

ПРАКТИЧНА РОБОТА 07

Структурне програмування мовою Python

План

1. Вбудовані структури даних у мові Python.
2. Організація Python-коду за допомогою функцій.
3. Індивідуальне завдання.

Система оцінювання

№	Тема	К-ть балів
1.	Структурне програмування мовою Python	7
2.	ІНДЗ-1	2
2.	ІНДЗ-2	2
	Всього	11

Вбудовані структури даних у мові Python

(Списки) Багато в чому списки є основою для інших вбудованих структур даних у мові Python. Зсередини вони реалізуються як С-масиви та поводять себе як масиви вказівників. У мові Python списки можуть містити як однотипні, так і різнотипні значення. Наприклад, введемо кілька списків:

```
emptyList = []          # список довільного розміру
nElementList = [None]*5  # список заданого розміру

names = ['Олександр', 'Ірина', 'Сергій', 'Вікторія', 'Владислав', 'Єлизавета', 'Олег']
something = [1, "a", "string", 1+2]
print(something)         #[1, 'a', 'string', 3]
```

Доступ до елементів списку здійснюється аналогічно до доступу до елементів масиву в мові С, проте допускається індексація як від початку списку, так і від його кінця (від'ємний індекс, починаючи з -1):

```
# доступ до елементів
print(names[2])
print(names[-2])
```

Виведеться наступне:

Сергій
Єлизавета

Вивід окремих елементів списку в консоль зазвичай здійснюється за допомогою циклів як поелементно, так і поіндексно. Поелементно:

```
# виведення елементів списку
for name in names:
    print(name)
```

У консолі з'явиться таке:

Олександр
Ірина
Сергій
Вікторія

Владислав
Єлизавета
Олег

Поіндексний вивід може відбуватись кількома способами:

```
i = 0
while i < len(names):
    print(names[i])
    i += 1
```

або

```
for i in range(len(something)):
    print(something[i])
```

де `len()` – це функція, що визначає кількість елементів переданого їй списку. Також зверніть увагу, що дана функція не обмежується тільки списками в якості вхідного параметру.

Для паралельної роботи з індексами та значеннями може бути зручно застосувати функцію `enumerate()`, яка дозволить циклу ітерувати як по індексах, так і по значенням, повертаючи кортежі (незмінювані списки у вигляді «індекс, значення»):

```
for num, name in enumerate(names, start=1):
    print("Name {}: {}".format(num, name))
```

Результатом виконання даного коду стане наступний вивід:

Name 1: Олександр
Name 2: Ірина
Name 3: Сергій
Name 4: Вікторія
Name 5: Владислав
Name 6: Єлизавета
Name 7: Олег

Формувати свій список можна на основі вже існуючих даних. Доступне використання спискових включень, які є синтаксичним цукром для компактного запису ітерування по списку за допомогою циклів. Також маємо можливість «дописати» елементи в кінець існуючого списку:

```
# формування нового списку на базі вже існуючих даних
names2 = [name+name for name in names] # спискове включення
print(names2)

extendedList = ['ПершийЕлемент']
extendedList.extend(names) # дописування списку до списку
print(extendedList)
```

На екран буде виведено таке:

```
['ОлександрОлександр', 'ІринаІрина', 'СергійСергій',
'ВікторіяВікторія', 'ВладиславВладислав',
'ЄлизаветаЄлизавета', 'ОлегОлег']
['ПершийЕлемент', 'Олександр', 'Ірина', 'Сергій', 'Вікторія',
'Владислав', 'Єлизавета', 'Олег']
```

Схожим чином працює вставка елементів у список за допомогою методів `append()` та `insert()`, проте результат може бути дещо відмінним:

```
appendedList = []
appendedList.append(17)          # вставка елементу в кінець списку
appendedList.append(names)       # вставка списку в список
appendedList.insert(2, something[3]) # вставка елементу в задану позицію у списку
print(appendedList)
```

У консолі буде відображено:

```
[17, ['Олександр', 'Ірина', 'Сергій', 'Вікторія', 'Владислав',
      'Єлизавета', 'Олег'], 3]
```

Зверніть увагу, що в ході вставки списку в список відбувається не дописування окремих елементів, як у випадку з `extend()`, а вставка єдиного елементу-списку, на що вказують додаткові квадратні дужки навколо переліку імен.

