**Модуль 8: Можливості деяких**

**вбудованих пакетів Python**

**Робота з датою і часом (datetime)**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#%D1%80%D0%BE%D0%B1%D0%BE%D1%82%D0%B0-%D0%B7-%D0%B4%D0%B0%D1%82%D0%BE%D1%8E-%D1%96-%D1%87%D0%B0%D1%81%D0%BE%D0%BC-datetime)

Робота з датою і часом у Python реалізована у пакеті **datetime.** Основні можливості **datetime:**

* визначення поточної дати і часу;
* обчислення інтервалу між двома подіями;
* визначення дня тижня, високосного року для будь-якої дати у минулому не раніше року **datetime.MINYEAR**або в майбутньому не пізніше року **datetime.MAXYEAR**;
* порівняння дати і часу декількох подій за допомогою операторів порівняння;
* робота з часовими зонами, порівняння подій з урахуванням часових зон та переходу на літній/зимовий час;
* перетворення дати/часу в рядок і навпаки.

**datetime**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#datetime)

Щоб отримати поточну дату і час без урахування часового пояса, можна викликати метод **now()** у **datetime:**

**from datetime import datetime  
  
current\_datetime = datetime.now()  
print(current\_datetime) *# 2020-10-09 22:13:35.053819***

У результаті виклику **now()** ми отримуємо об'єкт **datetime**, у якого є ряд корисних атрибутів:

**from datetime import datetime  
  
current\_datetime = datetime.now()  
  
print(current\_datetime.year) *# 2020*  
print(current\_datetime.month) *# 10*  
print(current\_datetime.day) *# 09*  
print(current\_datetime.hour) *# 22*  
print(current\_datetime.minute) *# 32*  
print(current\_datetime.second) *# 22*  
print(current\_datetime.microsecond) *# 819366***

В об'єкта **datetime** є методи, щоб отримати дату (без часу) та час (без дати):

**from datetime import datetime  
  
current\_datetime = datetime.now()  
print(current\_datetime.date()) *# 2020-10-09*  
print(current\_datetime.time()) *# 22:13:35.053819***

Щоб створити об'єкт **datetime** з будь-якою вибраною датою, можна зробити так:

**from datetime import datetime  
  
d1 = datetime(year=2012, month=1, day=7, hour=14)  
print(d1) *# 2012-01-07 14:00:00***

Щоб дізнатися день тижня, можна скористатися методом **weekday:**

**from datetime import datetime  
  
seventh\_day\_2020 = datetime(year=2020, month=1, day=7, hour=14)  
print(seventh\_day\_2020.weekday()) *# 1***

Дні тижня у Python починаються з понеділка і він буде 0. У прикладі вище 7 Січня 2020 року було вівторком.

Щоб порівняти два об'єкти **datetime**, достатньо скористатися оператором порівняння:

**from datetime import datetime  
  
current\_datetime = datetime.now()  
  
future\_month = (current\_datetime.month % 12) + 1  
future\_year = current\_datetime.year + int(current\_datetime.month / 12)  
future\_datetime = datetime(future\_year, future\_month, 1)  
  
print(current\_datetime < future\_datetime) *# True***

**timedelta**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#timedelta)

Якщо відняти від одного **datetime** об'єкту інший, то отримаємо **timedelta** об'єкт. Він відповідає за відрізок часу між двома датами.

**from datetime import datetime  
  
seventh\_day\_2019 = datetime(year=2019, month=1, day=7, hour=14)  
seventh\_day\_2020 = datetime(year=2020, month=1, day=7, hour=14)  
  
difference = seventh\_day\_2020 - seventh\_day\_2019  
print(difference) *# 365 days, 0:00:00*  
print(difference.total\_seconds()) *# 31536000.0***

Об'єкти **timedelta** можна створювати самостійно, щоб отримати дату/час віддалену від початкової:

**from datetime import datetime, timedelta  
  
seventh\_day\_2020 = datetime(year=2020, month=1, day=7, hour=14)  
four\_weeks\_interval = timedelta(weeks=4)  
  
print(seventh\_day\_2020 + four\_weeks\_interval) *# 2020-02-04 14:00:00*  
print(seventh\_day\_2020 - four\_weeks\_interval) *# 2019-12-10 14:00:00***

Об'єкт **timedelta** можна створити, задаючи тижні, дні, години, хвилини, секунди, мілісекунди і мікросекунди:

**from datetime import timedelta  
delta = timedelta(  
 days=50,  
 seconds=27,  
 microseconds=10,  
 milliseconds=29000,  
 minutes=5,  
 hours=8,  
 weeks=2  
)**Якщо якийсь параметр не заданий, то він дорівнює 0 за замовчуванням.

