype)
```

Тепер ми можемо завантажити записані дані:

In [6]:

```
con = sqlite3.  connect ( "data / test_db.sqlite")
df = pd.  read_sql ( "SELECT * from weather_2012 LIMIT 3", con)
df
```

Out [6]:

	index	Date / Time	Temp (C)	Dew Point Temp (C)	Rel Hum (%)	Wind Spd (km / h)	Visibility (km)	Stn Press (kPa)	Weather
0	0	2012-01-01 00:00:00	-1.8	-3.9	86	4	8.0	101.24	Fog
1	1	2012-01-01 1:00:00	-1.8	-3.7	87	4	8.0	101.24	Fog
2	2	2012-01-01 2:00:00	-1.8	-3.4	89	7	4.0	101.26	Freezing Drizzle, Fog

Головна перевага зберігання даних в базі в тому, що можна безпосередньо робити SQL запити. Особливо гостро це, якщо SQL для вас більш рідна мова. Наприклад, можна впорядкувати за колонці 'Weather' за допомогою лише SQL:

In [7]:

```
con = sqlite3.  connect ( "data / test_db.sqlite")
df = pd.  read_sql ( "SELECT * from weather_2012 ORDER BY Weather LIMIT 3", con)
df
```

Out [7]:

	index	Date / Time	Temp (C)	Dew Point Temp (C)	Rel Hum (%)	Wind Spd (km / h)	Visibility (km)	Stn Press (kPa)	Weather
0	67	2012-01-03 19:00:00	-16.9	-24.8	50	24	25.0	101.74	Clear
1	114	2012-01-05 18:00:00	-7.1	-14.4	56	11	25.0	100.71	Clear
2	115	2012-01-05 19:00:00	-9.2	-15.4	61	7	25.0	100.80	Clear

Інші бази даних

Для підключення до MySQL:

Щоб це працювало, у вас на машині повинна бути встановлена відповідна база даних

In []:

```
import MySQLdb
con = MySQLdb.  connect (host = "localhost", db = "test")
```

PostgreSQL:

In []:

```
import psycopg2
con = psycopg2.  connect (host = "localhost")
```

ТЕМА 10: ОБ'ЄДНАННЯ І ГРУПУВАННЯ ДАНИХ.

Ця частина показує способи угруповання, об'єднання і доповнення даних.

In [1]:

```
% Matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt

plt. style. use ( 'ggplot')

plt. rcParams [ 'figure.figsize'] = (10, 5)
```

Повернемося до нашого набору даних про велосипедистів. Припустимо, я живу в Монреалі, і мені цікаво, чи використовується велосипед для приміських поїздок, або для розваги - люди більше катаються на велосипеді в вихідні дні або в будні?

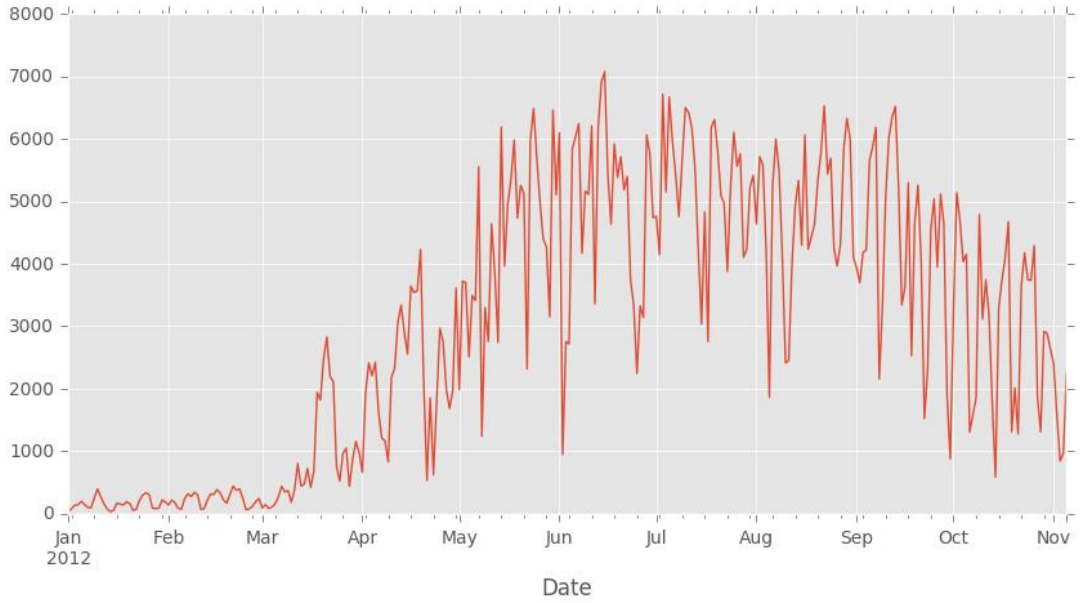
Додаємо стовпець "день тижня"

завантажимо дані

In [2]:

```
bikes = pd. read_csv ( 'data / bikes.csv', sep = ';', encoding = 'latin1', parse_dates = [ 'Date'], dayfirst = True, index_col = 'Date')
bikes [ 'Berri 1']. plot ()
```

Out [2]:



Подивимося на велодоріжку Berri. Це вулиця в Монреалі, з досить важливою велодоріжкою.

Створимо dataframe тільки з велодоріжкою Berri.

In [3]:

```
berri_bikes = bikes [[ 'Berri 1']]. copy ()
```

In [4]:

```
berri_bikes [: 5]
```

Out [4]:

	Berri 1

Date	
2012-01-01	35
2012-01-02	83
2012-01-03	135
2012-01-04	144
2012-01-05	197

Далі, потрібно додати колонку "день тижня". По-перше, ми отримаємо його з першого стовпчика (індекс). Ми не говорили про індекси раніше, але індекс - це те, що знаходиться лівіше за все dataframe, під 'Date'. Зараз це все дні в році.

In [5]:

```
berri_bikes.  index
```

Out [5]:

```
DatetimeIndex ([ '2012-01-01', '2012-01-02', '2012-01-03', '2012-01-04',
                 '2012-01-05', '2012-01-06', '2012-01-07', '2012-01-08',
                 '2012-01-09', '2012-01-10',
                 ...
                 '2012-10-27', '2012-10-28', '2012-10-29', '2012-10-30',
                 '2012-10-31', '2012-11-01', '2012-11-02', '2012-11-03',
                 '2012-11-04', '2012-11-05'],
                dtype = 'datetime64 [ns]', name = 'Date', length = 310, freq = None)
```

Деякі дні пропущені - тут тільки 310 днів.

Pandas має набір функціоналу для роботи з проміжками часу, тому якщо ми, наприклад, хочемо отримати день місяця для кожного рядка, то ми можемо написати:

In [6]:

```
berri_bikes.  index.  day
```

Out [6]:

```
array ([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 1, 2, 3,
        4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
       21, 22, 23, 24, 25, 26, 27, 28, 29, 1, 2, 3, 4, 5, 6, 7, 8,
        9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
       26, 27, 28, 29, 30, 31, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
       12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
       29, 30, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
       16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 1,
        2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
       19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 1, 2, 3, 4, 5,
        6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
       23, 24, 25, 26, 27, 28, 29, 30, 31, 1, 2, 3, 4, 5, 6, 7, 8,
        9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
       26, 27, 28, 29, 30, 31, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
       12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
       29, 30, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
```

```
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 1,
 2, 3, 4, 5], dtype = int32)
```

Ми хочемо день тижня, так що:

In [7]:

```
berri_bikes.  index.  weekday
```

Out [7]:

```
array ([6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0,
    1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2,
    3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4,
    5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6,
    0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1,
    2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3,
    4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5,
    6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0,
    1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2,
    3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4,
    5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6,
    0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1,
    2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3,
    4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0], dtype = int32)
```

Це дні тижня, 0 - понеділок. Тепер, коли ми знаємо, як *отримати* день тижня, ми можемо додати його як стовпець в dataframe.

In [8]:

```
berri_bikes.  loc [:, 'weekday'] = berri_bikes.  index.  weekday
berri_bikes [: 5]
```

Out [8]:

	Berri 1	**weekday**
Date		
2012-01-01	35	6
2012-01-02	83	0
2012-01-03	135	1
2012-01-04	144	2
2012-01-05	197	3

додаємо велосипедистів

Це дуже просто! Dataframe має метод `.groupby()` , який групує по одному або кількох стовпців. Детальніше можна прочитати в документації .

У нашому випадку, `berri_bikes.groupby('weekday').aggregate(sum)` означає "Згрупувати рядки за днями тижня і потім скласти всі значення з однаковим днем тижня".

In [9]:

```
weekday_counts = berri_bikes.  groupby ( 'weekday').  aggregate (sum)
```

```
# Weekday_counts = berri_bikes.groupby ( 'weekday'). Sum () - можна і так.  Навіть простіше.
weekday_counts
```

Out [9]:

	Berri 1
weekday	
0	134298
1	135305
2	сто п'ятьдесят дві тисячі дев'ятсот сімдесят дві
3	160131
4	сто сорок одна тисяча сімсот сімдесят одна
5	101578
6	99310

Тепер перейменуємо 0, 1, 2, 3, 4, 5, 6, щоб розуміти, що вони означають:

In [10]:

```
weekday_counts.  index = [ 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
weekday_counts
```

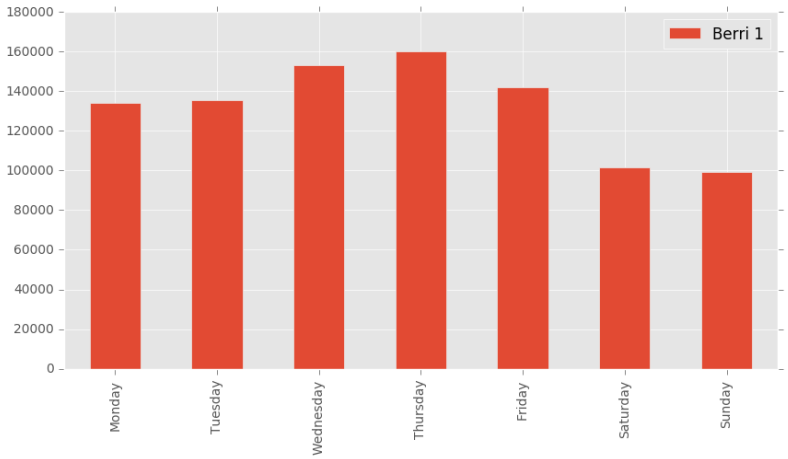
Out [10]:

	Berri 1
Monday	134298
Tuesday	135305
Wednesday	сто п'ятьдесят дві тисячі дев'ятсот сімдесят дві
Thursday	160131
Friday	сто сорок одна тисяча сімсот сімдесят одна
Saturday	101578
Sunday	99310

In [11]:

```
weekday_counts.  plot (kind = 'bar')
```

Out [11]:



У Монреалі частіше катаються по буднях - здорово!

з'єднуємо разом

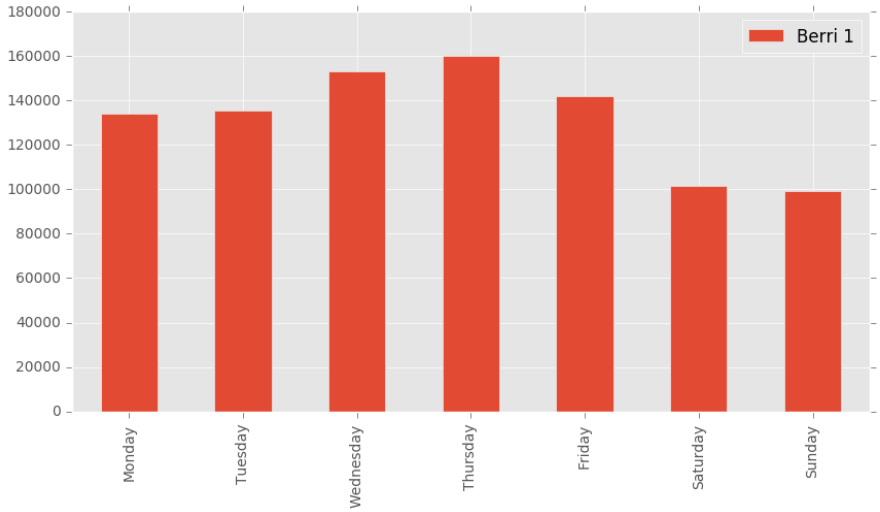
З'єднаємо всі разом. Всього 6 рядків коду!

Якщо хочете погратися, спробуйте поміняти `sum` на `max` , `numpy.median` , або будь-яку іншу функцію на ваш вибір.

In [12]:

```
bikes = pd.  read_csv ( 'data / bikes.csv',
                sep = ';'  , Encoding = 'latin1',
                parse_dates = [ 'Date'], dayfirst = True,
                index_col = 'Date')
 # Add the weekday column
 berri_bikes = bikes [[ 'Berri 1']].  copy ()
 berri_bikes.  loc [:, 'weekday'] = berri_bikes.  index.  weekday

 # Add up the number of cyclists by weekday, and plot!
 weekday_counts = berri_bikes.  groupby ( 'weekday').  aggregate (sum)
 weekday_counts.   index  =  [  'Monday',  'Tuesday',  'Wednesday',  'Thursday',  'Friday',
'Saturday', 'Sunday']
 weekday_counts.  plot (kind = 'bar')
```

Out [12]:



ТЕМА 11: ОСНОВИ MATPLOTLIB.

Після установки бібліотеки matplotlib викликається в консолі або в скрипті як модуль:

```
import matplotlib as mpl

# Висновок на екран поточної версії бібліотеки matplotlib
print ( 'Current version on matplotlib library is', mpl. __version__)
 ( 'Current version on matplotlib library is', '2.0.0')
```

Matplotlib Відбудеться з безлічі модулів. Модулі наповнені різними класами і функціями, які ієрархічно пов'язані між собою.

Створення малюнка в matplotlib схоже з малюванням в реальному житті. Так художнику потрібно взяти основу (полотно або папір), інструменти (кисті або олівці), мати уявлення про майбутнє малюнку (що саме він буде малювати) і, нарешті, виконати все це і намалювати малюнок деталь за деталлю.

У matplotlib всі ці етапи також існують, і в якості художника-виконавця тут виступає сама бібліотека. Від користувача потрібно управляти діями художника-matplotlib, визначаючи що саме він повинен намалювати і якими інструментами. Зазвичай створення основи і процес непросредственно відображення малюнка віддає повністю на відкуп matplotlib. Таким чином, користувач бібліотеки matplotlib виступає в ролі управлінця. І чим простіше йому управляти кінцевим результатом роботи matplotlib, тим краще.

Так як matplotlib організована ієрархічно, а найбільш простими для розуміння людиною є самі високорівневі функції, то знайомство з matplotlib починають з самого високорівневого інтерфейсу **matplotlib.pyplot.** Так, щоб намалювати гістограму за допомогою цього модуля, потрібно викликати всього одну команду: matplotlib.pyplot.hist(arr) .

Користувачеві не потрібно думати як саме бібліотека намалювала цю діаграму. Якби ми малювали гістрограмму самостійно, то помітили б, що вона складається з повторяющіхся за формою фігур - прямокутників. А щоб намалювати прямокутник, потрібно знати хоча б координату одного кута і ширину / довжину. Малювали ж би ми прямокутник лініями, поєднуючи кутові точки прямокутника.

Цей приклад відображає ієрархічність малюнків, коли підсумкова діаграма (високий рівень) складається з простих геометричних фігур (нижчий, середній рівень), створених кількома універсальними методами малювання (низький рівень). Якби кожен малюнок потрібно було б створювати ось так, з нуля, це було б дуже довго і виснажливо.

Інтерфейс matplotlib.pyplot є набором команд і функцій, які роблять синтаксис графічних matploltib команд схожим на команди, які використовуються в середовищі MATLAB (c). Спочатку matplotlib планувався як вільна альтернатива MATLAB (c), де в одному середовищі були б кошти як для малювання, так і для чисельного аналізу. Саме так в Matplotlib з'явився pylab, який об'єднує модулі pyplot і numpy в один простір імен.

NB Мабуть, pylab виявився не дуже вдалою ідеєю. Є думка, що використання pylab - це поганий тон. Навчання за допомогою pylab може привести до неправильного розуміння роботи matplotlib за рахунок використання неявного імпортування. Так як у pylab немає істотних переваг, то далі будемо працювати тільки з pyplot або в об'єктно-орієнтованому стилі (ООС).

У той же час для більш серйозних завдань (впровадження matplotlib в призначену для користувача GUI) потрібно більше контролю над процесом і більше гнучкості, ніж можуть надати ці два модуля. Необхідний доступ до більш низькорівневим можливостям бібліотеки, яка реалізована в об'єктно-орієнтованому стилі. ООС помітно складніше для новачків і вимагає знань про роботу конкретних класів і їх методах, але надає найбільші можливості по взаємодії з бібліотекою matplotlib.

Автор вважає, що для просунутого (advanced) користувача необхідно розуміти як працювати з matplotlib не тільки через pytplot (початковий рівень), але і через ООС. При цьому не варто відмовлятися від використання pyplot там, де це зручно (наприклад, при відображенні створити малюнок на екран).

Ієрархічна структура малюнка в matplotlib

Головною одиницею (об'єктом найвищого рівня) при роботі з matplotlib є малюнок (Figure). Будь-який малюнок в matplotlib має вкладену структуру і чимось нагадує матрьошку. Призначена для користувача робота має на увазі операції з різними рівнями цієї матрьошки:

Figure (Малюнок) -> Axes (Область малювання) -> Axis (Координатна вісь)

- **Малюнок (Figure)**

Малюнок є об'єктом самого верхнього рівня, на якому розташовуються одна або кілька областей малювання (Axes), елементи малюнка Artisits (заголовки, легенда і т.д.) і основа-полотно (Canvas). На малюнку може бути кілька областей малювання Axes, але дана область малювання Axes може належати тільки одному малюнку Figure.

- **Область малювання (Axes)**

Область малювання є об'єктом середнього рівня, який є, напевно, головним об'єктом роботи з графікою matplotlib в об'єктно-орієнтованої стилі. Це те, що асоціюється зі словом "plot", це частина зображення з простором даних. Кожна область малювання Axes містить дві (або три в разі тривимірних даних) координатних осі (Axis об'єктів), які впорядковують відображення даних.

- **Координатна вісь (Axis)**

Координатна вісь є об'єктом середнього рівня, які визначають область зміни даних, на них наносяться поділу ticks і підписи до розподілам ticklabels. Розташування поділів визначається об'єктом Locator, а підписи поділок обробляє об'єкт Formatter. Конфігурація координатних осей полягає в комбінуванні різних властивостей об'єктів Locator і Formatter.

- **Елементи малюнка (Artists)**

Елементи малюнка Artists є як би червоною лінією для всіх ієрархічних рівнів. Практично все, що відображається на малюнку є елементом малюнка (Artist), навіть об'єкти Figure, Axes і Axis. Елементи малюнка Artists включають в себе такі прості об'єкти як текст (Text), плоска лінія (Line2D), фігура (Patch) та інші.

Коли відбувається відображення малюнка (figure rendering), всі елементи малюнка Artists наносяться на основу-полотно (Canvas). Велика частина з них пов'язується з областю малювання Axes. Також елемент малюнка не може спільно використовуватися декількома областями Axes або бути переміщений з одного на інший.

Інтерфейс прикладного програмування matplotlib API

У matplotlib образотворчі функції логічно розділені між декількома об'єктами, причому кожен з них сам має досить складну структуру. Можна виділити три рівні інтерфейсу прикладного програмування (matplotlib API):

1. **matplotlib.backend_bases.FigureCanvas** - абстрактний базовий клас, який дозволяє малювати і візуалізувати результати команд.
2. **matplotlib.backend_bases.Renderer** - об'єкт (абстрактний клас), який знає як малювати на FigureCanvas;
3. **matplotlib.artist.Artist** - об'єкт, який знає, як використовувати визуализатор (renderer), щоб малювати на полотні (canvas).

FigureCanvas і Renderer обробляють деталі, необхідні для взаємодії із засобами призначеного для користувача інтерфейсу, так як це робить WxPython або мову малювання PostScript. А Artist обробляє всі конструкції високого рівня такі як уявлення і розташування малюнка, тексту і ліній.

Звичайний користувач буде витрачати 95% свого часу, працюючи з Artists!

Надалі ми будемо мати справу виключно з Artists. Саме цей об'єкт дає ті високорівневі (простіше кажучи, більш зрозумілі людині) можливості для створення малюнків. Деталі як саме створюється і відображається графіка в matplotlib тут розглядатися не будуть.

Існує два типи об'єктів-класів Artists:

- примітиви (primitives);
- контейнери (containers).

Примітиви є стандартні графічні об'єкти: плоску лінію (Line2D), прямокутник (Rectangle), текст (Text), зображення (AxesImage) і т.д. А контейнери - це об'єкти-сховища, на які можна наносити графічні примітиви. До контейнерів відносяться малюнок (Figure), область малювання (Axes), координатна вісь (Axis), ділення (Ticks). Розглянемо контейнери докладніше, так як саме за допомогою звернень до різних контейнерів класу Artists, об'єднаних логічно в єдину структуру, буде здійснюватися настройка малюнків в matplotlib.

Всього існує 4 види Artists контейнерів:

- **Контейнери малюнка (Figure containers)**

Figure - це контейнер найвищого рівня. На ньому розташовуються всі інші контейнери і графічні примітиви.

- **Контейнери областей малювання (Axes containers)**

Axes - дуже важливий контейнер, так як саме з ним найчастіше працює користувач. Примірники Axes - це області, розташовані в контейнері Figure, для яких можна задавати координатну систему (декартова або полярна). На ньому розташовуються всі інші контейнери, крім Figure, і графічні примітиви. Це області на малюнку, на яких розташовуються графіки та діаграми, в які вставляються зображення і т.д. Мультіоконние малюнки складаються з набору областей Axes.

- **Контейнери осей (Axis containers)**

Axis схожий на Axes за назвою, але не варто їх плутати. Цей контейнер обслуговує екземпляри Axes. Він відповідає за створення координатних осей, на які будуть наноситися поділу осей, підписи поділок і ліній допоміжної сітки. Його спеціалізація (і відміну від контейнера Tick) - це розташування поділок і ліній, їх позиціонування та форматування підписів ділень, їх відображення.

- **Контейнери поділів (Tick containers)**

Контейнер нижчого рівня. Його спеціалізація (і відміну від контейнера Axis) - задавати характеристики (колір, товщина ліній) ліній сітки, поділів і їх підписів (розміри і типи шрифтів).

При створенні малюнка в matplotlib зазвичай надходять так: створюють екземпляр класу Figure (Figure instance), на якому виділяють одну або кількох областей Axes (або примірників Subplot), і використовують допоміжні методи екземпляра класу Axes (Axes instance) для створення графічних примітивів (primitives) . Якщо автоматично підібрані характеристики координатної сітки, поділів і їх підписів не влаштовують користувача, то вони налаштовуються за допомогою примірників контейнерів Axis і Tick, які завжди присутні на створеної області малювання Axes.

Pyplot

Інтерфейс pyplot дозволяє користувачеві зосередитися на виборі готових рішень і налаштування базових параметрів малюнка. Це його головна перевага, тому вивчення matplotlib найкраще починати саме з інтерфейсу pyplot.

Існує де-факто стандарт виклику pyplot в python:

In [3]:

```
# Де-факто стандарт виклику pyplot в python
import matplotlib.pyplot as plt
```

Малюнки в matplotlib створюються шляхом послідовного виклику команд: або в інтерактивному режимі (в консолі), або в скрипті (текстовий файл з python-

кодом). Графічні елементи (точки, лінії, фігури і т.д.) нашаровуються одна на іншу послідовно. При цьому наступні перекривають попередні, якщо вони займають загальне ділянки на малюнку (регулюється параметром **zorder).**

У matplotlib працює правило *"поточної області" ("current axes"),* яке означає, що всі графічні елементи наносяться на поточну область малювання. Незважаючи на те, що областей малювання може бути кілька, однf з них завжди є поточної.

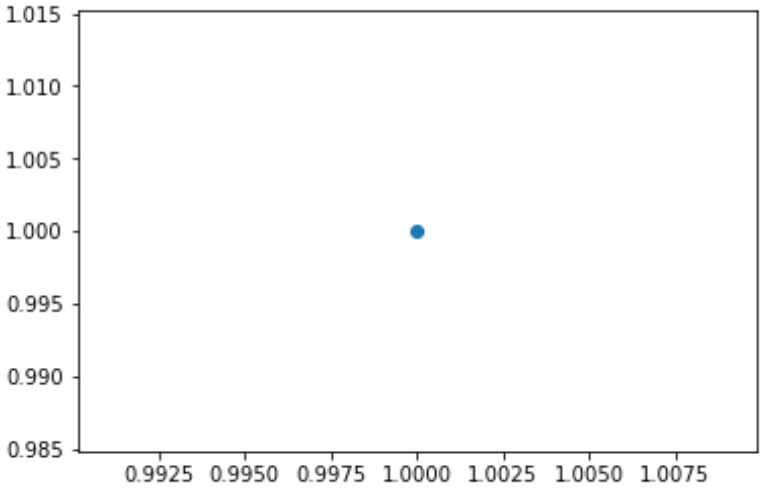
Як було сказано вище найголовнішим об'єктом в matplotlib є малюнок Figure. Тому створення наукової графіки потрібно починати саме з створення малюнка. Створити малюнок в matplotlib означає поставити форму, розміри і властивості основи-полотна (canvas), на якому буде створюватися майбутній графік.

Створити малюнок figure дозволяє метод plt.figure (). Після виклику будь-якої графічної команди, тобто функції, яка створює будь-якої графічний об'єкт, наприклад, plt.scatter () або plt.plot (), завжди існує хоча б одна область для малювання (за замовчуванням прямокутної форми).

Щоб результат малювання, тобто поточний стан малюнка, відбилося на екрані, можна скористатися командою plt.show() . Будуть показані всі малюнки (figures), які були створені.

In [7]:

```
import matplotlib.pyplot as plt

fig = plt.  figure () # Створення об'єкта Figure
print (fig. axes) # Список поточних областей малювання порожній
print (type (fig)) # тип об'єкта Figure
plt.  scatter (1.0, 1.0) # scatter - метод для нанесення маркера в точці (1.0, 1.0)

# Після нанесення графічного елементу у вигляді маркера
# Список поточних областей складається з однієї області
print (fig. axes)

# Дивись преамбулу
save (name = 'pic_1_4_1', fmt = 'pdf')
save (name = 'pic_1_4_1', fmt = 'png')

plt.  show ()
 []
<Class 'matplotlib.figure.Figure'>
[<Matplotlib.axes._subplots.AxesSubplot object at 0x0000000007C2C2B0>]
```



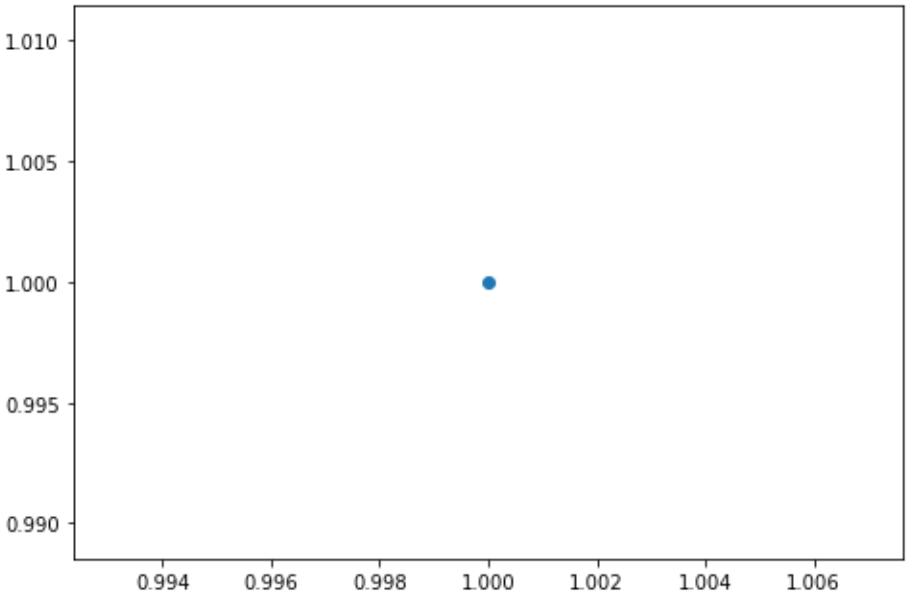
Зазвичай малюнок в matplotlib являє собою прямокутну область, задану в відносних координатах: від 0 до 1 включно по обох осях. Другий поширений варіант типу малюнка - кругла область (polar plot). Детальніше про таких типах графіків дивись главу "Графіки в полярних координатах" .

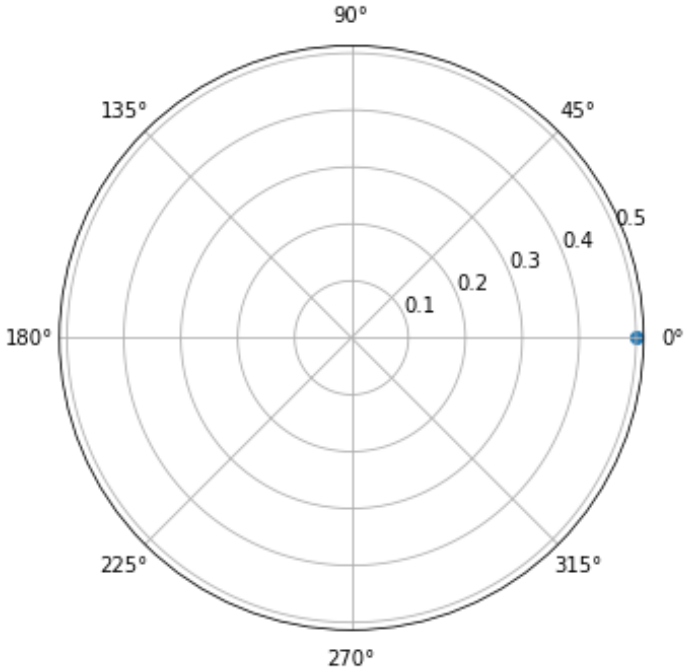
Щоб зберегти отриманий малюнок потрібно скористатися методом plt.savefig() . Він зберігає поточну конфігурацію поточного малюнка в графічний файл з деяким розширенням (png, jpeg, pdf та ін.), Який можна задати через параметр fmt. Тому її потрібно викликати в кінці вихідного коду, після всіх виклику всіх інших команд. Якщо в python-скрипті створити кілька малюнків figure і спробувати зберегти їх однією командою plt.savefig() , то буде збережено останній малюнок figure.

In [8]:

```
# Різні за формою області малювання

import matplotlib.pyplot as plt

fig = plt.  figure ()
# Додавання на малюнок прямокутної (за замовчуванням) області малювання
ax = fig.  add_axes ([0, 0, 1, 1])
print (type (ax))
plt.  scatter (1.0, 1.0)
plt.  savefig ( 'example 142a.png', fmt = 'png')


fig = plt.  figure ()
# Додавання на малюнок кругової області малювання
ax = fig.  add_axes ([0, 0, 1, 1], polar = True)
plt.  scatter (0.0, 0.5)
plt.  savefig ( 'example 142b.png', fmt = 'png')

# Дивись преамбулу
save ( 'pic_1_4_2', fmt = 'pdf')
save ( 'pic_1_4_2', fmt = 'png')

plt.  show ()
```

```
<Class 'matplotlib.axes._axes.Axes'>
```





Елементи малюнка Artists

Весь простір малюнка Figure (прямокутної або іншої форми) можна використовувати для нанесення інших елементів малюнка, наприклад, контейнерів Axes, графічних примітивів у вигляді ліній, фігур, тексту і так далі. У будь-якому випадку кожен малюнок можна структурно представити таким чином:

1. Область малювання Axes
- Тема області малювання -> plt.title() ;
2. Вісь абсцис Xaxis
- Підпис осі абсцис OX -> plt.xlabel() ;
3. Вісь абсцис Yaxis
- Підпис осі абсцис OY -> plt.ylabel() ;
4. Легенда -> plt.legend()
5. Колірна шкала -> plt.colorbar()
- Підпис горизонтальній осі абсцис OY -> cbar.ax.set_xlabel() ;
- Підпис вертикальної осі абсцис OY -> cbar.ax.set_ylabel() ;
6. Поділу на осі абсцис OX -> plt.xticks()
7. Поділу на осі ординат OY -> plt.yticks()

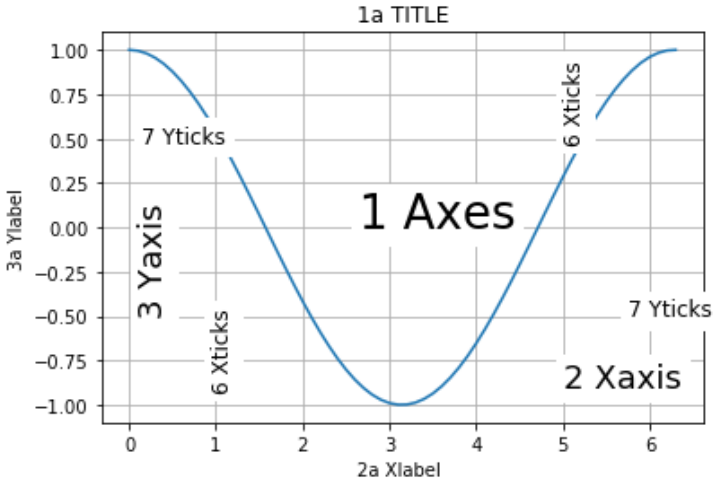
Для кожного з перерахованих рівнів-контейнерів є можливість нанести заголовок (title) або підпис (label). Підписи до малюнка полегшують розуміння того, в яких одиницях представлені дані на графіку або діаграмі.

Також часто на малюнок наносяться лінії допоміжної сітки (grid). У pyplot вона викликається командою plt.grid() . Допоміжна сітка пов'язана з поділами координатних осей (ticks), які визначаються автоматично виходячи з значень вибірки. Надалі буде показано як визначати положення і задавати значення поділок на координатних осях. Варто сказати, що в matplotlib існують головні ділення (major ticks) і допоміжні (minor ticks) для кожної координатної осі. За замовчуванням малюються тільки головні поділів і пов'язані з ними лінії сітки grid. У плані налаштування головні поділу нічим не відрізняються від допоміжних.

Якщо на малюнку присутній так званий "mappable object", то на малюнку може бути намальована колірна шкала (colorbar). До шкалою також можна робити підписи вздовж різних сторін. При цьому сама колірна може бути розташована як на поточній області малювання axes, відбираючи у неї деяку частку, або може бути розміщена на самостійної області малювання. Детальніше про кольорову шкалою в розділі "Колірна шкала" .

In [17]:

```
# Елементи простого малюнка
import matplotlib.pyplot as plt
import numpy as np
lag = 0.1
x = np.  arange (0.0, 2 * np. pi + lag, lag)
y = np.  cos (x)
fig = plt.  figure ()
plt.  plot (x, y)

plt.  text (np. pi - 0.5, 0, '1 Axes', fontsize = 26, bbox = dict (edgecolor = 'w', color = 'w'))
plt.  text (0.1, 0, '3 Yaxis', fontsize = 18, bbox = dict (edgecolor = 'w', color = 'w'), rotation = 90)
plt.  text (5, - 0.9, '2 Xaxis', fontsize = 18, bbox = dict (edgecolor = 'w', color = 'w'))
plt.  title ( '1a TITLE')
plt.  ylabel ( '3a Ylabel')
plt.  xlabel ( '2a Xlabel')

plt.  text (5, 0.85, '6 Xticks', fontsize = 12, bbox = dict (edgecolor = 'w', color = 'w'), rotation = 90)
plt.  text (0.95, - 0.55, '6 Xticks', fontsize = 12, bbox = dict (edgecolor = 'w', color = 'w'), rotation = 90)

plt.  text (5.75, - 0.5, '7 Yticks', fontsize = 12, bbox = dict (edgecolor = 'w', color = 'w'))
plt.  text (0.15, 0.475, '7 Yticks', fontsize = 12, bbox = dict (edgecolor = 'w', color = 'w'))

plt.  grid (True)

# Дивись преамбулу
save ( 'pic_1_5_1', fmt = 'pdf')
save ( 'pic_1_5_1', fmt = 'png')

plt.  show ()
```



In []:

```
# Елементи більш складного малюнка

import matplotlib.pyplot as plt
import numpy as np
```

```
N = 100
n = np.  sqrt (N)
x = np.  arange (n)
# Задаємо вибірку з Гамма-розподіленого з параметрами форми = 1.  і масштабу = 0.
z = np.  random.  random (N).  reshape (n, n)
y = z [5,:]

fig = plt.  figure ()
cc = plt.  contourf (z, alpha = 0.5) # тривимірне поле
plt.  plot (x, y, label = 'line', color = 'red') # червона лінія

plt.  title ( '1a. Title') # заголовок
plt.  xlabel ( '2a. Xlabel') # підпис осі OX
plt.  ylabel ( '3a. Ylabel') # підпис осі OY
plt.  legend () # легенда
cbar = plt.  colorbar (cc) # колірна шкала

plt.  text (2.5, 7, '1. Axes', fontsize = 26, bbox = dict (edgecolor = 'w', color = 'w'))
plt.  text (4, - 0.5, '2. XAxis', fontsize = 18, bbox = dict (edgecolor = 'w', color = 'w'))
plt.  text (- 0.5, 3.8, '3. YAxis', fontsize = 18, bbox = dict (edgecolor = 'w', color = 'w'), rotation = 90)
plt.  text (6.3, 7.2, '4. Legend', fontsize = 16, bbox = dict (edgecolor = 'w', color = 'w'))
plt.  text (9.1, 5., '5. Colorbar', fontsize = 16, bbox = dict (edgecolor = 'w', color = 'w'), rotation = 90)
plt.  text (7., 0.8, '6. Xticks', fontsize = 12, bbox = dict (edgecolor = 'w', color = 'w'))
plt.  text (0.8, 8.4, '7. Yticks', fontsize = 12, bbox = dict (edgecolor = 'w', color = 'w'), rotation = 90)

# Підписи для колірних шкал мають відмінний від інших підписів синтаксис
cbar.  ax.  set_xlabel ( '5a. Colorbar Xlabel', color = 'k', rotation = 30)
cbar.  ax.  set_ylabel ( '5b. Colorbar Ylabel', color = 'k')

plt.  text (2.8, 4.8, '6. Grid lines', fontsize = 14)
plt.  grid (True)

# Дивись преамбулу
save ( 'pic_1_5_2', fmt = 'pdf')
save ( 'pic_1_5_2', fmt = 'png')

plt.  show ()
```

Властивості графічних елементів

Різноманіття і зручність створення графіки в matplotlib забезпечується не тільки за рахунок створених графічних команд, але і за рахунок багатого арсеналу по конфігурації типових форм. Ця установка включає в себе роботу з кольором, формою, типом лінії або маркера, товщиною ліній, ступенем прозорості елементів, розміром і типом шрифту і іншими властивостями.

Параметри, які визначають ці властивості в різних графічних командах, зазвичай мають однаковий синтаксис, тобто називаються однаково. Стандартним способом завдання властивостей будь-якого створюваного об'єкта (або методу) є передача по ключу: ключ = значення. Найбільш часто зустрічаються назви параметрів зміни властивостей графічних об'єктів перераховані нижче:

- *color / colors / c* - колір;
- *linewidth / linewidths* - товщина лінії;
- *linestyle* - тип лінії;
- *alpha* - ступінь прозорості (від повністю прозорого 0 до непрозорого 1);
- *fontsize* - розмір шрифту;
- *marker* - тип маркера;
- *s* - розмір маркера в методі plt.scatter (тільки цифри);
- *rotation* - поворот рядки на X градусів.

При створенні функцій або методів класів, особливо в разі успадкування (див. Додаток 1), параметри часто передають у вигляді об'єднань послідовностей: кортежу або словника. Для цього існують спеціальні символи-приставки: "*" або "**" відповідно. Це особливо корисно у випадках, коли функція / метод може приймати змінна число параметрів.

Прийнято, що для передачі кортежу використовується змінна **args,** а в разі зі словником - **kwargs.** Якщо перед змінної args вказано символ "*", то всі додаткові аргументи, передані функції / методу, збережуться в args у вигляді кортежу. Якщо перед args буде вказано символ "**", то всі додаткові параметри будуть розглядатися як пари "ключ - значення" в словнику.

У функціях / методах описують властивості таких графічних об'єктів як лінія, текст, прямокутник, параметри часто поєднують у вигляді послідовностей * args, або словників ** kwargs. Так зручніше при створенні класів і їх методів (дивись приклад 1.6.1).

Якщо в описі графічного методу вказано приблизно так, як в,

plt.plot (* args, ** kwargs)

то це означає, що в якості вхідних даних потрібно спочатку список / кортеж параметрів (найчастіше потрібна хоча б одна послідовність типу значень функції Y), а після цього можна передавати значення параметрів за ключовими іменами цих параметрів (color, linewidth і т.д.).

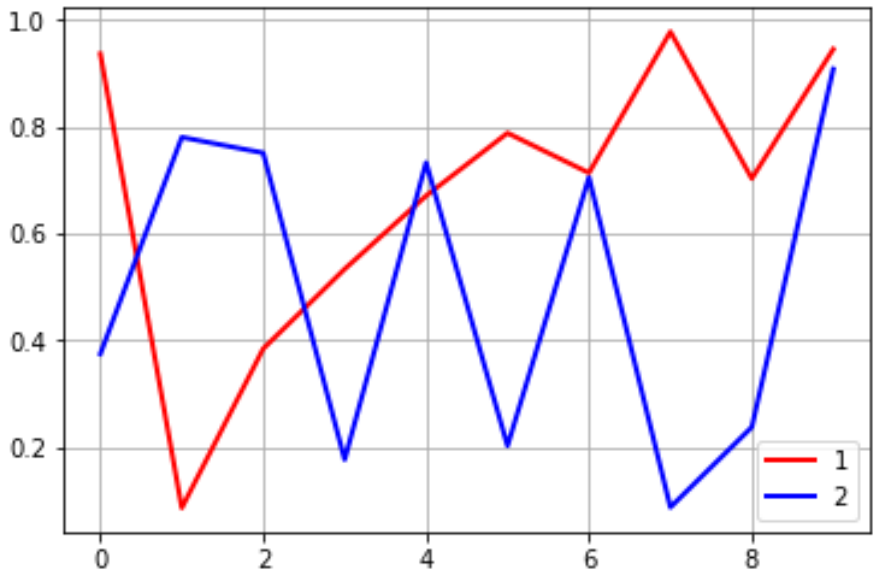
In [18]:

```
import numpy as np

# Приклад функції з об'єднанням в кортеж * args
def f_sums ( * args ):

    list1 = []
    for arg in args :
        a = 0
        for i in arg :
            a += i
        list1 . append ( a )

    return list1

# Приклад функції з об'єднанням в словник ** kwargs
def f_words ( ** kwargs ):
    ''' 
Функція pop повертає значення для заданого ключа (якщо він є в словнику)
і видаляє з словника пару "ключ - значення"
' "


    print 'Словник kwargs ДО виклику методу pop:' , kwargs
    per1 = kwargs . pop ( 'solo' , 'Han' )
```

```
    per2 = kwargs . pop ( 'wookie' , 'Chubbaca' )
    act = kwargs . pop ( 'loves' , 'loves' )
    str1 = ' % s % s % s ' % ( per1 , act , per2 ) print
     'Словник kwargs ПІСЛЯ виклику методу pop:' , kwargs

    return str1

# Приклад функції з об'єднанням і в кортеж args і в словник ** kwargs
def f_plot ( * args , ** kwargs ):

    xlist = []
    ylist = []
    for i , arg in enumerate ( args ):
      if ( i % 2 == 0 ):
        xlist . append ( arg )
      else :
        ylist . append ( arg )

    colors = kwargs . pop ( 'colors' , 'k' )
    linewidth = kwargs . pop ( 'linewidth' , 1. )

    fig = plt . figure ()
    ax = fig . add_subplot ( 111 )
    i = 0
    for x , y , color in zip ( xlist , ylist , colors ):
      i + = 1
      ax . plot ( x , y , color = color , linewidth = linewidth , label = str ( i ))

    ax . grid ( True )
    ax . legend ()
    save ( 'ex_1_6_1' , fmt = 'pdf' )
    save ( 'ex_1_6_1' , fmt = 'png' )

# ==================================
# MAIN SCRIPT BODY

x = np . arange ( 10 )
y = np . random . random ( 20 )
z = np . linspace ( - 15 , - 7.5 , 37 )
xyz = [ x , y , z ]

abc = { 'solo' : 1 , 'wookie' : 'green' , 'friend' : True }

res1 = f_sums ( * xyz )
res2 = f_words ( ** abc )

print ( 'res1' , type ( res1 ), res1 )
print ( 'res2' , type ( res2 ), res2 )

""
 Оскільки в plt.plot немає обов'язкових параметрів, то передані
в цю функцію через кома послідовності або масиви будуть оброблені
```

Тут приклад передачі двох ліній - дві послідовності з пари OX-OY (x, y2) і (x, y3). Їм у відповідність представлена послідовність кольорів colors.
"" "
Colors = ['red' , 'blue']
N = 10
x = np . Arange (N)
y2 = np . Random . Random (N)
y3 = np . Random . random (N)

f_plot (x , y2 , x , y3 , colors = colors , linewidth = 2.)
Словник kwargs ДО виклику методу pop: { 'wookie': 'green', 'solo': 1, 'friend': True}
Словник kwargs ПІСЛЯ виклику методу pop: { 'friend': True}
('Res1', <type 'list'>, [45, 10.895519422104705, -416.25])
('Res2', <type 'str'>, '1 loves green')



In [14]:
 # Властивості графічних елементів

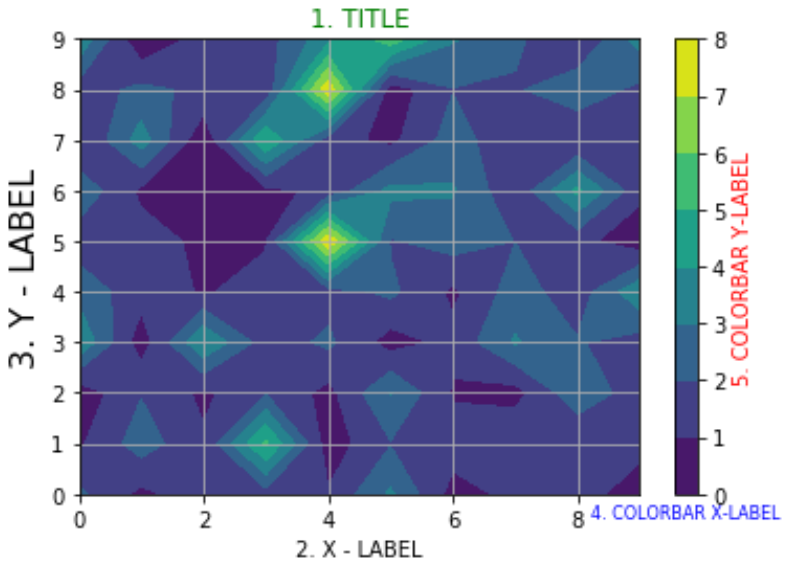
import matplotlib.pyplot as plt
import numpy as np

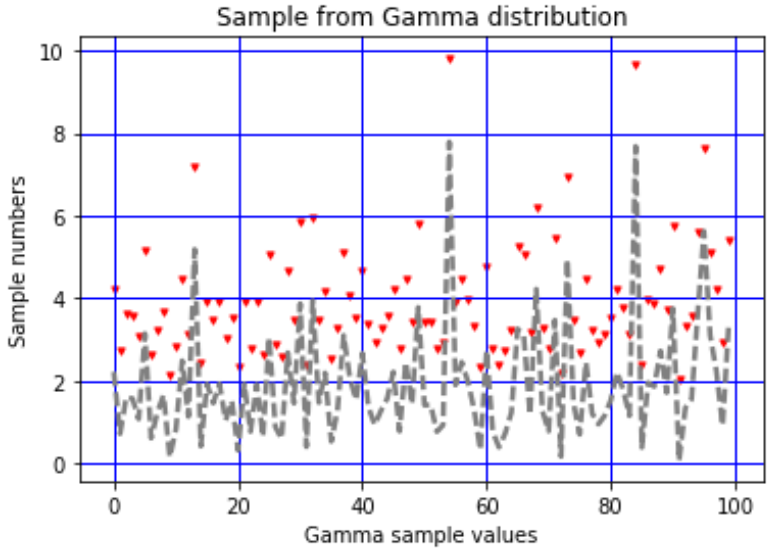
N = 100
x = np . arange (N)
 # Задаємо вибірку з Гамма-розподіленого з параметрами форми = 1. і масштабу = 0.
z = np . random . gamma (2. , 1. , N)
z1 = np . cos (x / 10.)
z2 = np . cos (x / 20.)

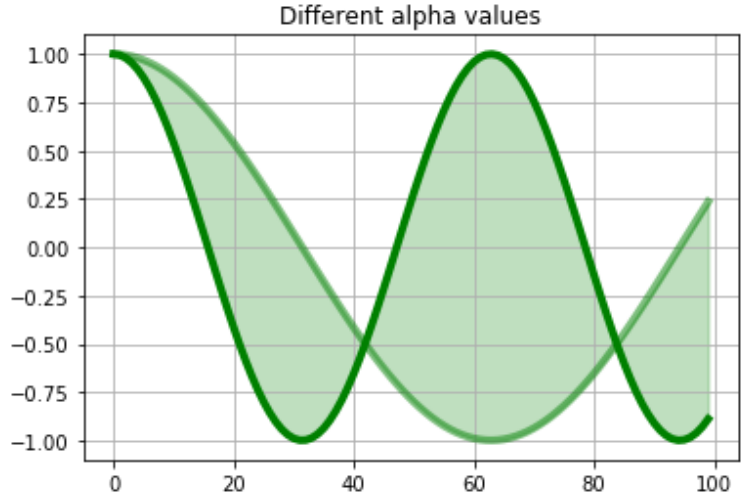
y = z . reshape (10 , 10)
#y = np.cos (y)

fig = plt . figure ()
cc = plt . contourf (y)
cbar = plt . colorbar (cc)