При роботі з ітерованими об'єктами доступна можливість робити зрізи (slices), зокрема «вирізати» значення з одного списку в інший. Для цього у квадратних дужках через двокрапку записуються ліва межа, права межа та крок зрізу:

```
print(names[0:5:2])          # ['Олександр', 'Сергій', 'Владислав']
```

Популярними є подібні зрізи:

- `[:]`/`::` — «весь список»;
- `::2` — «непарні по порядку елементи»;
- `[1::2]` — «парні по порядку елементи»;
- `::-1` — «всі елементи в зворотному порядку»;
- `[2:]` — «всі елементи, починаючи з третього»;
- `[3]` — «всі елементи, не включаючи ті, що після другого».

Поширеними при роботі зі списками є агрегатні функції: `sum()`, `count()`, `min()`, `max()`, а також функція `index()` для знаходження позиції першої зустрічі заданого значення в списку. Застосуємо функцію `sum()` до списку з числами:

```
numbers = [78.45, 152, 21.654, -86.4, 9999]
print(sum(numbers))          # 10164.704
```

Очевидно, що для списків, у яких є нечислові значення, функція переважно викидатиме помилки, оскільки не знатиме, як додавати такі об'єкти.

Аналогічно підрахуємо, скільки разів зустрічається імена «Олег» та «Ольга» в списку `names` за допомогою методу `count()`:

```
print(names.count('Олег'))    # 1
print(names.count('Ольга'))   # 0
```

Метод `index()` дозволить визначити позицію імені в списку, починаючи з 0:

```
print(names.index("Олег"))    # 6
print(names.index('Ольга'))   # 0
```

Оскільки імені Ольга в списку немає, отримаємо помилку при виконанні:
ValueError: 'Ольга' is not in list

Визначення мінімуму та максимуму тісно пов'язане з порівнянням та сортуванням елементів набору. При цьому слід звернути увагу, що при сортуванні слів у стандартному для Python кодуванні UTF-8 літери [українського алфавіту](#) не повністю впорядковані за своїми кодами. Зокрема, кирилиця в UTF-8 записується в діапазоні кодів 0410-0474, проте літера Є має шістнадцятковий код 0404, І – 0406, Ї – 0407, Г – 0490, г – 0491. Це можна перевірити за допомогою вбудованої функції ord(), яка повертає код переданого символу:

```
print(hex(ord('Є')))    #0x404
print(hex(ord('І')))    #0x406
print(hex(ord('А')))    #0x410
print(hex(ord('Б')))    #0x411
```

Тому сортування імен може мати дещо несподіваний ефект:

```
print(sorted(names))
# даватиме вивід
# ['Єлизавета', 'Ірина', 'Владислав', 'Вікторія', 'Олег',
# 'Олександр', 'Сергій']
```

У функції sorted є необов'язкові параметри key (функція, яка визначатиме критерій сортування) та reverse (порядок сортування від меншого до більшого – False, від більшого до меншого – True). Передача функції в якості параметру іншої функції є проявом можливостей функціонального програмування, мова про яке піде набагато пізніше. У якості частинного виправлення проблеми можна запропонувати свій власний критерій сортування. Для цього створимо окрему функцію, яка міститиме в собі український алфавіт та повертатиме індекс (номер в алфавіті) літери.

```
def ukrABC(ch):
    abc = 'абвгґдеежзиіїйклмнопрстуфхццщщя'
    return abc.index(ch)
```

Тепер можемо задати цю функцію як критерій сортування за допомогою параметру key:

```
print(sorted(names, key=lambda word: ukrABC(word[0].lower())))
```

Зверніть увагу, що в даному випадку сортування відбувається лише за першою літерою кожного з рядків. Також перші букви зводяться до маленьких літер, щоб відповідати записаному в функції ukrABC() алфавіті за допомогою методу lower(). Для передачі функції в якості параметру використовується лямбда-вираз, який вже стосується функціонального програмування та розглядатиметься пізніше. Результат роботи буде таким:

```
['Вікторія', 'Владислав', 'Єлизавета', 'Ірина', 'Олександр',
'Олег', 'Сергій']
```