Робота з часовими поясами припускає знання класів у Python і ми розглянемо її пізніше.

**timestamp**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#timestamp)

Окремо потрібно сказати про **timestamp. timestamp** — це кількість секунд, що пройшло з 00 годин 00 хвилин 1 Січня 1970 року в UTC (часовий пояс Гринвіча). Це просто прийнята константа і нічого особливого вона не означає. Просто для зручності колись почали відраховувати час в секундах з цієї миті і це виявилося дуже зручно. Адже порівняти два числа завжди легше і швидше, ніж порівняти складну структуру дат і часів.

Звичайно можна з **timestamp** отримати дату/час і навпаки:

**from datetime import datetime  
  
seventh\_day\_2020 = datetime(year=2020, month=1, day=7, hour=14)  
ts = seventh\_day\_2020.timestamp()  
print(ts) *# 1578398400.0*  
  
ts += 100\_000  
print(datetime.fromtimestamp(ts)) *# 2020-01-08 17:46:40***

**Перетворення в рядок і з рядка**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#%D0%BF%D0%B5%D1%80%D0%B5%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B2-%D1%80%D1%8F%D0%B4%D0%BE%D0%BA-%D1%96-%D0%B7-%D1%80%D1%8F%D0%B4%D0%BA%D0%B0)

Коли потрібно перетворити дату/час в рядок, ви можете скористатися функцією **str,** яка перетворить **datetime** у рядок. Але часто формат такого перетворення незручний і в Python є окрема мінімова для опису, як перетворити дату/час в рядок:

**from datetime import datetime  
  
seventh\_day\_2020 = datetime(year=2020, month=1, day=7, hour=14)  
print(seventh\_day\_2020.strftime('%A %d %B %Y')) *# Tuesday 07 January 2020***

Та ж міні-мова використовується для конвертації вже рядків в дату/час:

**from datetime import datetime  
  
s = '10 January 2020'  
print(datetime.strptime(s, '%d %B %Y')) *# 2020-01-10 00:00:00***

Деталі можна вивчити на [сторінці офіційної документації](https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes).

**Генерація випадкових чисел**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%86%D1%96%D1%8F-%D0%B2%D0%B8%D0%BF%D0%B0%D0%B4%D0%BA%D0%BE%D0%B2%D0%B8%D1%85-%D1%87%D0%B8%D1%81%D0%B5%D0%BB)

Для генерації випадкових (псевдовипадкових) чисел у Python є пакет **random.** Він досить хороший для ряду побутових завдань, але не для криптографії. На жаль, вбудований генератор псевдовипадкових чисел досить скоро починає повторюватися і не є достатньо криптостійким. Проте, для прикладних завдань поза сферою криптографії його цілком вистачає.

Отримання випадкового цілого числа з рівномірного розподілу в інтервалі між 1 та 1000 включно:

**import random  
  
random.randint(1, 1000)**

Якщо потрібно отримати випадкове число в інтервалі 0, 1 включно:

**import random  
  
random.random()**

Коли у вас є список об'єктів і вам потрібно перемішати порядок елементів в цьому списку на випадковий:

**import random  
  
fruits = ['apple', 'banana', 'orange']  
random.shuffle(fruits)  
print(fruits) *# ['banana', 'orange', 'apple']***

Якщо потрібно вибрати випадковий елемент зі списку:

**import random  
  
fruits = ['apple', 'banana', 'orange']  
print(random.choice(fruits)) *# 'banana'***Щоб вибрати N випадкових елементів зі списку:

**import random  
  
fruits = ['apple', 'banana', 'orange']  
print(random.choices(fruits, k=4)) *# ['banana', 'orange', 'orange', 'orange']***Щоб вибрати N елементів, що не повторюються, зі списку:

**import random  
  
fruits = ['apple', 'banana', 'orange']  
print(random.sample(fruits, k=2)) *# ['banana', 'orange']***Зверніть увагу, що **k** не може бути більше довжини **fruits.**

Якщо вам потрібно скористатися відмінним від рівномірного розподілу, можете вибрати один із [підтримуваних розподілів](https://docs.python.org/3/library/random.html#real-valued-distributions).