```
 plt. title  (  '1. TITLE'  ,  color  =  'green'  )
 plt. xlabel  (  '2. X - LABEL'  )
 plt. ylabel  (  '3. Y - LABEL'  ,  fontsize  =  16  )

# Підписи для колірних шкал мають відмінний від інших підписів синтаксис
cbar  .  ax  .  set_xlabel  (  '4. COLORBAR X-LABEL'  ,  color  =  'b'  )
cbar  .  ax  .  set_ylabel  (  '5. COLORBAR Y-LABEL'  ,  color  =  'r'  )
 plt. grid  (  True  )

fig  =  plt  .  figure  ()

my_dict  =  {  'color'  :  'grey'  ,  'linewidth'  :  2.5  ,  'linestyle'  :  '-'  }
xz  =  [  x  ,  z  ]

# Передача параметрів через список xz і словник my_dict. Наявність знаків * і ** обов'язково!
cc  =  plt  .  plot  (  *  xz  ,  **  my_dict  )
# результат аналогічний такого запису
#cc = plt.plot (x, z, color = 'grey', linewidth = 2.5, linestyle = '-')

 plt. scatter  (  x  ,  y  +  2.0  ,  marker  =  'v'  ,  s  =  10  ,  color  =  'red'  )

 plt. title  (  'Sample from Gamma distribution'  )
 plt. xlabel  (  'Gamma sample values'  )
 plt. ylabel  (  'Sample numbers'  )

# Підписи для колірних шкал мають відмінний від інших підписів синтаксис
cbar  .  ax  .  set_xlabel  (  '4. COLORBAR X-LABEL'  ,  fontsize  =  8  )
cbar  .  ax  .  set_ylabel  (  '5. COLORBAR Y-LABEL'  ,  color  =  'r'  )
 plt. grid  (  True  ,  color  =  'blue'  ,  linewidth  =  1.0  )

fig  =  plt  .  figure  ()

# Створення словника
my_dict  =  {  'color'  :  'green'  ,  'linewidth'  :  4.0  ,  'alpha'  :  0.5  }

 plt. fill_between  (  x  ,  z2  ,  z1  ,  color  =  'green'  ,  alpha  =  0.25  )
 plt. plot  (  x  ,  z1  ,  color  =  'green'  ,  linewidth  =  4.0  )
 plt. plot  (  x  ,  z2  ,  **  my_dict  )
 plt. title  (  'Different alpha values'  )
 plt. grid  (  True  )

# Дивись преамбулу
save  (  'pic_1_6_2'  ,  fmt  =  'pdf'  )
save  (  'pic_1_6_2'  ,  fmt  =  'png'  )

 plt. show  ()
```


1. TITLE
3. Y - LABEL
2. X - LABEL
5. COLORBAR Y-LABEL
4. COLORBAR X-LABEL


Sample from Gamma distribution
Sample numbers
Gamma sample values


Different alpha values

ТЕМА 12: СТРУКТУРА МАЛЮНКА В MATPLOTLIB.

Малюнок **Figure** є основою кожного зображення, які створюється в matplotlib. Зазвичай після створення екземпляра малюнка (точніше - екземпляра класу matplotlib.figure.Figure) про нього можна забути. Проте, саме на основі примірника Figure викликаються екземпляри Axes, які несуть основне навантаження при створенні наукової графіки.

Варто пам'ятати, що поки йде робота з одним єдиним екземпляром типу Figure, то результат виконання всіх графічних команд будуть відображатися саме на ньому. При роботі з багатьма екземплярами Figure не можна забувати про очищення "полотна" від "фарб" Попередня малюнка.

Електронні ресурси:

- Опис елементів малюнка Artists в matplotlib .

Контейнер Figure

Верхнім логічним рівнем серед контейнерів Artists (див. Главу 1) є matplotlib.figure.Figure, який включає все, що є на малюнку (figure). Фоном (background) малюнка, який обирається в Figure.patch, за замовчуванням є прямокутник (Rectangle). Коли на малюнку додаються екземпляри subplots (наприклад, за допомогою методу add_subplot ()) або axes (за допомогою методу add_axes ()), вони автоматично додаються в список Figure.axes. Так як matplotlib підтримує концепцію "поточної області" ("current axes"; докладніше див. Figure.gca і Figure.sca) для роботи з pylab / pyplot, то не рекомендується додавати або прибирати axes зі списку (axes list) вручну. Краще використовувати методи add_subplot () / add_axes () для додавання примірників Axes і delaxes () для їх видалення.

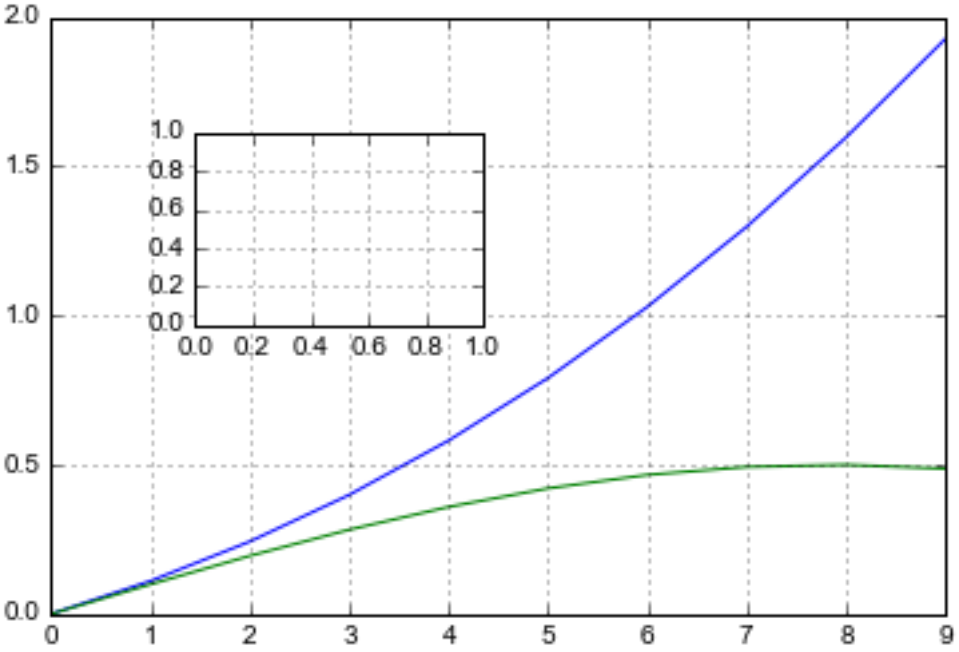
Порада

Можна використовувати список fig.axes () для роботи з екземплярами Axes. Наприклад, для додавання ліній сітки за допомогою методу grid () на всі axes малюнка.

In [49]:

```
import matplotlib.pyplot as plt
import numpy as np

fig = plt.  figure ()

print type (fig)
# Створення примірника Axes с допомогою Figure-методу add_subplot ()
ax = fig.  add_subplot (111)
# або так
box = [0.25, 0.5, 0.25, 0.25]
# Створення примірника Axes с допомогою Figure-методу add_axes ()
ax2 = fig.  add_axes (box)

x = np.  arange (0.0, 1.0, 0.1)
y = np.  sin (x) * np.  exp (x)
z = np.  cos (x) * np.  sin (x)

# Методи plot () викликаються через екземпляри ax, а не plt (інтерфейс pyplot)
ax.  plot (y)
ax.  plot (z)

for ax in fig.  axes:
```

```
ax.  grid (True)

save ( 'pic_5_1', fmt = 'png')
save ( 'pic_5_1', fmt = 'pdf')

plt.  show ()
 <Class 'matplotlib.figure.Figure'>
```



Список елементів малюнка (Artists) для контейнера Figure:

1. **axes** - список примірників Axes (включаючи Subplot);
2. **images** - список FigureImages patches (ефективно для відображення пікселів);
3. **legends** - список примірників Figure Legend (відрізняється від Axes.legends);
4. **lines** - список примірників Figure Line2D (рідко використовується, див. Axes.lines);
5. **patch** - фон Rectangle;
6. **patches** - список Figure patches (рідко використовується, див. Axes.patches);
7. **texts** - список примірників Figure Text.

Конфігурація Figure

Одним із стандартних методів створення екземпляра Figure є наступний:

In [50]:

```
 import matplotlib.pyplot as plt

fig = plt.  figure ()
print (u 'Тип Figure% s'% type (fig))
 Тип Figure <class 'matplotlib.figure.Figure'>
 <Matplotlib.figure.Figure at 0xbd4a9b0>
```

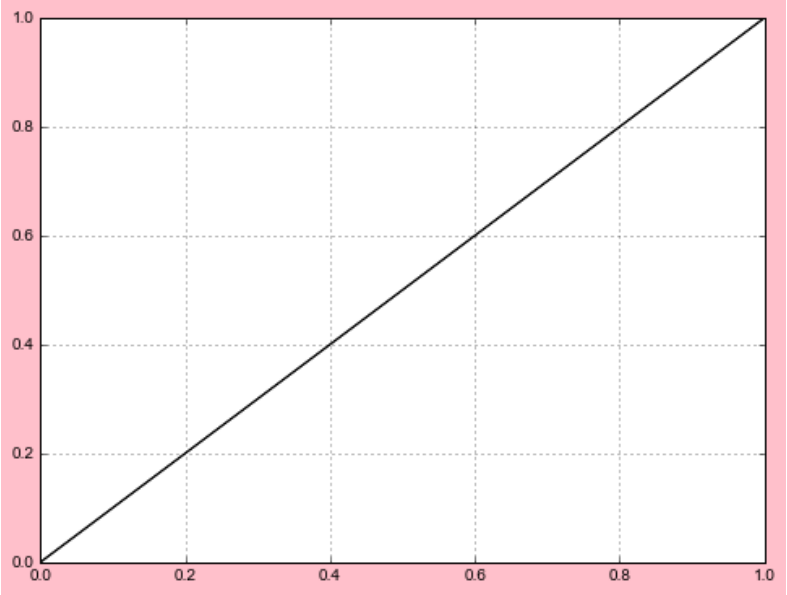
Основні параметри створеного екземпляра Figure визначаються в файлі конфігурації matplotlibrc. Атрибути Figure можна змінити під час створення примірника:

- **figsize:** кортеж з цілих або дійсних чисел, який визначає ширину і висоту в дюймах. За замовчуванням дорівнює значенню figure.figsize з налаштувань rcParams;
- **dpi:** дозвіл малюнка. За замовчуванням дорівнює значенню figure.dpi з налаштувань rcParams;
- **facecolor:** колір фону. За замовчуванням дорівнює значенню figure.facecolor з налаштувань rcParams;

- **edgecolor:** колір кордону. За замовчуванням дорівнює значенню figure.edgecolor з налаштувань rcParams.

In [51]:

```
import matplotlib.pyplot as plt
from matplotlib import rcParams

figsize = (8, 6)

fig = plt.  figure (figsize = figsize, facecolor = 'pink', frameon = True)
plt.  plot ([[0,0, 0.0], [1., 1.]], 'k')
plt.  grid (True)
rcParams [ 'figure.edgecolor'] = 'blue'

save ( 'pic_5_2_1', fmt = 'png')
save ( 'pic_5_2_1', fmt = 'pdf')

plt.  show ()
```

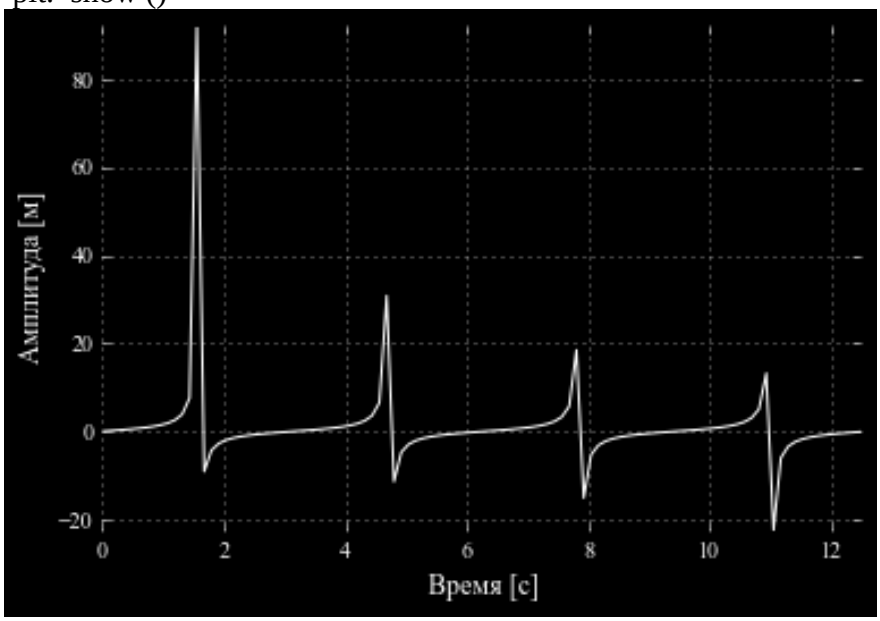


За замовчуванням кольору координатної координатних осей, ліній допоміжної сітки отрісовиваємих чорним кольором, а колір фону (основи малюнка) - білим. Замінивши параметри, що відповідають за відповідні кольори, можна створити, наприклад, інвертований біло-чорний малюнок. Іноді така графіка виглядає дуже наочно, стильно і привертає до вашого матеріалу підвищену увагу.

In [52]:

```
from matplotlib import rcParams
import matplotlib.pyplot as plt
import numpy as np

rcParams [ 'font.family'] = 'Times New Roman', 'Arial', 'Tahoma'
rcParams [ 'font.fantasy'] = 'Times New Roman'

# Зміна параметрів малювання (зміна чорного по білому на біле по чорному)
facecolor = 'k'

rcParams [ 'figure.edgecolor'] = facecolor
```

```
rcParams [ 'figure.facecolor'] = facecolor
rcParams [ 'axes.facecolor'] = facecolor
#rcParams [ 'axes.edgecolor'] = 'k'
rcParams [ 'grid.color'] = 'w'
rcParams [ 'xtick.color'] = 'w'
rcParams [ 'ytick.color'] = 'w'
rcParams [ 'axes.labelcolor'] = 'w'

x = np.  arange (0, 4 * np. pi, 0.12)
y = np.  tan (x)

fig = plt.  figure ()
plt.  plot (x, y, 'w')

plt.  xlim (x [0], x [- 1])
plt.  ylim (np. min (y), np. max (y))

plt.  xlabel (u 'Час [с]', fontsize = 12)
plt.  ylabel (u 'Амплітуда [м]', fontsize = 12)
plt.  grid (True, color = 'w')

save ( 'pic_5_2_2', fmt = 'png')
save ( 'pic_5_2_2', fmt = 'pdf')

plt.  show ()
```



Збереження малюнка

Зберегти вийшов малюнок можна за допомогою методу savefig, застосованого або до примірника Figure (fig.savefig ()), або через pyplot (plt.savefig ()).

Варто пам'ятати, що за замовчуванням метод savefig має значення атрибутів типу facecolor відмінні від значень figure.facecolor. У файлі конфігурації matplotlibrc можна налаштувати відповідні параметри як для savefigure, так і для figure. Тому щоб зберегти призначені для користувача настройки малюнка методом savefig, потрібно змінити атрибути savefig, а не відповідні атрибути figure (dpi, format, edgecolor, facecolor, frameon) або передати відповідні значення атрибутів явно при виклику метод savefig:

In [53]:

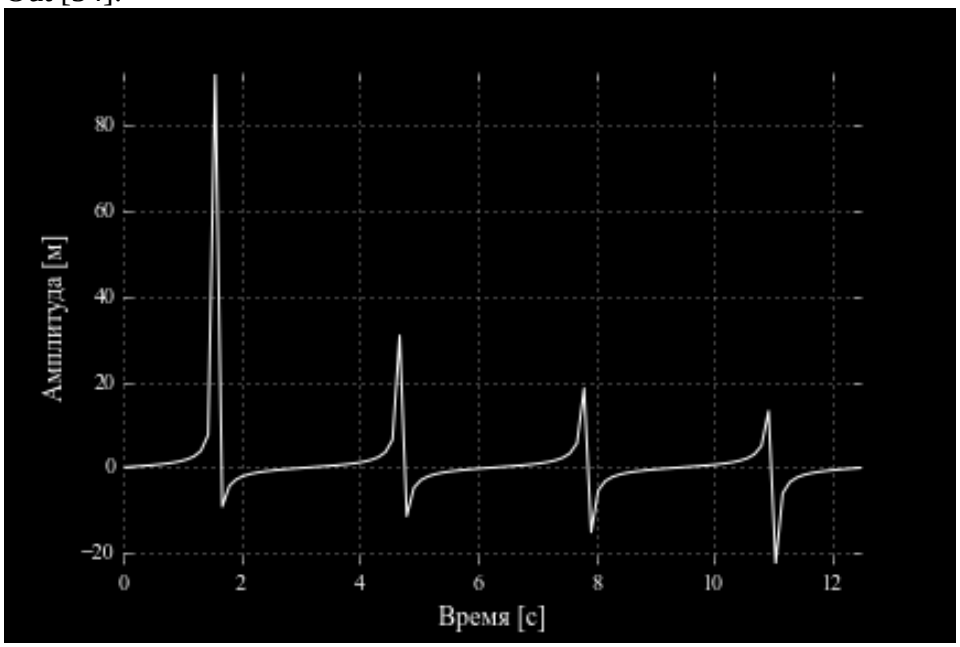
 # Збереження малюнка з параметрами кольору фону за замовчуванням rcParams ['savefig.edgecolor'] = 'k' rcParams ['savefig.facecolor'] = 'k' rcParams ['grid.color'] = 'w' rcParams ['xtick. color '] =' w 'rcParams [' ytick.color '] =' w 'rcParams [' axes.labelcolor '] =' w 'fig. savefig ('./pictures/png/pic_5_3_1.png', format = 'png') fig. savefig ('./pictures/png/pic_5_3_1.pdf', format = 'pdf') *# Збереження малюнка до призначених для користувача параметрами кольору фону* rcParams ['savefig.edgecolor'] = 'w' rcParams ['savefig.facecolor'] = ' w 'rcParams [' grid.color '] =' k 'rcParams [' xtick.color '] =' k 'rcParams [' ytick.color '] =' k 'rcParams [' axes.labelcolor '] =' k ' *# Атрибут facecolor передається явно* fig. savefig ('./pictures/png/pic_5_3_2.png', format = 'png', facecolor = 'k') fig. savefig ('./pictures/png/pic_5_3_2.pdf', format = 'pdf', facecolor = 'k')

In [54]:

```
print ( 'figure.facecolor:% s'% rcParams [ 'figure.facecolor'])

from IPython.display import Image
#Image (filename = 'pic example 5_3_1.png')
Image (filename = './pictures/png/pic_5_3_1.png')
```

figure.facecolor: k

Out [54]:

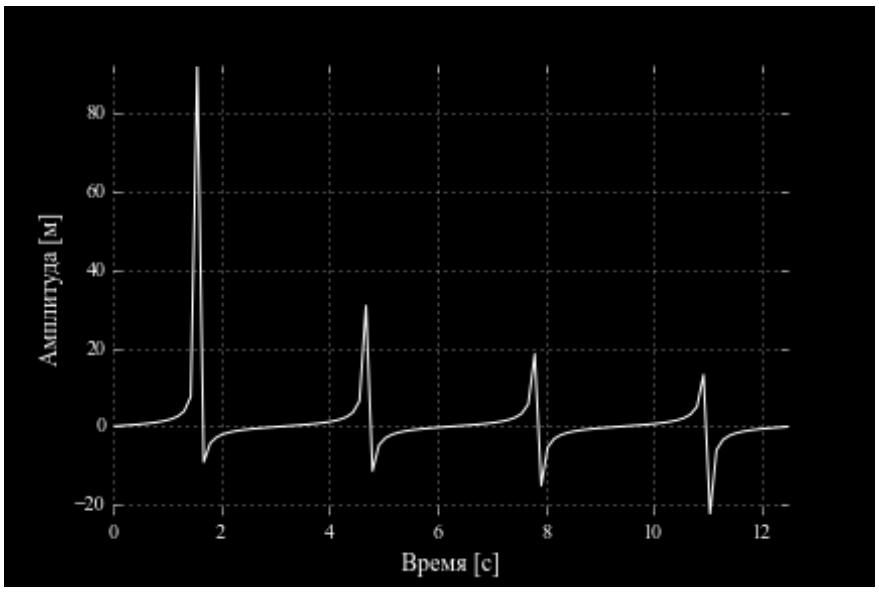


In [55]:

```
from IPython.display import Image

print ( 'figure.facecolor:% s'% rcParams [ 'figure.facecolor'])

# Тут атрибут facecolor був переданий явно (figure.facecolor = 'k')
Image (filename = './pictures/png/pic_5_3_2.png')
```

figure.facecolor: k

Out [55]:



Так як для збереження малюнків методом savefig потрібно явно задати деякі атрибути, то серед атрибутів при створенні екземпляра Figure має сенс ставити ті, які не є вхідними для методу savefig, а саме figsize. Figsize дозволяє зробити малюнок менше або більше і визначає фізичний розмір малюнка. Так можна підібрати потрібний розмір для друку.

Для довідки:

1 дюйм = 2.54 см

Лист А4 має розміри 210х297 мм

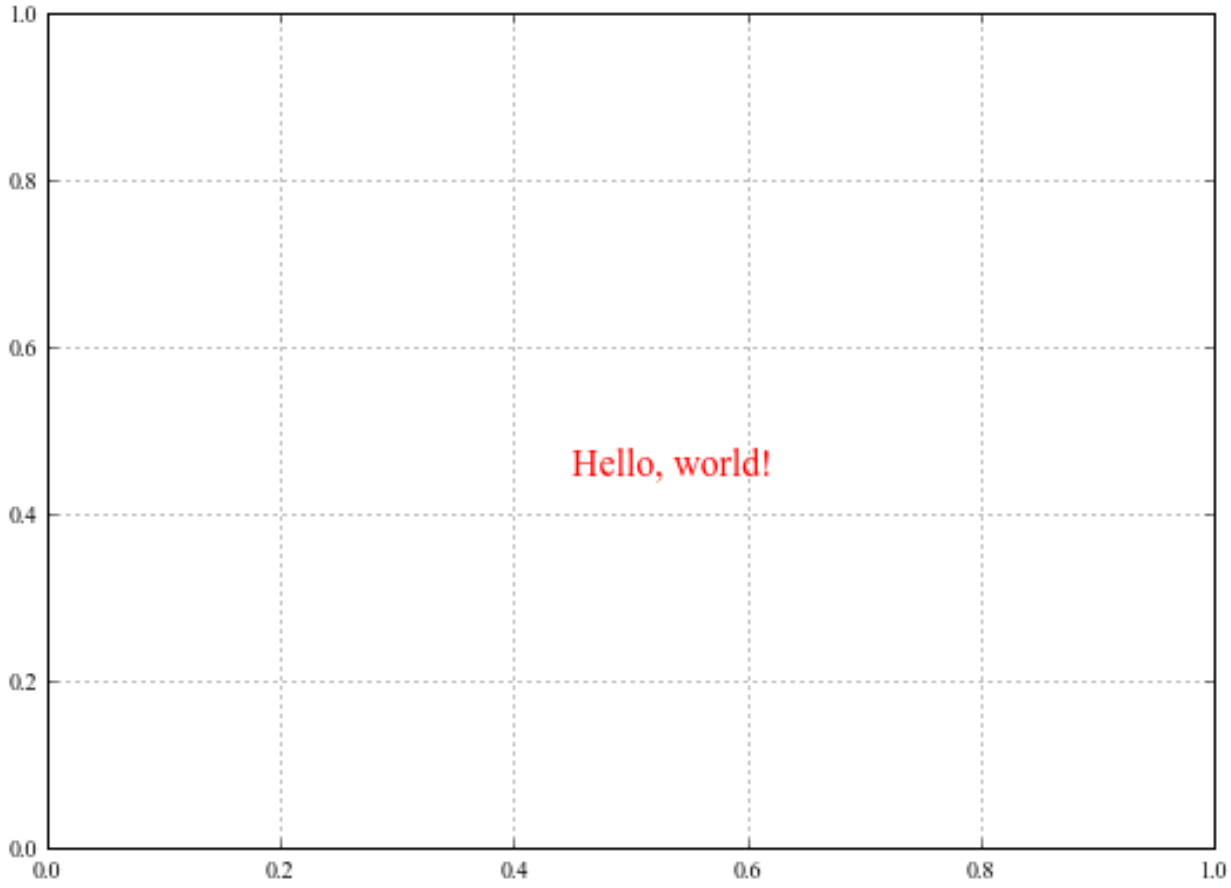
Дозвіл малюнка можна отримати помноживши значення атрибутів figure.figsize і figure.dpi. DPI показує скільки точок-пікселів доводиться на один дюйм.

In [56]:

```
from matplotlib import rcParams
import matplotlib.pyplot as plt
import numpy as np

def sm2inch (a):
    ''' 
Converts inches to santimeters
'''
    b = []
    for i in a:
        b. append (i / 2.54)
    return b

# Повертаємо звичайні значення кольорів для подальшої інтерактивної роботи

rcParams [ 'figure.edgecolor'] = 'w'
rcParams [ 'figure.facecolor'] = 'w'
rcParams [ 'axes.facecolor'] = 'w'
rcParams [ 'axes.edgecolor'] = 'k'
rcParams [ 'grid.color'] = 'k'
rcParams [ 'xtick.color'] = 'k'
rcParams [ 'ytick.color'] = 'k'
rcParams [ 'axes.labelcolor'] = 'k'

figsize = (3.5, 2.5) # см
```

```
figsize = sm2inch (figsize)
print figsize
fig2 = plt.  figure (figsize = figsize, facecolor = 'w')
rcParams [ 'figure.edgecolor'] = 'w'
plt.  text (0.45, 0.45, 'Hello, world!', color = 'red', fontsize = 16)
plt.  grid (True)

save ( 'pic_5_3_3', fmt = 'png')
save ( 'pic_5_3_3', fmt = 'pdf')

plt.  show ()
```

```
 [8.89, 6.35]
```



ТЕМА 13: СПЕЦІАЛЬНІ ЕЛЕМЕНТИ МАЛЮНКА В MATPLOTLIB.

У попередніх розділах ми рассмотриваются в основному прості графіки, де область малювання Axes була одна. У цей параграфі ми поговоримо про складні малюнках, які можуть містити не одну, а кілька областей малювання. Замість слова "кілька" можна було б вжити слово "багато", але краще ніколи не малювати багато графіків на одному малюнку.

Золоте правило:

Неприпустимо, якщо не всі частини або елементи малюнка зрозумілі або добре розрізняється на тлі інших. Малюнок повинен бути читабельним, а не коробкою, в яку автор напхав від жадібності (або жорстких вимог журналу / видавця) все, що зміг створити. Потрібно слідувати python-заповіді (див. `import this`): "складне краще, ніж заплутане". Як зробити так, щоб заплутані малюнки стали складними, зберігши при цьому функціональність і привабливий вигляд, показано в цьому розділі.

Методи створення мультіокон

У matplotlib реалізовано кілька споб створення малюнків з кількома областями для діаграм (multi-panel plots):

1. **fig.add_axes()** - базовий метод, зручний при створенні діаграми-врізки;
2. **fig.add_subplot()** - додавання одного subplot на малюнок. Зручно для відображення 2-3 діаграм;
3. **plt.subplot()** - аналогічний попередньому за результатом метод для pyplot;
4. **plt.subplots()** - зручний метод для автоматизованого створення декількох subplots;
5. **plt.GridSpec()** - метод для об'єднання осередків subplots в більш складні конфігурації. Дозволяє створювати різні за формою subplots на малюнку.

Перші два методи (з приставкою `add_`) є більш нізкоуровнемимі, і для їх виклику потрібно об'єкт Figure. Останні три методи реалізовані в pyplot інтерфейсі. Кожен з цих методів створює один або більше примірників типу subplot (в прикладах буде приведений точний тип цього об'єкта) або axes. Кожен такий об'єкт - окрема область малювання і до неї застосовні всі ті прийоми, які ми розглянули в попередньому розділі, коли працювали з екземпляром-об'єктом типу axes. Хоча метод `fig.add_axes` створює об'єкт типу Axes, а всі інші - типу AxesSubplot, тобто різні типи об'єктів, вони є спорідненими, і працювати з ними можна одними і тими ж методами.

Найпростішим способом розбити, тобто поділити, малюнок на кілька частин є спосіб `fig.add_subplot()` . Сенс створення subplots полягає в тому, що вони наповнюють малюнок осередками (як в таблиці) в кожній з яких тепер можна створювати свій власний графік або діаграму (або карту-схему).

Метод `fig.add_subplot()` завжди має три обов'язкових аргументу: число осередків по вертикалі (число рядків), число осередків по горизонталі (число стовпців) і номер комірки (відлік ведеться від 1 зліва направо, зверху вниз). Три числа перераховані через кому (3,2,5) або триплет у вигляді одного числа (325), передані методу fig.add_subplot (), створюють екземпляр (instance) " `matplotlib.axes._subplots.AxesSubplot` ", що є підвидом більш широкого класу " `matplotlib.axes` ".

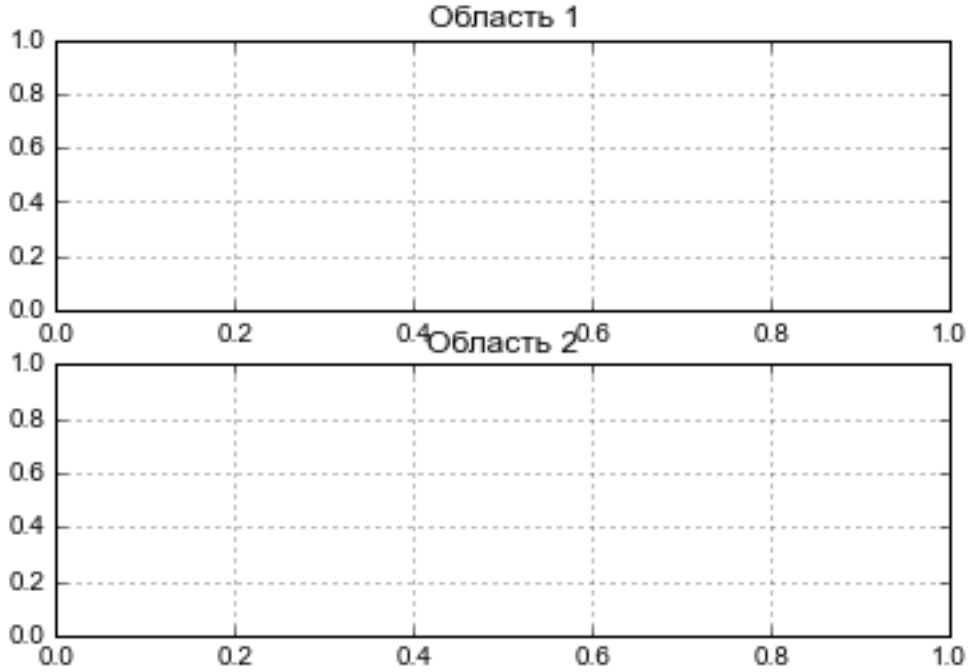
`fig.add_subplot()` оптимальний, коли необхідно швидко зобразити кілька (зазвичай 2-3) графіків-діаграм поруч. Бажано при цьому, щоб вони не перекривалися, за винятком випадків, коли області малюнка "злипаються" і утворюють спільні кордони (як по осі абсцис, так і по осі ординат).

In [2]:

 # Приклад Створення двох областей на малюнку за допомогою fig.add_subplot ()

```
import matplotlib.pyplot as plt

fig = plt.  figure ()

tri = 211
ax1 = fig.  add_subplot (tri)
ax1.  set_title (u 'Область 1')
ax2 = fig.  add_subplot (2, 1, 2)
ax2.  set_title (u 'Область 2')

# Дізнаємося координати областей, які займають subplots
print u 'Область 1', ax1
print type (ax1)
print u 'Область 2', ax2