З іншого боку, відсортувати імена можна, наприклад, за довжиною – функція len():

```
print(sorted(names, key=len)) #['Олег', 'Ірина', 'Сергій', 'Вікторія',
'Олександр', 'Владислав', 'Єлизавета']
```

Параметр `key` є необов'язковим параметром і для функцій `min()` і `max()`.

```
print(min(names, key=len))    # Олег
print(max(names, key=len))    # Олександр
```

Зверніть увагу, що максимальний елемент не є останнім у відсортованому за цим критерієм списку. Це перший елемент з максимальною довжиною. Тут маємо три дев'ятибуквенних імені, тому автоматично вони стають в порядку їх появи в первинному списку.

Для списків характерний метод `sort()`, який працює за аналогією до функції `sorted()`, проте має дещо іншу сигнатуру:

```
print("Початковий список: \t\t" + str(names))
print(sorted(names, key=len))
print("Список після sorted(): \t" + str(names))
print(names.sort(key=len))
print("Список після sort(): \t" + str(names))
```

Вивід такий:

```
Початковий список:      ['Олександр',      'Ірина',      'Сергій',
                          'Вікторія', 'Владислав', 'Єлизавета', 'Олег']
                          ['Олег',      'Ірина',      'Сергій',      'Вікторія',      'Олександр',
                          'Владислав', 'Єлизавета']
Список після sorted():  ['Олександр',      'Ірина',      'Сергій',
                          'Вікторія', 'Владислав', 'Єлизавета', 'Олег']
None
Список після sort():    ['Олег',      'Ірина',      'Сергій',
                          'Вікторія', 'Олександр', 'Владислав', 'Єлизавета']
```

Як бачимо, початковий список після застосування `sorted()` насправді не змінюється. На бажання можна зберегти відсортовану послідовність в іншу змінну. Проте метод `sort()` застосовується напямую до початкового списку та нічого не повертає (вивело `None`). Звідси, маємо перетворення первинного варіанту списку після сортування. Також зверніть увагу, що `sort()` визначений лише для списків, тоді як функція `sorted()` працює і з іншими колекціями.

За аналогічною ідеєю працюють функція `reversed()` і метод `reverse()` для зміни порядку елементів списку від кінця до початку:

```
print(list(reversed(names)))
names.reverse()
print(names)
```

Функція `reversed()` повертає спеціальний об'єкт-ітератор, тому результат потрібно явно звести до типу `list()`.

Мова Python також пропонує методи `pop()` або `remove()` та ключове слово `del` для видалення елементів списку. Приклад роботи з методом `pop()`:

```
print(names)    # ['Олександр', 'Ірина', 'Сергій', 'Вікторія', 'Владислав',
                  'Єлизавета', 'Олег']
print(names.pop()) # Олег
print(names.pop()) # Єлизавета
print(names)    # ['Олександр', 'Ірина', 'Сергій', 'Вікторія', 'Владислав',
```

```
'Єлизавета']
print(names.pop(3)) # Вікторія
print(names)        # ['Олександр', 'Ірина', 'Сергій', 'Владислав', 'Єлизавета']
print(names.pop(2)) # Сергій
print(names)        # ['Олександр', 'Ірина', 'Владислав']
```

Назва методу повторює назву аналогічної операції для стеку, тому принцип «останній прийшов – перший пішов» працює й тут: елементи видаляються з кінця списку. Проте при передачі індексу в метод видалення відбувається відносно початку списку з індексом 0. Якщо індекс виходить за межі списку, буде викинута відповідна помилка: `IndexError: pop index out of range`

У результаті роботи методу повертається не змінений список, а лише видалений елемент. Водночас, метод `remove()` дозволяє видаляти елементи списку не за індексом, а за значенням:

```
print(names)        # ['Олександр', 'Ірина', 'Сергій', 'Вікторія', 'Владислав', 'Єлизавета', 'Олег']
print(names.remove("Ірина")) # None
print(names)        # ['Олександр', 'Сергій', 'Вікторія', 'Владислав', 'Єлизавета', 'Олег']
```