**Контроль точності обчислень (decimal)**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C-%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%96-%D0%BE%D0%B1%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D1%8C-decimal)

Комп'ютер усі обчислення робить в бінарному вигляді, а десяткові числа використовуються тільки для "спілкування" з користувачем для зручності останнього. Через це і через те, що точність обчислень в комп'ютері обмежена, виникають помилки округлення під час виконання математичних операцій.

**print(0.1 + 0.2 == 0.3) *# False*  
print(0.1 + 0.2) *# 0.30000000000000004***

Перший вираз може збити вас з пантелику, оскільки математика стверджує однозначно, що **0.1 + 0.2 = 0.3.** Але помилка округлення під час виконання обчислювальних операцій з дійсними числами у двійковій системі обчислення призводить до такої неоднозначності.

Проблема точності обчислень так і не розв'язана остаточно, і математика продовжує розвиватися в цьому напрямі.

Щоб контролювати точність обчислень більш явно, у Python є пакет **decimal.**

**from decimal import Decimal, getcontext  
  
  
getcontext().prec = 6  
Decimal(1) / Decimal(7) *# Decimal('0.142857')*  
  
getcontext().prec = 28  
Decimal(1) / Decimal(7) *# Decimal('0.1428571428571428571428571429')***

У цьому прикладі ми вирахували вираз **1 / 7** з точністю до 6 знаків після коми і до 28 знаків. Щоб встановити точність обчислення, ми скористалися функцією **getcontext**, яка повертає поточні налаштування точності, та встановили налаштування **prec** у 6 та 28 відповідно.

Об'єкти **Decimal** поводяться так само, як **float**, але їх і не можна використовувати в одному виразі разом. Виконання виразу на кшталт **Decimal(0.1) + 0.2** призведе до помилки.

Повертаючись до нашого прикладу із додаванням **0.1** та **0.2:**

**from** **decimal** **import** **Decimal,** **getcontext**  
  
  
**getcontext().prec = 6  
float(Decimal(0.1) + Decimal(0.2)) == 0.3 *# True***

На жаль, **Decimal** має ще ту особливість, що при створенні **Decimal** із **float** його точність береться максимальною для цієї платформи, а не з налаштувань **getcontext**.

Саме тому:

**Decimal(0.2) + Decimal(0.1) == Decimal(0.3) *# False***

але

**Decimal(0.2) + Decimal(0.1) == Decimal(0.3) + Decimal(0.0) *# True***

При конвертації в **float** точність береться з налаштувань **getcontext**.

**Можливості пакетів math, cmath**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#%D0%BC%D0%BE%D0%B6%D0%BB%D0%B8%D0%B2%D0%BE%D1%81%D1%82%D1%96-%D0%BF%D0%B0%D0%BA%D0%B5%D1%82%D1%96%D0%B2-math-cmath)

Для математичних обчислень у Python доданий пакет **math**. Цей пакет містить ряд часто використовуваних математичних функцій та констант:

* тригонометричні функції
  + **sin / asin**
  + **cos / acos**
  + **tan / atan**
  + **hypot,** обчислення відстані Евкліда між точками
* Перетворення кутів та радіанів:
  + **degrees**
  + **radians**
* Гіперболічні функції:
  + **sinh** **/** **asinh**
  + **cosh / acosh**
  + **tanh / atanh**
* степеневі функції:
  + **exp**
  + **pow** (перетворить аргументи в **float**, на відміну від вбудованої **pow**)
  + **sqrt**
  + **log2**
  + **log10**
  + **log** логарифм із основою **e**, якщо основа не задана або із заданою основою.
* константи:
  + **pi**
  + **e**
  + **tau**
  + **inf**, нескінченність
  + **nan**, не число

**import** **math**  
  
**math.sin(math.pi / 4) *# 0.7071067811865475*  
math.degrees(math.pi / 4) *# 45.0***

Якщо вам потрібна комплексна математика, то можна скористатися пакетом **cmath**. Він надає той самий API, але вміє працювати з комплексними числами.

**import** **cmath**  
  
**cmath.sqrt(-4) *# 2j***

**Collections**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#collections)

У Python є пакет з додатковими колекціями, які можуть дуже знадобитися розробнику-початківцю і сильно розширяти його арсенал.