# Намалюємо в кожному subplot лінію сітки
for ax in fig.  axes:
    ax.  grid (True)

save ( 'pic_7_1_1', fmt = 'png')
save ( 'pic_7_1_1', fmt = 'pdf')

plt.  show ()
```

```
 Область 1 Axes (0.125,0.547727; 0.775x0.352273)
<Class 'matplotlib.axes._subplots.AxesSubplot'>
Область 2 Axes (0.125,0.125; 0.775x0.352273)
```



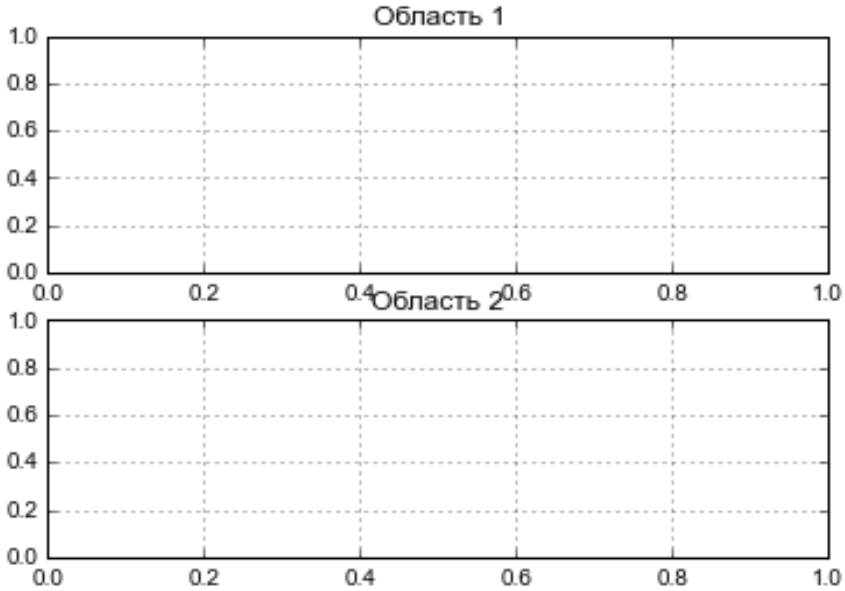
Аналогічного результату можна домогтися і "в лоб", тобто самостійно задати кордону потрібних областей. Для прямого виділення положення областей на малюнку потрібно використовувати метод fig.add_axes() . На малюнку в відносних координатах (від 0 до 1) можна задати кілька (скільки завгодно) областей, на кожній з яких можна буде малювати графіки та діаграми (не забуваємо про карти і зображення-фотографії). Кожна область задається окремо і вимагає вказати параметри кордону в вигляді списку [ліва,

нижня, ширина, висота]. Варто зазначити, що цей метод дозволяє задати багато важливих настройки створюваної області малювання і зокрема тип області малювання (якщо параметр polar=True або projection='polar' , то область малювання буде не прямокутної, а кругом).

In [3]:

```
# Приклад
import matplotlib.pyplot as plt

fig = plt.  figure ()

# У прикладі для завдання кордонів областей була використана інформація
# Про межі subplots з попереднього прикладу

ax1 = fig.  add_axes ([0.125, 0.547727, 0.775, 0.352273])
ax1.  set_title (u 'Область 1')
ax2 = fig.  add_axes ([0.125, 0.125, 0.775, 0.352273])
ax2.  set_title (u 'Область 2')

# Дізнаємося координати областей, які займають subplots
print u 'Область 1', ax1
print type (ax1)
print u 'Область 2', ax2

# Намалюємо в кожному subplot лінію сітки
for ax in fig.  axes:
    ax.  grid (True)

save ( 'pic_7_1_2', fmt = 'png')
save ( 'pic_7_1_2', fmt = 'pdf')

plt.  show ()
```

```
Область 1 Axes (0.125,0.547727; 0.775x0.352273)
<Class 'matplotlib.axes._axes.Axes'>
Область 2 Axes (0.125,0.125; 0.775x0.352273)
```



У прикладах області ax1 - це екземпляри різних класів:
ax1 axes -> matplotlib.axes._axes.Axes

ax1 subplots -> matplotlib.axes._subplots.AxesSubplot

Якщо ви хочете змінити розмір і розміщення примірника subplot після того, як він був створений, то можна використовувати метод ax.set_position() .

Приклад: ax.set_position([0.1,0.1, 0.5, 0.5]) .

Близьке розташування областей малювання

Якщо створювати області малювання за допомогою методу fig.add_axes() , то створені області, які мають загальне місце на малюнку, будуть перекривати один одного. Так як в matplotlib використовується ідеологія малювання поточної області ("current axis"), то останнім намальоване зображення одного рівня (області, лінії і т.д.) буде перекривати попередні.

Знову создавамая область малювання типу subplot вписується в малюнок в межах, визначених відповідними параметрами rcParams:

rcParams['figure.subplot.left'] = 0.125 # ліва межа
rcParams['figure.subplot.right'] = 0.9 # права межа
rcParams['figure.subplot.bottom'] = 0.125 # нижня межа
rcParams['figure.subplot.top'] = 0.9 # верхня межа

При використання методів fig.add_subplot() або plt.subplots() за замовчуванням залишається деякий вільний простір між створюваними областями. Змінюючи розмір цих "буферів" за допомогою відповідних параметрів настройки rcParams, можна домогтися ефекту з'єднання або склеювання.

rcParams['figure.subplot.hspace'] = 0.2 # визначає сукупне вертикальне відстань між subplots

rcParams['figure.subplot.wspace'] = 0.2 # визначає сукупне горизонтальне відстань між subplots

Для аналогічної настройки можна скористатися методом pyplot plt.tight_layout() . Даний метод дозволяє "навести красу" одним рядком. Часто це дуже економить час від "вилизування" дріб'язкових графіків до пристойного вигляду. Метод має чотири параметри:

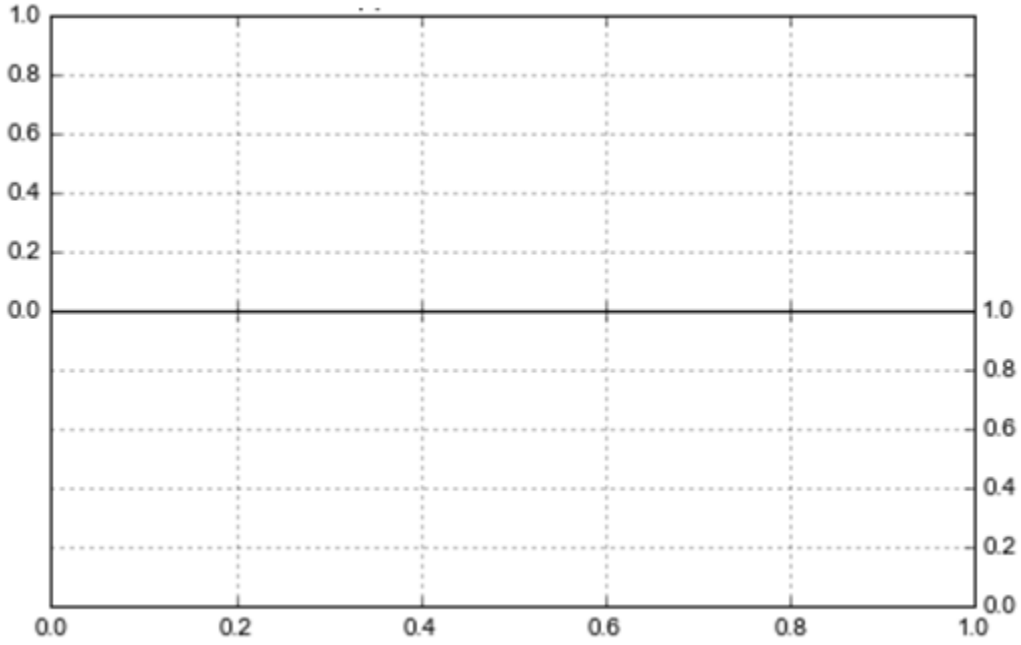
- **pad** - відстань між краями малюнка Figure і краями subplots, виражене у вигляді частки fontsize (див. rcParams ['font.size'])
- **h_pad** - сукупне відстань по вертикалі між subplots. За замовчуванням має значення pad_inches (rcParams ['savefig.pad_inches'])
- **w_pad** - відстань по горизонталі між краями сусідніх subplots. За замовчуванням має значення pad_inches (rcParams ['savefig.pad_inches'])
- **rect** - якщо заданий, то це прямокутник (лівий, низ, правий, верх) у відносних координатах figure, в який будуть вписані всі області subplots (включаючи підписи). За замовчуванням (0, 0, 1, 1).

У сукупності нижні чотири параметри - це ті ж самі параметри, які використовуються в якості вхідних для параметра rect в методі plt.tight_layout() . Даним методом можна визначати взаємне (НЕ але окреме) становище subplots і зрушувати їх, як віддаляючи їх один від одного, так і зближуючи (негативні значення h_pad і w_pad).

In [4]:

```
# Приклад

import matplotlib.pyplot as plt

fig = plt.  figure ()

ax1 = fig.  add_subplot (211)
ax1.  set_title (u 'Дві області злиплися')
ax2 = fig.  add_subplot (212)
```

```
# Щоб підписи осей координатних сіток не зливалися разнесём їх
# В різні боки - одну залишимо зліва, а іншу винесемо направо
ax1.  tick_params (axis = 'x', labelbottom = 'off', labeltop = 'off')
ax2.  tick_params (axis = 'y', labelleft = 'off', labelright = 'on', left = False, right = True)
# Дізнаємося координати областей, які займають subplots
print u 'Область 1', ax1
print type (ax1)
print u 'Область 2', ax2

# Намалюємо в кожному subplot лінію сітки

for ax in fig.  axes:
    ax.  grid (True)

# Параметр підібраний емпірично, на око.  це кепсько

plt.  tight_layout (h_pad = - 0.88)

save ( 'pic_7_2_1', fmt = 'png')
save ( 'pic_7_2_1', fmt = 'pdf')

plt.  show ()
 Область 1 Axes (0.125,0.547727; 0.775x0.352273)
<Class 'matplotlib.axes._subplots.AxesSubplot'>
Область 2 Axes (0.125,0.125; 0.775x0.352273)
```

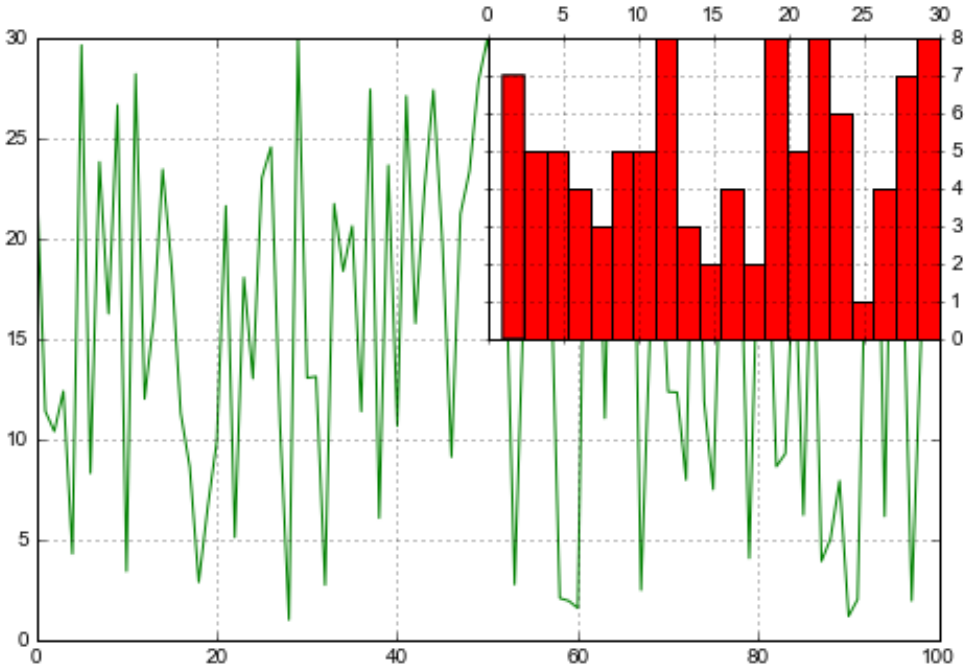


У прикладі нижче область ax2 створена пізніше, ніж ax1. Значить саме ax2 буде перекривати область ax1. Якщо поміняти місцями ax1 і ax2 (рядки коду, де вони створюються, а не де на цих областях малюються лінії), то область з блакитною лінією буде перекрита і ми її взагалі не побачимо.

In [5]:

```
# Приклад 7.2.2

import matplotlib.pyplot as plt
```

```
import numpy as np

fig = plt.  figure ()

N = 100
x = np.  arange (N)
y = np.  random.  random (N) * 30.

# Область ax1 вималюється першої
ax1 = fig.  add_axes ([0., 0., 1.0, 1.0])
# Область ax2 перекриє область ax1 і закриє її частина
ax2 = fig.  add_axes ([0.5, 0.5, 0.5, 0.5])

rect0 = [0,0, 0.0]

ax1.  plot (rect0, 'r')
ax1.  plot (x, y, 'green')

ax2.  plot (rect0, 'c')
ax2.  hist (y, 20, edgecolor = 'k', facecolor = 'r')
ax2.  tick_params (axis = 'y', which = 'major', direction = 'inout',
            left = True, right = True, labelleft = False, labelright = True)
ax2.  tick_params (axis = 'x', which = 'major', direction = 'inout',
            bottom = True, top = True, labelbottom = False, labeltop = True)

print u 'Область 1', ax1
print type (ax1)
print u 'Область 2', ax2

for ax in fig.  axes:
    ax.  grid (True)

save ( 'pic_7_2_2', fmt = 'png')
save ( 'pic_7_2_2', fmt = 'pdf')

plt.  show ()
 Область 1 Axes (0,0; 1x1)
<Class 'matplotlib.axes._axes.Axes'>
Область 2 Axes (0.5,0.5; 0.5x0.5)
```



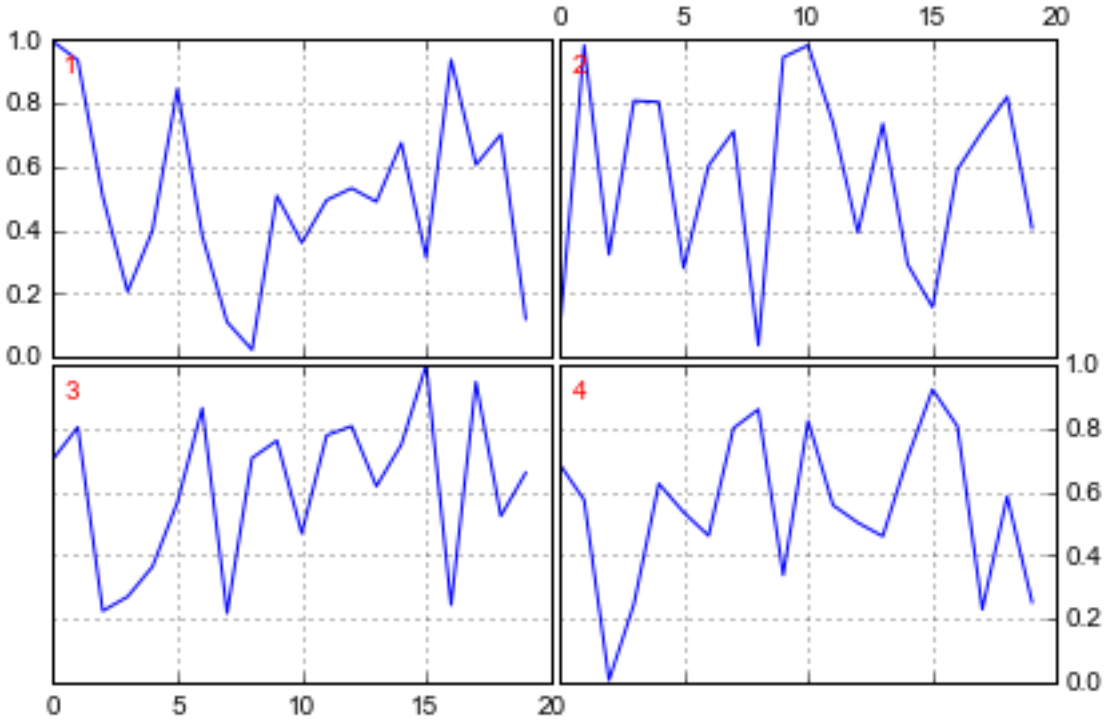
Автоматизоване створення мультіокон

Для створення безлічі областей зручно не просто додавати їх на малюнок послідовно, по одному, а всього одним рядком розбити малюнок на кілька областей малювання. Це дозволяє зробити, наприклад, метод plt.subplots() , який вимагає вказати число рядків і стовпців створюваної таблиці кожна з осередків якої і є об'єкт-екземпляр subplots. Метод повертає об'єкт типу figure і масив з створених subplots. Їх можна перебрати в циклі "в лоб", але краще користуватися перебором зі списку fig.axes, куди автоматично додаються всі області малювання поточного малюнка fig.

In [6]:

```
# Приклад

import numpy as np
import matplotlib.pyplot as plt

fig, subplots = plt.  subplots (nrows = 2, ncols = 2, sharex = True, sharey = True)

x = np.  arange (20)

i = - 1
for ax in fig.  axes:
    i + = 1
    y = np.  random.  rand (np. size (x))
    ax.  grid (True)
    ax.  text (0.5, 0.9, str (i + 1), color = 'red')
    ax.  plot (x, y)
    if ((i + 1)% 2 == 0):
        ax.  tick_params (axis = 'y', labelleft = 'off', labelright = 'on', left = False, right = True)
    if ((i == 0) or (i == 1)):
        ax.  tick_params (axis = 'x', labelbottom = 'off', labeltop = 'off', left = False, right = True)
    if ((i == 1) or (i == 2)):
        ax.  tick_params (axis = 'y', labelleft = 'off', labelright = 'off', left = False, right = False)
    if (i == 3):
        ax.  tick_params (axis = 'x', labelbottom = 'off', labeltop = 'off')
    if (i == 1):
```

```
        ax.  tick_params (axis = 'x', labelbottom = 'off', labeltop = 'on')
    if (i == 0):
        ax.  tick_params (axis = 'y', left = True, right = False)
# Параметри підібрані емпірично, на око
plt.  tight_layout (h_pad = - 0.15, w_pad = - 0.2)
save ( 'pic_7_3', fmt = 'png')
save ( 'pic_7_3', fmt = 'pdf')

plt.  show ()
```



Мультіокна різних розмірів. Gridspec

Методи fig.add_subplot() і plt.subplots() дозволяють розбити малюнок на рівні за розмірами області і в цьому полягає одна з їх основних переваг. О'єдініть осередку subplots в більш складні конструкції дозволяє метод plt.subplot2grid() , який також іменується як метод Gridspec.

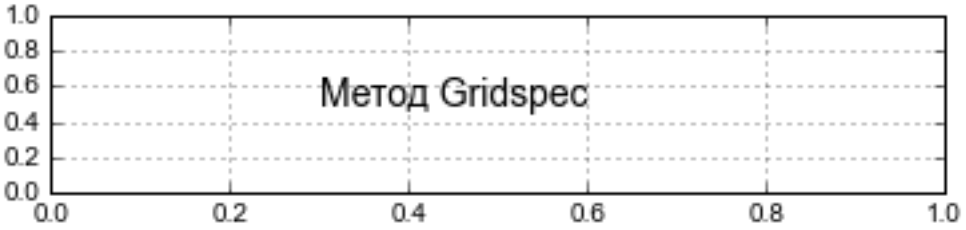
Цей метод дозволяє налаштувати індивідуальні розміри створюваних subplots. Основний принцип методу - об'єднання осередків в більш довгі або високі плитки-області. Таким чином, екземпляри subplots можуть бути різних форм і розмірів.

Створення осередків здійснюється за допомогою методу subplot2grid() , якому необхідно задати число рядків і стовпців у вигляді кортежу, а також розташування осередків у вигляді кортежу номерів від 0 до максимального числа осередків N-1. Тобто для створення таблиці 2x2 потрібно задати розмір (2,2) і розташування в межах від (0,0) до (1,1).

In [7]:

```
 # Приклад

import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

# 3,3 - три стовпці і три осередки

ax = plt.  subplot2grid ((3, 1), (0, 0))
```

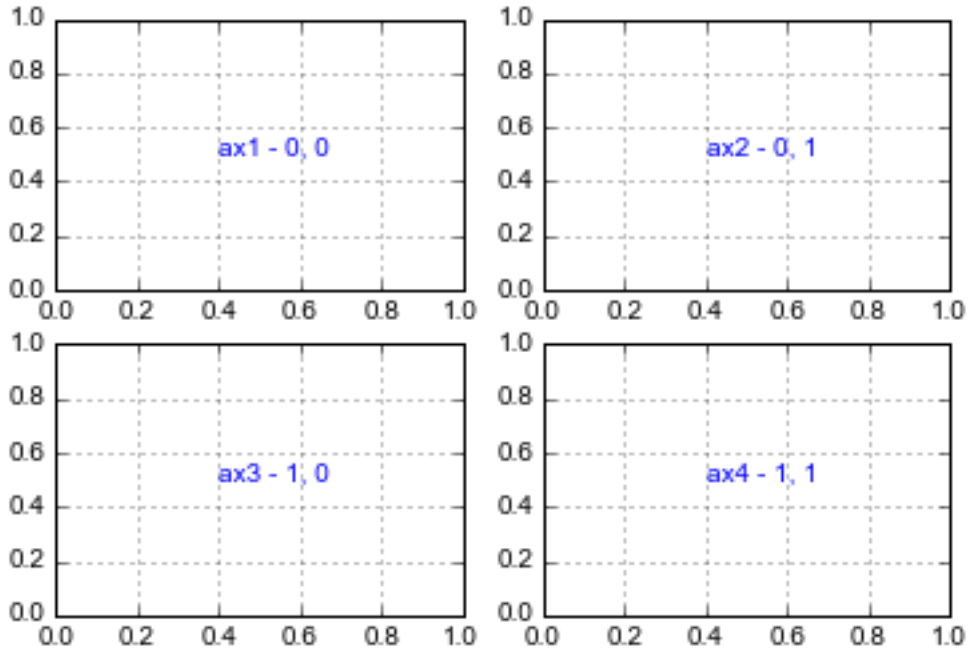
```
print type (ax)

# Це запис еквівалентна більш детальної записи

gs = gridspec.  GridSpec (3, 1)

ax = plt.  subplot (gs [0, 0])
ax.  grid (True)
# Положення тексту задано в відносних координатах
ax.  text (0.3, 0.5, u 'Метод Gridspec', fontsize = 14, transform = ax. transAxes)

save ( 'pic_7_4_1', fmt = 'png')
save ( 'pic_7_4_1', fmt = 'pdf')

plt.  show ()
 <Class 'matplotlib.axes._subplots.AxesSubplot'>
```



In [8]:

 # Приклад Створення декількох областей. Приклад завдання сітки-таблиці (grid) і розташування (loc)

```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
fig = plt.  figure ()
ax1 = plt.  subplot2grid ((2, 2), (0, 0))
ax2 = plt.  subplot2grid ((2, 2), (0, 1))
ax3 = plt.  subplot2grid ((2, 2), (1, 0))
ax4 = plt.  subplot2grid ((2, 2), (1, 1))
i = - 1
jj = [0, 0, 1, 1]
kk = [0, 1, 0, 1]
for ax in fig.  axes:
   i + = 1
   stext = 'ax% d -% d,% d'% (i + 1, jj [i], kk [i])
   ax.  text (0.4, 0.5, stext, color = 'b')
   ax.  grid (True)

save ( 'pic_7_4_1', fmt = 'png')
save ( 'pic_7_4_1', fmt = 'pdf')
plt.  show ()
```



Головною перевагою даного методу є те, що одержувані subplots можуть бути різних форм і розмірів. При цьому конфігурація задається у вигляді об'єднання осередків, а не через вказівку явних кордонів, як це можна було б зробити за допомогою методу ax.add_axes() .

Для об'єднання осередків при створенні таблиці-сітки потрібно додати наступні параметри в метод subplot2grid:

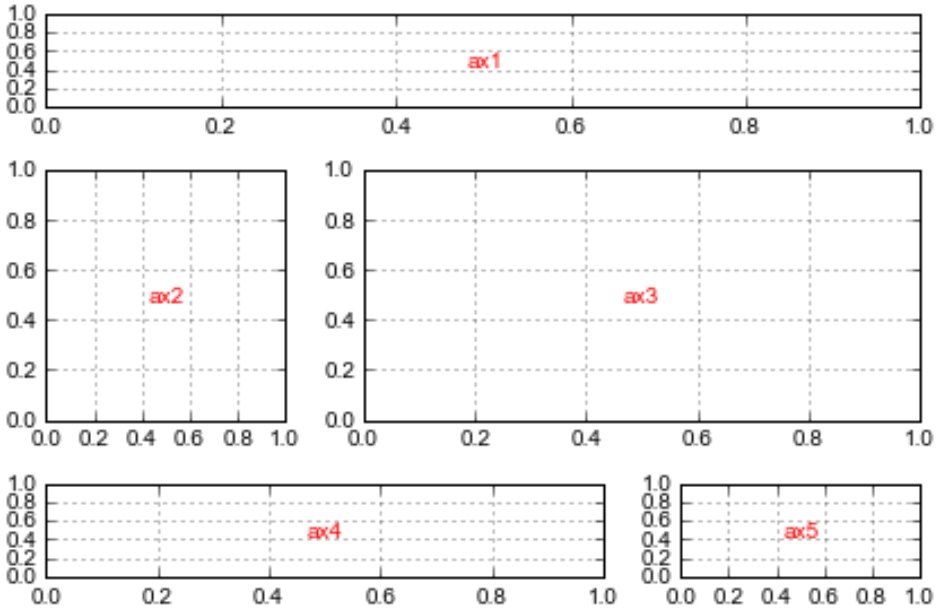
- **colspan** - об'єднати стовпи;
- **rowspan** - об'єднати рядки.

Значення відповідних параметрів показує скільки осередків сітки об'єднати. Причому відлік ведеться від точки локалізації осередку включаючи її. Якщо потрібно об'єднати і стовпці і рядки, то вказуються обидва параметри одночасно.

In [9]:

```
# Приклад

import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

fig = plt.  figure ()

# Сітка складається з 4 рядків і 3 стовпців.  Перша осередок має номер (0,0),
# А остання - (3,2)
egrid = (4, 3)
ax1 = plt.  subplot2grid (egrid, (0, 0), colspan = 3)
ax2 = plt.  subplot2grid (egrid, (1, 0), rowspan = 2)
ax3 = plt.  subplot2grid (egrid, (1, 1), rowspan = 2, colspan = 2)
ax4 = plt.  subplot2grid (egrid, (3, 0), colspan = 2)
ax5 = plt.  subplot2grid (egrid, (3, 2))

for i, ax in enumerate (fig. axes):
    ax.  text (0.5, 0.5, "ax% d"% (i + 1), va = "center", ha = "center",
          color = 'red', transform = ax.  transAxes)
    ax.  grid (True)

# Налаштування відстаней між межами створених subplots
```

```
plt.  tight_layout ()

save ( 'pic_7_4_3', fmt = 'png')
save ( 'pic_7_4_3', fmt = 'pdf')

plt.  show ()
```



Створення мультивіконних малюнків - це завжди компроміс між зручністю подання інформації (все в одному місці) і читаністю самого малюнка. Перевантажити малюнок використовуючи перераховані вище методи простіше простого, якщо загальне число осередків перевищило число 4. Тому зловживати мультіоknами не потрібно. Розмістити кілька графіків на малюнку - будь ласка! Але якщо на кожній з 4 областей буде по 3-4 графіка, то це вже перебір. Вчіться виділяти головне, те, що ви точно хочете показати, що вигідно і безпрограшно підтверджує ваші тези.

Важливим є також ясність підписів. При створенні малюнка з безліччю діаграм підписи до осей стають або нечитабельним, або виглядають неохайно. На слайді або лекції це куди не йшло, але в науковий журнал таке не пошлеш! Потрібно стежити за читаністю легенд і розміщувати їх так, щоб вони не заважали читання малюнка. При цьому не потрібно дублювати інформацію. Якщо графіки розташовані один під одним і у них єдина шкала по осі абсцис, то на деяких областях можна прибрати підписи залишивши допоміжну сітку.

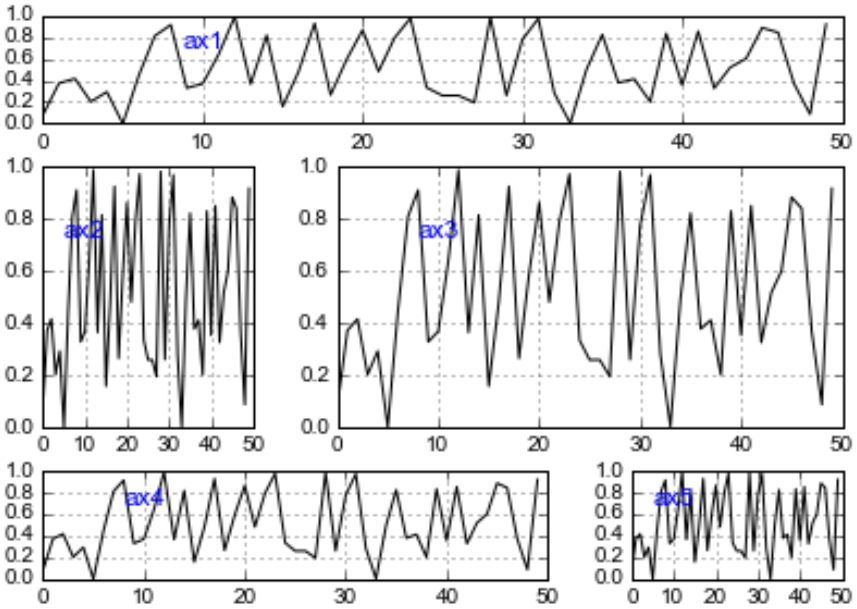
Одним словом треба вміти рухати створені тим чи іншим способом області, якщо підписи до них або до осей вилазять або перекриваються. Найпростішим є метод **plt.tight_layout()** . Він самостійно визначає параметри малюнка так, щоб всі об'єкти на ньому виглядали добре. На жаль, він погано ладнає одночасно з методом fig.add_axes() .

Метод **fig.subplots_adjust()** дозволяє оновити SubplotParams (за замовчуванням визначає їх з rcParams, якщо жоден аргумент методу не був визначений) і перевизначити параметри розташування subplots. Також можна безпосередньо визначати параметри розташування subplots через **matplotlib.rcParams** .

In [10]:

```
# Приклад

import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from matplotlib import rcParams
```

```
# Ліва межа subplots на малюнку
rcParams [ 'figure.subplot.left'] = 0.1
# Права межа subplots на малюнку
rcParams [ 'figure.subplot.right'] = 0.95
# Нижня межа subplots на малюнку
rcParams [ 'figure.subplot.bottom'] = 0.05
# Верхня межа subplots на малюнку
rcParams [ 'figure.subplot.top'] = 0.95

# Зміна параметрів wspace і hspace методом rcParams

# Загальна висота (вертикаль), виділена для вільного простору між subplots
rcParams [ 'figure.subplot.hspace'] = 0.2
# Загальна ширина (горизонталь), виділена для вільного простору між subplots
rcParams [ 'figure.subplot.wspace'] = 0.2

N = 50
x = np.  arange (N)
y = np.  random.  rand (np. size (x))

fig = plt.  figure ()

# Сітка складається з 4 рядків і 3 стовпців.  Перша осередок має номер (0,0),
# А остання - (3,2)
egrid = (4, 3)
ax1 = plt.  subplot2grid (egrid, (0, 0), colspan = 3)
ax2 = plt.  subplot2grid (egrid, (1, 0), rowspan = 2)
ax3 = plt.  subplot2grid (egrid, (1, 1), rowspan = 2, colspan = 2)
ax4 = plt.  subplot2grid (egrid, (3, 0), colspan = 2)
ax5 = plt.  subplot2grid (egrid, (3, 2))

for i, ax in enumerate (fig. axes):
      ax.  plot (x, y, 'k')
      ax.  text (0.2, 0.75, "ax% d"% (i + 1), va = "center", ha = "center",
            color = 'blue', transform = ax.  transAxes, fontsize = 'large')
      ax.  grid (True)