Ключове слово `del` працює на більш глобальному рівні. Даний оператор мови програмування (інструкція) дозволяє видаляти не просто окремі елементи колекцій, а й цілі об'єкти загалом:

```
print(names)        # ['Олександр', 'Ірина', 'Сергій', 'Вікторія', 'Владислав', 'Єлизавета', 'Олег']

del names[2]
print(names)        # ['Олександр', 'Ірина', 'Вікторія', 'Владислав', 'Єлизавета', 'Олег']

del names
print(names)        # NameError: name 'names' is not defined
```

(Кортежі) Кортежі (tuple, вимовляється як «тапл») Працюють у мові Python так же, як і списки, проте за своєю природою є незмінюваними (immutable). Унаслідок цього методи для вставки та видалення елементів недоступні для кортежів, проте можна конкатенувати (поєднувати) кортежі за допомогою оператора «+»:

```
emptyTuple = ()
someTuple = 78, 16
namesTuple = tuple(names)

print(hex(id(emptyTuple))) # 0x298dbb00040
print(hex(id(someTuple)))  # 0x298dc1cd300
print(hex(id(namesTuple))) # 0x298dc1c99a0

emptyTuple += namesTuple
someTuple += namesTuple

print(hex(id(emptyTuple))) # 0x298dc1c99a0
print(hex(id(someTuple)))  # 0x298dc196eb0
print(hex(id(namesTuple))) # 0x298dc1c99a0
```

Простежити за незмінюваністю кортежів можна за ідентифікаторами їх програмних об'єктів. Для демонстрації було створено три кортежі: порожній, з двома цілими значеннями (дужки при створенні не є обов'язковими) та перетворений зі списку з попереднього блоку. Всі вони мають різні ідентифікатори.

Тепер спробуємо конкатенувати останній кортеж з першими двома та переглянемо ідентифікатори сформованих об'єктів. З'єднання з порожнім кортежем призвело до того, що `emptyTuple` та `namesTuple` отримали однакові ідентифікатори. Це означає, що інтерпретатор оптимізував розміщення кортежів у пам'яті, оскільки зараз вони однакові. Проте за результатами конкатенації непорожніх кортежів утворився новий об'єкт `someTuple`, оскільки його ідентифікатор відрізняється від ідентифікаторів старих кортежів. Це означає, що незмінюваність формує наступну політику: при внесенні змін у початковий кортеж відбувається формування нового зміненого кортежу замість коригування його початкової версії.

Водночас, багато можливостей списків застосовні до кортежів. Зокрема, зрізи, функція `len()`, за умови сумісності типів елементів кортежу функції `sorted()`, `min()`, `max()`, а також інструкція `del` щодо всього об'єкта-кортежа.

(Множини) Програмні об'єкти, подібні до математичних множин. Множини (`sets`) тримають в собі лише унікальні елементи, причому неупорядковано, на відміну від списків та кортежів. Подібно до співвідношення між списками й кортежами в контексті незмінюваності, існують звичайні (змінювані) та заморожені (незмінювані) множини.

Створити порожню множину можна тільки за допомогою конструктора `set()`, оскільки літерал «`{ }`» використовується не лише множинами, а й словниками.

Операції з множинами в Python побудовані на принципах теорії множин:

- `len(s)` – розмір (кількість елементів) множини;
- `in` – оператор приналежності елемента множині;
- `set1.isdisjoint(set2)` – повертає `True`, якщо `set1` та `set2` не мають спільних елементів;
- `set1 == set2` – рівність множин: всі елементи `set1` належать `set2`, і навпаки;
- `set1.issubset(set2)` або `set1 <= set2` – всі елементи `set1` належать `set2`;
- `set1.issuperset(set2)` або `set1 >= set2` – всі елементи `set2` належать `set1`;
- `set1.union(set2, ...)` або `set1 | set2 | ...` – об'єднання кількох множин;
- `set1.intersection(set2, ...)` або `set1 & set2 & ...` – переріз кількох множин;
- `set1.difference(set2, ...)` або `set1-set2-...` – різниця множин, множина всіх елементів `set1`, які не належать жодним іншим `set`;
- `set1.symmetric_difference(set2)`; `set1 ^ set2` – симетрична різниця множин, множина з елементів, які зустрічаються лише в одній з множин `set`;
- `set.copy()` – копіює множину.

```
emptySet = set()
namesSet = set(names)
wordSet = set('hello')
someSet1 = {6, 12, -5, 18, 7}
someSet2 = {5, 7, 23, 56, 12, 11, 10}

print(type({}))      # <class 'dict'>
```

```

print(names)          # ['Олександр', 'Ірина', 'Сергій', 'Вікторія', 'Владислав',
                        'Єлизавета', 'Олег']
print(namesSet)       # {'Олександр', 'Ірина', 'Єлизавета', 'Олег', 'Владислав',
                        'Вікторія', 'Сергій'}
print(wordSet)        # {'h', 'l', 'o', 'e'}

print("Олег" in namesSet)      # True
print(someSet1.union(someSet2)) # {5, 6, 7, 10, 11, 12, 18, 23, 56, -5}
print(someSet1.intersection(someSet2)) # {12, 7}
print(someSet1.difference(someSet2))   # {18, -5, 6}
print(someSet1.symmetric_difference(someSet2)) # {5, 6, 10, 11, 18, 23, 56, -5}

frozenSet = frozenset(someSet1)
print(someSet1 == frozenSet)      # True
print(type(frozenSet))            # <class 'frozenset'>
print(type(frozenSet - someSet2)) # <class 'frozenset'>
print(type(someSet2 - frozenSet)) # <class 'set'>

```

Також для звичайних множин доступний набір операцій, які можуть безпосередньо змінювати ці множини.

- `set.update(other, ...)`; `set |= other | ...` – об'єднання;
- `set.intersection_update(other, ...)`; `set &= other & ...` – переріз;
- `set.difference_update(other, ...)`; `set -= other | ...` – різниця;
- `set.symmetric_difference_update(other)`; `set ^= other` – симетрична різниця;
- `set.add(elem)` – вставка елемента в множину;
- `set.remove(elem)` – видалення елемента з множини. `KeyError`, якщо такого елемента немає.
- `set.discard(elem)` – видаляє елемент, якщо він присутній у множині;
- `set.pop()` – видаляє перший елемент з множини. Через неупорядкованість множин спрогнозувати, яким саме буде цей елемент, важко;
- `set.clear()` – очистка множини.

(Словники) Зберігають дані у вигляді пар «ключ-значення» (entry, item), де ключі (keys) виступають в якості унікальних ідентифікаторів значень (values) та спрощують їх пошук у великих наборах даних. За своєю суттю, словник без значень – це незмінювана множина, тому вони обидва використовують фігурні дужки для формування та представлення свого наповнення. Загальний синтаксис:

```
dictionary = {ключ : значення }
```

```
numbers = {'one':1, 'two':2, 'three':3}
```

Доступ до значення відбувається за його ключем. Водночас такий же синтаксис дозволяє здійснювати вставку нових пар за ключем

```

print(numbers['two'])    #2
numbers['ninety'] = 90

```

Видалення елементів покладається на методи `pop` (повертає видалене значення), `popitem` (повертає кортеж «ключ, значення» видаленого елементу) та `clear` (очищає словник повністю):

```

my_dict = {'First': 'Python', 'Second': 'Java', 'Third': 'Ruby'}
a = my_dict.pop('Third')

```



```

print('Value:', a)           # Value: Ruby
print('Dictionary:', my_dict) # Dictionary: {'First': 'Python', 'Second':
                              'Java'}
b = my_dict.popitem()
print('Key, value pair:', b) # Key, value pair: ('Second', 'Java')
print('Dictionary', my_dict) # Dictionary {'First': 'Python'}
my_dict.clear()
print('n', my_dict)         # {}