**Іменовані кортежі**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#%D1%96%D0%BC%D0%B5%D0%BD%D0%BE%D0%B2%D0%B0%D0%BD%D1%96-%D0%BA%D0%BE%D1%80%D1%82%D0%B5%D0%B6%D1%96)

Використання кортежів у Python для передачі даних між обробниками — це хороша та поширена практика. Але є одна незручність у кортежів, вам необхідно пам'ятати індексацію елементів у кортежі і не плутати їх порядок. Це не завжди зручно і для ситуацій, коли в кортежі є 2 або більше елементів, такий підхід ускладнює читабельність коду:

**person = ('Mick', 'Nitch', 35, 'Boston', '01146')**

Після створення **person**там, де ви його використовуєте, вам потрібно пам'ятати, що ім'я на першому місці, а вік — на третьому. Щоб не плутатися, доведеться постійно повертатися до коду, де створюється **person**. Це незручно і спеціально для таких випадків додали іменовані кортежі:

**import collections  
  
Person = collections.namedtuple('Person', ['name', 'last\_name', 'age', 'birth\_place', 'post\_index'])  
person = Person('Mick', 'Nitch', 35, 'Boston', '01146')  
person.name *# 'Mick'*  
person.post\_index *# '01146'*  
person.age *# 35*  
person[3] *# 'Boston'***

Тепер, використовуючи **Person**, ви можете створювати кортежі, які обов'язково повинні містити 5 елементів і у таких кортежів, окрім індексів, є атрибути. Звертатися до елементів такого кортежу можна як за індексом, так і за ім'ям. За такого підходу вам достатньо один раз визначити **Person** і більше не повертатися до нього, щоб згадати, який елемент за що відповідає.

**Counter**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#counter)

Часто вам потрібно підрахувати кількість елементів у певній послідовності. Для цього зручно скористатися словником.

**student\_marks = [4, 2, 4, 6, 7, 4, 2 , 3, 4, 5, 6, 6, 7, 1, 1, 1, 3, 5]  
mark\_counts = {}  
for mark in student\_marks:  
 if mark in mark\_counts:  
 mark\_counts[mark] += 1  
 else:  
 mark\_counts[mark] = 1  
print(mark\_counts) *# {4: 4, 2: 2, 6: 3, 7: 2, 3: 2, 5: 2, 1: 3}***Таке завдання зустрічається досить часто і, щоб не писати одні й ті самі 6 рядків коду постійно, у **collections**додали спеціальний словник **Counter**:

**import collections  
  
student\_marks = [4, 2, 4, 6, 7, 4, 2 , 3, 4, 5, 6, 6, 7 , 1, 1, 1, 3, 5]  
mark\_counts = collections.Counter(student\_marks)  
print(mark\_counts) *# Counter({4: 4, 6: 3, 1: 3, 2: 2, 7: 2, 3: 2, 5: 2})***

Але на цьому корисні властивості **Counter**не закінчуються. Він може вивести елементи за частотою появи:

**import collections**  
  
**student\_marks = [4, 2, 4, 6, 7, 4, 2 , 3, 4, 5, 6, 6, 7 , 1, 1, 1, 3, 5]  
mark\_counts = collections.Counter(student\_marks)  
print(mark\_counts.most\_common(1)) *# [(4, 4)]*  
print(mark\_counts.most\_common(2)) *# [(4, 4), (6, 3)]***

Ще **Counter**може відняти кількість елементів одного **Counter**від другого поелементно:

**from collections import Counter  
  
c = Counter(a=4, b=2, c=0, d=-2)  
d = Counter(a=1, b=2, c=3, d=4)  
c.subtract(d)  
print(c) *# Counter({'a': 3, 'b': 0, 'c': -3, 'd': -6})***

**defaultdict**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#defaultdict)

Це спеціальний словник, який створює значення для ключів, яких в словнику не було за запитом. Наприклад, у вас є список слів і ви хочете розбити його на декілька списків, залежно від першої літери слова.

**words = ['apple', 'zoo', 'lion', 'lama', 'bear', 'bet', 'wolf', 'appendix']  
grouped\_words = {}  
  
for word in words:  
 char = word[0]  
 if char not in grouped\_words:  
 grouped\_words[char] = []  
 grouped\_words[char].append(word)  
  
print(grouped\_words)**

У консолі ми побачимо:

**{  
 'a': ['apple', 'appendix'],  
 'z': ['zoo'],  
 'l': ['lion', 'lama'],  
 'b': ['bear', 'bet'],  
 'w': ['wolf']  
}**