# Зміна параметрів wspace і hspace методом fig.subplots_adjust ()
fig.  subplots_adjust (wspace = 0.4, hspace = 0.4)

save ( 'pic_7_5', fmt = 'png')
save ( 'pic_7_5', fmt = 'pdf')

plt.  show ()
```


ax1
ax2
ax3
ax4
ax5

ТЕМА 14: ШТУЧНИЙ ІНТЕЛЕКТ НА МОВІ PYTHON.

За останні роки комп'ютерний зір набрало популярність і виділилося в окремий напрямок. Розробники створюють нові додатки, якими користуються по всьому світу.

Навіть технологічні гіганти готові ділитися новими відкриттями та інноваціями з усіма, щоб технології не залишалися привілеєм багатих.

Одна з таких технологій - розпізнавання осіб. При правильному і етичному використанні ця технологія може застосовуватися в багатьох сферах життя.

Розпізнавання осіб: потенційні сфери застосування

Наведу кілька потенційних сфер застосування технології розпізнавання осіб.

Розпізнавання облич в соцмережах. Facebook замінив присвоєння тегів зображень вручну на автоматично генеруються пропозиції тегів для кожного зображення, що завантажується на платформу. Facebook використовує простий алгоритм розпізнавання осіб для аналізу пікселів на зображенні і порівняння його з відповідними користувачами.

Розпізнавання осіб у сфері безпеки. Простий приклад використання технології розпізнавання осіб для захисту особистих даних - розблокування смартфона «по обличчю». Таку технологію можна впровадити і в пропускну систему: людина дивиться в камеру, а вона визначає дозволити йому увійти чи ні.

Розпізнавання облич для підрахунку кількості людей. Технологію розпізнавання осіб можна використовувати при підрахунку кількості людей, які відвідують будь-який захід (наприклад, конференцію або концерт). Замість того щоб вручну підраховувати учасників, ми встановлюємо камеру, яка може захоплювати зображення осіб учасників і видавати загальна кількість відвідувачів. Це допоможе автоматизувати процес і заощадити час.

Налаштування системи: вимоги до апаратного та програмного забезпечення

Розглянемо, як ми можемо використовувати технологію розпізнавання осіб, звернувшись до доступним нам інструментам з відкритим вихідним кодом.

Я використовував такі інструменти, які рекомендую вам:

- Веб-камера (Logitech C920) для побудови моделі розпізнавання осіб в реальному часі на ноутбуці Lenovo E470 ThinkPad (Core i5 7th Gen). Ви також можете використовувати вбудовану камеру свого ноутбука або відеокамеру з будь-якої зручної системою для аналізу відео в режимі реального часу замість тих, які використовував я.
- Переважно використовувати графічний процесор для більш швидкої обробки відео.
- Ми використовували операційну систему Ubuntu 18.04 з усім необхідним ПЗ.

Перш ніж приступити до побудови нашої моделі розпізнавання осіб, розберемо ці пункти більш докладно.

Крок 1: Налаштування апаратного забезпечення

Перевірте, чи правильно налаштована камера. З Ubuntu це зробити просто: подивіться, упізнано чи пристрій операційною системою. Для цього виконайте наступні кроки:

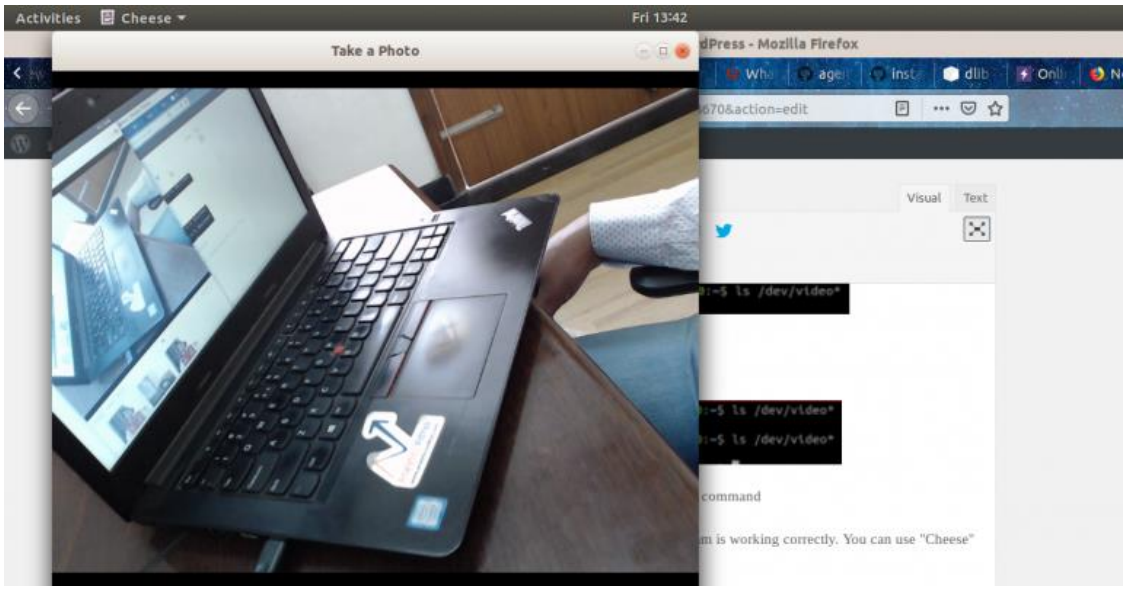
1. Перш ніж підключити веб-камеру до ноутбука, перевірте всі підключені відео пристрої, надрукувавши в командному рядку ls /dev/video* . В результаті вийде список всіх відео пристроїв, підключених до системи.

```
analyticsvidhya@analyticsvidhya-ThinkPad-E470:~$ ls /dev/video*
/dev/video0
```

2. Підключіть веб-камеру і задайте команду знову. Якщо веб-камера підключена правильно, новий пристрій буде відображено в результаті виконання команди.

```
analyticsvidhya@analyticsvidhya-ThinkPad-E470:~$ ls /dev/video*
/dev/video0
analyticsvidhya@analyticsvidhya-ThinkPad-E470:~$ ls /dev/video*
/dev/video0  /dev/video1
```

3. Також ви можете використовувати ПО веб-камери для перевірки її коректної роботи. В Ubuntu для цього можна використовувати програму «Cheese».



Крок 2: Налаштування програмного забезпечення

Крок 2.1: Установка Python

Крок 2.2: Установка OpenCV

OpenCV - бібліотека з відкритим кодом, яка призначена для створення додатків комп'ютерного зору. Установка OpenCV проводиться за допомогою pip :

```
pip3 install opencv-python
```

Крок 2.3: Встановіть face_recognition API

Ми будемо використовувати face_recognition API , який вважається самим простим API для розпізнавання осіб на Python у всьому світі. Для установки використовуйте:

```
pip install dlib pip install face_recognition
```

впровадження

Після настройки системи переходимо до впровадження. Для початку, ми створимо програму, а потім пояснимо, що зробили.

Створіть файл face_detector.py і потім скопіюйте наведений нижче код:

```
# import libraries
import cv2
import face_recognition

# Get a reference to webcam
video_capture = cv2.VideoCapture("/dev/video1")

# Initialize variables
face_locations = []

while True:

# Grab a single frame of video

ret, frame = video_capture.read()


# Convert the image from BGR color (which OpenCV uses) to RGB color (which face_recognition uses)

rgb_frame = frame[:, :, ::-1]


# Find all the faces in the current frame of video

face_locations = face_recognition.face_locations(rgb_frame)


# Display the results

for top, right, bottom, left in face_locations:

# Draw a box around the face

cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)


# Display the resulting image

cv2.imshow('Video', frame)


# Hit 'q' on the keyboard to quit!

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Release handle to the webcam
video_capture.release()
cv2.destroyAllWindows()
```

Потім запустіть цей файл Python, надрукувавши:

python face_detector.py

Якщо все працює правильно, відкриється нове вікно з запущеним функцією розпізнавання осіб в реальному часі.

Підіб'ємо підсумки і пояснимо, що зробив наш код:

Спочатку ми **вказали апаратне забезпечення,** на якому буде проводитися аналіз відео.

1. Далі зробили **захоплення відео** в реальному часі кадр за кадром.
2. Потім **обробили кожен кадр** і **витягли місцезнаходження всіх осіб** на зображенні.
3. У підсумку, **відтворили ці кадри в формі відео** разом з вказівкою на те, де розташовані особи.

Приклад застосування технології розпізнавання осіб

На цьому все найцікавіше не закінчується. Ми зробимо ще одну класну річ: створимо повноцінний приклад застосування на основі коду, наведеного вище. Внесемо невеликі зміни в код, і все буде готово.

Припустимо, що ви хочете створити автоматизовану систему з використанням відеокамери для відстеження, де спікер перебуває в даний момент часу. Залежно від його положення, система повертає камеру так, що спікер завжди залишається в центрі кадру. Перший крок - створіть систему, яка ідентифікує людини або людей на відео і фокусується на місцезнаходження спікера.

Розберемо, як це зробити. Як приклад я вибрав відео на YouTube з виступом спікерів конференції «DataHack Summit 2017».

Спочатку імпортуємо необхідні бібліотеки:

import cv2 import face_recognition

Потім зчитуємо відео і встановлюємо довжину:

input_movie = cv2.VideoCapture("sample_video.mp4") length = int(input_movie.get(cv2.CAP_PROP_FRAME_COUNT))

Після цього створюємо файл виведення з необхідним дозволом і швидкістю передачі кадрів, аналогічної тій, що була в файлі введення.

Завантажуємо зображення спікера в якості зразка для розпізнання його на відео:

image = face_recognition.load_image_file("sample_image.jpeg") face_encoding = face_recognition.face_encodings(image)[0] known_faces = [face_encoding,]

Закінчивши, запускаємо цикл, який буде:

- Витягувати кадр з відео.

- Знаходити все обличчя і ідентифікувати їх.
- Створювати нове відео, яке буде поєднувати в собі оригінал кадру із зазначенням місцезнаходження особи спікера з підписом.

Подивимося на код, який буде це виконувати:

```
# Initialize variables
face_locations = []
face_encodings = []
face_names = []
frame_number = 0

while True:

# Grab a single frame of video

ret, frame = input_movie.read()

frame_number += 1


# Quit when the input video file ends

if not ret:

break


# Convert the image from BGR color (which OpenCV uses) to RGB color (which
face_recognition uses)

rgb_frame = frame[:, :, ::-1]

# Find all the faces and face encodings in the current frame of video
face_locations = face_recognition.face_locations(rgb_frame, model="cnn")
face_encodings = face_recognition.face_encodings(rgb_frame, face_locations)

face_names = []
for face_encoding in face_encodings:
# See if the face is a match for the known face(s)
match = face_recognition.compare_faces(known_faces, face_encoding, tolerance=0.50)

name = None
if match[0]:
name = "Phani Srikant"

face_names.append(name)

# Label the results

for (top, right, bottom, left), name in zip(face_locations, face_names):
if not name:
```

```
continue

      # Draw a box around the face
cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

# Draw a label with a name below the face

cv2.rectangle(frame, (left, bottom - 25), (right, bottom), (0, 0, 255), cv2.FILLED)
      font = cv2.FONT_HERSHEY_DUPLEX

cv2.putText(frame, name, (left + 6, bottom - 6), font, 0.5, (255, 255, 255), 1)

# Write the resulting image to the output video file

print("Writing frame {} / {}".format(frame_number, length))

output_movie.write(frame)

# All done!
input_movie.release()
cv2.destroyAllWindows()
```

ТЕМА 15: МАШИННЕ НАВЧАННЯ НА PYTHON.

Машинне навчання - це спроба наділити комп'ютери здатністю навчитися виконання певних завдань без безпосереднього програмування цих завдань. Це здійснюється за рахунок того, що обчислювальної системі передається інформація, яку вона перетворює в моделі прийняття рішень, які використовуються для прогнозування результатів в подальшому.

В даному уроці ми поговоримо як про самому машинному навчанні, так і про деякі засадничі поняття, без яких неможливо починати з ним знайомство. Крім того, ми напишемо на мові Python кілька прикладів алгоритмів ймовірнісної ідентифікації елементів або подій.

Знайомство з машинним навчанням

Машинне навчання - це технологія, метою якої є навчання на основі досвіду. Як приклад можна уявити людину, яка вчиться грати в шахи, просто спостерігаючи, як це роблять інші. Подібним чином і комп'ютери можуть бути запрограмовані шляхом надання їм інформації, завдяки якій вони навчаються, набуваючи здатність з високою ймовірністю ідентифікувати елементи або їх ознаки.

Давайте уявимо, що нам необхідно написати програму, яка зможе визначити, чи є той чи інший фрукт апельсином або лимоном. Може здатися, що написати подібний алгоритм досить просто, і він буде видавати необхідний результат, але варто зауважити, що ефективність подібної програми знижується при роботі з великим об'ємом даних. Ось в таких ситуаціях і потрібно машинне навчання.

Існують різні етапи в машинному навчанні:

1. збір даних
2. сортування даних
3. аналіз даних
4. вироблення алгоритму
5. перевірка виробленого алгоритму
6. використання алгоритму для подальших висновків

У машинному навчанні для пошуку закономірностей використовуються різні алгоритми, які поділяються на дві групи:

- кероване навчання
- самостійне навчання

кероване навчання

Методика керованого навчання передбачає вироблення комп'ютером здатності розпізнавати елементи на основі наданої добірки зразків. Комп'ютер вивчає зразки і виробляє здатність розпізнавати нові дані на основі вивченої інформації.

Наприклад, можна навчити комп'ютер фільтрувати повідомлення спаму на основі раніше отриманої інформації.

Кероване навчання застосовувалося в багатьох додатках. Наприклад в Facebook - для пошуку зображень, що підходять під певний опис. Саме за рахунок цього в Facebook сьогодні можна здійснювати пошук зображень за словами, що описує вміст фотографії. Завдяки тому, що сайт цієї соціальної мережі має базу даних зображень і їх заголовків, алгоритм здатний з певним ступенем точності знаходити фотографії та зіставляти їх вміст з описом.

Кероване навчання включає тільки два етапи:

- навчання
- перевірка

Деякі алгоритми керованого навчання включають:

- схеми прийняття рішень
- методи опорних векторів (схожі алгоритми навчання)
- імовірнісний класифікатор на основі теореми Байєса

- метод k-найближчих сусідів
- лінійну регресію

приклад

Ми напишемо просту програму для демонстрації того, як працює кероване навчання. Для цього ми будемо використовувати бібліотеку Sklearn і мову Python. Sklearn - це бібліотека машинного навчання для мови програмування Python, яка надає безліч можливостей, таких як багатоступінчастий аналіз, регресія і алгоритми кластеризації.

Крім того Sklearn добре взаємодіє з бібліотеками NumPy і SciPy.

установка Sklearn

Інструкція по установці Sklearn пропонує дуже простий спосіб установки для різних платформ. Для роботи бібліотеки потрібно кілька залежностей:

- Python (> = 2.7 or> = 3.3),
- NumPy (> = 1.82)
- SciPy (> = 0.13.3)

Якщо ці залежності вже встановлені, то можна встановити Sklearn, просто виконавши команду:

1

pip install -U scikit-learn pip install -U scikit-learn

Більш простим способом є установка Anaconda. Даний пакет сам встановить всі залежності, так що вам не доведеться встановлювати їх по одній.

Щоб перевірити, що Sklearn працює коректно, просто імпортуйте цю бібліотеку в інтерпретаторі мови Python:

import sklearn import sklearn

Якщо це не викликало помилок, значить все готово до роботи.

Після того, як ми розібралися з установкою, давайте повернемося до нашого завдання. Припустимо, ми хочемо навчитися розрізняти тварин. Для цього ми створимо алгоритм, який зможе визначити, чи є та чи інша тварина конем або куркою.

Спершу нам необхідно зібрати вихідні дані для кожного виду тварини. Деякі вихідні дані представлені в таблиці нижче.

Зріст (см)	**Вага, кг)**	**Температура (гр. Цельсія)**	**Назва**
18	0.6	40	Курка (0)
18	0.6	41	Курка (0)
94	600	37	Кінь (1)
94	600	38	Кінь (1)

Отримані нами вихідні дані містять деякі основні характеристики та їх значення для наших двох тварин. Чим більше вихідних даних, тим більш точними і менш упередженими будуть результати.

Грунтуючись на наявних даних, ми можемо написати алгоритм і навчити його визначати тварина на основі вивчених даних і класифікувати його як кінь або курку. Тепер ми напишемо алгоритм, який буде виконувати поставлене завдання.

Як видно з прикладу, ви змусили алгоритм вивчити всі характеристики і назви двох тварин, і знання про ці дані далі використовуються при ідентифікації нових тварин.

Самостійне навчання

При самостійному навчанні ваша машина отримує тільки набір вступних даних. Після чого машина сама буде здатна визначити взаємозв'язку між введеними даними і будь-якими іншими приблизною даними. На відміну від керованого навчання, при якому машині надаються деякі перевірочні дані для навчання, самостійне навчання передбачає, що комп'ютер сам знайде закономірності і взаємозв'язку між різними наборами даних.

Самостійне навчання може далі підрозділятися на:

- кластеризацию
- асоціювання

Кластеризація: кластеризації називають органічне групування даних. Наприклад, можна згрупувати купівельні переваги клієнтів і використовувати їх в рекламі, показуючи тільки ті оголошення, які відповідають їх покупкам або перевагам.

Асоціювання: асоціювання - це визначення правил, що описують великі набори ваших даних. Такий вид навчання може застосовуватися при пропозиції, наприклад, різних книг одного автора або однієї категорії, будь то мотивуючі, фантастичні або освітні книги.

Деякі з популярних алгоритмів самостійного навчання включають:

- кластеризацию k-середніх
- ієрархічну кластеризацію

Самостійне навчання буде дуже важливою технологією в найближчому майбутньому. Це обумовлено тим, що в даний час існує багато необробленої інформації, яка ще не була оцифрована.

Машинне навчання може застосовуватися майже у всіх сферах нашого життя, наприклад: боротьба з шахрайством, персоналізовані стрічки новин в соціальних мережах, відповідні перевагам користувачів, фільтрація електронної пошти і шкідливих програм, прогноз погоди і навіть в сфері електронної торгівлі для прогнозування купівельних переваг клієнтів.