```

Поширені методи для отримання окремо наборів ключів, значень чи пар «ключ-значення»

```

print(my_dict.keys())        # dict_keys(['First', 'Second', 'Third'])
print(my_dict.values())      # dict_values(['Python', 'Java', 'Ruby'])
print(my_dict.items())       # dict_items([('First', 'Python'), ('Second',
'Java'), ('Third', 'Ruby')])

```

Практичні завдання

Оберіть задачі для виконання в сумі до 7 балів. **Після здачі задачі вісьмома особами з підгрупи подальша її подача іншими учасниками підгрупи не приймається!**

- 0,3 бала При аналізі даних, зібраних у результаті експерименту, часто може виникати потреба у видаленні найбільш екстремальних (занадто великих чи малих) значень до виконання обчислень. Ваша програма має видалити зі списку на n (ціле додатне число, введене користувачем) елементів по трое найбільших та найменших значень і вивести решту значень (порядок не важливий). Також забезпечте коректну обробку ситуації, коли у списку не вистачатиме елементів для видалення.
- 0,4 бала Напишіть програму, що буде обчислювати ітогові дані по роках, щорічні та щомісячні середні значення опадів за період у декілька років
{4.3, 4.3, 4.3, 3.0, 2.0, 1.2, 0.2, 0.2, 0.4, 2.4, 3.5, 6.6},
{8.5, 8.2, 1.2, 1.6, 2.4, 0.0, 5.2, 0.9, 0.3, 0.9, 1.4, 7.3},
{9.1, 8.5, 6.7, 4.3, 2.1, 0.8, 0.2, 0.2, 1.1, 2.3, 6.1, 8.4},
{7.2, 9.9, 8.4, 3.3, 1.2, 0.8, 0.4, 0.0, 0.6, 1.7, 4.3, 6.2},
{7.6, 5.6, 3.8, 2.8, 3.8, 0.2, 0.0, 0.0, 0.0, 1.3, 2.6, 5.2}
- 0,4 бала Напишіть програму, яка зчитує від користувача слова, кожне з нового рядка. Ввід закінчується порожнім рядком, після чого виводиться перелік всіх унікальних слів, які були введені. Наприклад, для вводу
first
second
first
third
second
програма може вивести
first
second
third
- 0,4 бала Код Морзе – схема кодування, яка використовує точки та тире для представлення цифр і букв. Напишіть програму, яка буде використовувати словник для зберігання відображення букв і цифр на код Морзе (таблиця).

Letter	Code	Letter	Code	Letter	Code	Number	Code
A	. -	J	. - - -	S	. . .	1	. - - - -
B	- . . .	K	- . -	T	-	2	. . - - -
C	- . - .	L	. - . .	U	. . -	3	. . . - -
D	- . .	M	- -	V	. . . -	4 -
E	.	N	- .	W	. - -	5
F	. . - .	O	- - -	X	- . . -	6	-
G	- - .	P	. - - .	Y	- . - -	7	- - . . .
H	Q	- - . -	Z	- - . .	8	- - - . .
I	. .	R	. - .	0	- - - - -	9	- - - - .

Ваша програма має читати повідомлення користувача та перетворювати його в код Морзе з наступним виводом закодованого тексту. Пробіли між словами слід залишати, а решту символів, що не є цифрою або буквою – ігнорувати. Наприклад, Hello, World! буде мати код

.... . -.-. -.-. --- .-- --- -. .-.- -.

5. *0,4 бала* Напишіть додаток, який використовує генерування випадкових чисел для створення речень. Використовуйте 4 списки рядків: article, noun, verb та preposition. Створіть речення, обравши з кожного масиву слова в такому порядку: артикль, іменник, дієслово, прийменник, артикль та іменник. Після вибору кожного слова виконайте його конкатенацію з реченням. Слова мають відокремлюватись пробілами. Загальне речення має починатись з великої літери та закінчуватись крапкою. Додаток має згенерувати та відобразити 10 речень.

Артикли: "the", "a", "one", "some" та "any";

Іменники: "boy", "girl", "dog", "town" та "car";

Дієслова: "drove", "jumped", "ran", "walked" та "skipped";

Прийменники: "to", "from", "over", "under" та "on".