Таким чином ми можемо отримати всі слова із **words**, що починаються на якусь літеру. Подібні завдання зустрічаються досить часто. Щоб не перевіряти, чи є список на цю літеру в словнику **grouped\_words**, ми можемо скористатися **defaultdict** із **collections** та задати значенням за замовчуванням порожній список:

**from collections import defaultdict  
  
  
words = ['apple', 'zoo', 'lion', 'lama', 'bear', 'bet', 'wolf', 'appendix']  
grouped\_words = defaultdict(list)  
  
for word in words:  
 char = word[0]  
 grouped\_words[char].append(word)**

Результат виконання буде ідентичний. **defaultdict**приймає у якості аргументу функцію, яка буде використовуватися для створення значення за замовчуванням. В цьому прикладі ми використали **list**, але ви можете передати будь-яку функцію, яка викликається без аргументів.

**deque**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#deque)

Списки у Python реалізовані таким чином, що вибір елемента за індексом відбувається за константний час (дуже швидко) і додавання/видалення елементу в кінець списку теж відбувається дуже швидко. Але ось додавання елементу в будь-яке інше місце в списку змушує Python перерахувати індекси усіх елементів списку до кінця. Для великих списків це може бути дуже невигідно. Щоб швидко додавати елементи на початок списку, в Python в пакеті **collections**є така колекція як **deque**:

**from collections import deque  
  
d = deque()  
d.append('last')  
d.appendleft('first')  
d.insert(1, 'middle')  
print(d) *# deque(['first', 'middle', 'last'])*  
  
print(d.pop()) *# 'last'*  
print(d.popleft()) *# 'first'*  
print(d) *# deque(['middle'])***

Окрім додавання на початок за допомогою **appendleft**, у **deque**є і швидке видалення першого елементу **popleft**.

Ще однією особливістю **deque**є можливість обмежити розмір **deque**:

**from collections import deque  
  
d = deque(maxlen=5)  
for i in range(10):  
 d.append(i)  
  
print(d) *# deque([5, 6, 7, 8, 9], maxlen=5)***

Як видно з прикладу, нові елементи витісняють старіші, але розмір залишається незмінним.

В іншому **deque** веде себе точно як список Python.

**Comprehensions**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#comprehensions)

Часто потрібно створити колекцію і відразу заповнити її елементами. Зробити це одним виразом не можна, доводиться писати цикл. Наприклад, створимо список квадратів чисел від 1 до 5:

**sq = []  
for i in range(1, 5+1):  
 sq.append(i\*\*2)  
  
print(sq) *# [1, 4, 9, 16, 25]***

Подібні операції, які ми робимо зі змінюваними колекціями (list, dict, set) у циклі **for**, ми можемо виконати одним виразом за допомогою конструкції **comprehension**.

**comprehensions** — це синтаксична конструкція Python, створена спеціально, щоб зменшити кількість коду, коли потрібно для кожної ітерації циклу **for** додати один елемент у нову колекцію.

Суть синтаксису **comprehensions**: всередині дужок (для **list** — квадратні, для **dict** та **set** — круглі) записується вираз (тільки один), який потрібно виконати з кожним елементом циклу **for** та сам цикл, але без двокрапки у кінці.

Найпростіше зрозуміти цю конструкцію на прикладах.

**list**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#list)

Щоб записати до списку числа від 1 до 5 за допомогою **comprehensions**:

**numbers = [i for i in range(1, 5+1)]**

Попередній приклад із квадратами чисел від 1 до 5 можна переписати за допомогою **comprehension**:

**sq = [i \*\* 2 for i in range(1, 5+1)]**

**set**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#set)

Для множин ситуація абсолютно аналогічна. Збережімо множини квадратів чисел зі списку:

**numbers = [1, 4, 1, 3, 2, 5, 2, 6]  
sq = {i \*\* 2 for i in numbers}  
print(sq) *# {1, 4, 36, 9, 16, 25}***

**dict**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson08/lesson-08/#dict)

Для словників синтаксис **comprehension** трохи відрізняється, оскільки потрібно явно вказати ключ та значення:

**numbers = [1, 4, 1, 3, 2, 5, 2, 6]  
sq = {i: i \*\* 2 for i in numbers}  
print(sq)** *# {1: 1, 4: 16, 3: 9, 2: 4, 5: 25, 6: 36}*

В цьому прикладі ми записали в словник **sq** числа від 1 до 5 у якості ключів, а їх квадрати як значення. Основна відмінність за синтаксисом — це двокрапка між ключем і значенням в лівій частині виразу