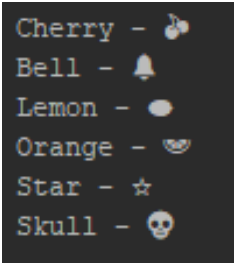
6. *0,4 бала* На більшості кнопчних телефонів для набору текстового повідомлення використовують цифрову клавіатуру. Оскільки кожна кнопка прив'язана до кількох букв, її потрібно натискати декілька разів для вибору потрібної відповідно до таблиці. Натиснення цифри 2, 3, 4 або 5 разів генерує другу, третю, четверту або п'яту букву для цієї кнопки.

Кнопка	Символи
1	.,?!:
2	A B C
3	D E F
4	G H I
5	J K L
6	M N O
7	P Q R S
8	T U V
9	W X Y Z
0	пробіл

Напишіть програму, яка показує, скільки та яких натиснень кнопок потрібно, щоб набрати текст користувача. Сконструйте словник, який відображає кожен символ на кнопку, яку потрібно натиснути. Наприклад, для тексту Hello, World! програма має вивести 4433555555666110966677755531111. Забезпечте коректну роботу як з малими, так і великими літерами. Ігноруйте символи, не представлені в таблиці.

7. *0,4 бала* Багато фінансових документів передбачають запис суми грошей як у числовій, так і в текстовій формі. Напишіть програму, яка буде конвертувати суми від 0 до 10 000 грн. у текстове представлення. Наприклад, 3726 грн – три тисячі сімсот двадцять шість

гривень. Використовуйте списки для переліку можливих текстових представлень для одиниць, десятків, сотень та тисяч гривень.

8. *0,5 бала* Створіть плейліст, кожна пісня якого має назву, виконавця, альбом та тривалість. Плейліст дозволяє додавати та видаляти пісню, отримувати розмір та очищати його, а також виконувати форматований вивід композицій і повну тривалість його відтворення.
9. *0,4 бала* Невелика авіакомпанія потребує систему автоматизації резервування місць у літаку на 12 місць. Ваша програма має пропонувати наступні альтернативи:
Натисніть 1 для першого класу (місця 1-6)
Натисніть 2 для економного класу (місця 7-12).
Потім Ваш застосунок повинен відображати посадкові місця, вказуючи на замовлене місце та його клас.
Використовуйте список булевих значень для представлення посадкових місць у літаку. Ваш застосунок має ніколи не давати резервувати вже зарезервовані місця, а повинен видавати перше вільне місце в класі. Коли місця з економного класу закінчились, програма повинна запропонувати квиток першого класу. Якщо таке не підходить, програма виведе повідомлення «Наступний виліт через 3 години».
10. *0,5 бала* Напишіть гру «Однорукий бандит» для консолі. У процесі будуть відображатись три випадкових символи з переліку:
Cherry - u"\U0001F352",
Bell - u"\U0001F514",
Lemon - u"\U0001F34B",
Orange - u"\U0001F34A",
Star - u"\u2606",
Skull - u"\U0001F480"
- 
- Гравцю видається кредит у розмірі 100 грн., а кожний запуск коштує 5 грн. Якщо «однорукий бандит» показує два однакових символи, користувач виграє 10 грн, а якщо три – 25 грн. У випадку випадіння трьох дзвіночків (bells) сума виграшу складає 100 грн. Якщо випадає два черепа – гравець додатково втрачає 5 грн, а при отриманні трьох черепів вся сума згорає. Гравець може зняти вигране до початку нової спроби або продовжувати грати, поки на рахунку будуть гроші.
11. *0,5 бала* Напишіть програму, яка зчитує дату від користувача та обчислює дату наступного дня. Дата буде вводиться у числовій формі трьома окремими виразами (рік, місяць, день). Наприклад,
1) якщо користувач вводить значення, яке представляє день 2017-05-10, Ваша програма має показати повідомлення, яке говоритиме, що наступним днем буде 2017-05-11.
2) Якщо користувач вводить значення на зразок 2017-04-30, програма має вказати, що наступним днем буде 2017-05-01.
3) Якщо користувач вводить значення, що представляє дату 2016-12-31, програма має вивести наступний день - 2017-01-01.
4) Забезпечте коректну роботу для високосних років.

Не використовуйте вбудованих методів для обчислення наступного дня!

12. *0,3 бала* Число Капрекара – це невід’ємне ціле число, квадрат якого в цій системі можна розбити на дві частини, сума яких дає початкове число. Наприклад, 45 є числом Капрекара, оскільки $45^2 = 2025$, причому $20 + 25 = 45$. Напишіть функцію, яка отримуватиме число та повертатиме дві частини, якщо це число Капрекара. Інакше функція поверне -1. Протестуйте роботу функції на різних числах.
13. *0,3 бала* Генерування пароллю шляхом вибору випадкових символів дає відносно безпечний, проте загалом складний для запам’ятовування пароль. Альтернативним способом є конструювання пароллю з двох англійських слів з подальшою конкатенацією. Такий пароль простіше запам’ятати, хоча і зламати також. Напишіть програму, яка містить список слів, обирає два випадковим чином та з’єднує їх в одне. При створенні пароллю переконайтесь, що загальна його довжина знаходиться в діапазоні від 8 до 10 символів, а кожне використане слово складається принаймні з трьох букв. Зробіть так, щоб у паролі кожне слово починалось з великої літери та виведіть пароль для користувача.
14. *0,4 бала* Ціле число n називають досконалим, якщо сума всіх його дільників дорівнює цьому числу. Наприклад, 28 є досконалим, оскільки його дільники 1, 2, 4, 7 та 14 у сумі дають 28. Створіть функцію, яка перевірятиме, чи є число досконалим. Напишіть програму, яка визначатиме досконалі числа з проміжку від 1 до 10 000 і записуватиме їх у список. Після цього список потрібно вивести на екран.
15. *0,6 бала* Редакційна відстань (edit distance) між двома рядками є мірою їх схожості. Чим менша редакційна відстань, тим більше схожі між собою рядки відносно мінімальної кількості вставок, видалень та заміन при перетворенні одного рядка в інший. Наприклад, рядки kitten та sitting. Перший рядок можна перетворити в другий, якщо: замінити k на s, замінити e на i, вставити g в кінець рядка. Три – це найменша кількість операцій, що можна виконати для перетворення kitten у sitting. Напишіть рекурсивну функцію, яка обчислює редакційну відстань між 2 рядками. Використовуйте наступний алгоритм:

```
Візьмемо рядки s і t
If довжина s = 0
    Повернути довжину t
Else if довжина t = 0
    Повернути довжину s
Else
    cost = 0
    If останній символ в s не дорівнює останньому символу в t
        cost = 1
    d1 = (редакційна відстань між всіма символами, крім
        останнього в s та всіма символами в t) + 1
    d2 = (редакційна відстань між всіма символами в s та всіма
```

```

        символами в t, крім останнього) + 1
d3 = (редакційна відстань між всіма символами, крім
      останнього, в s та всіма
      символами в t, крім останнього) + 1
Return min(d1, d2, d3)

```

Використайте дану рекурсивну функцію, щоб написати програму, яка зчитує 2 рядки від користувача і виводить редакційну відстань між ними.

16. *0,3 бала* (**Спискове включення**) Функція foo(C) потрібна, щоб повернути «зворотну» версію символьного рядка:

```

>>> foo("hawk")
'kwah'
>>> foo("amazing")
'gnizama'

```

Напишіть таку функцію відповідно до наступної стратегії. Спочатку перевірте, чи можете написати range-вираз, який проходить від індексу останньої букви до першого індексу. Наприклад, для рядка "thousand" range-вираз має повернути [7,6,5,4,3,2,1,0]. Використайте цей вираз, щоб зробити власний список, який виконуватиме схожий код: [C[i] for i in range(. . .)]

Це допоможе спростити foo(C), яка повертатиме цей кастомний список. Не використовуйте reverse(). Залишіться перетворити список символів у рядок. Використовуйте для цього метод join().

17. *0,3 бала* Опишіть рекурсивну функцію GCD, яка повертає найбільший спільний дільник чисел x та y. Обчислення НСД відбувається рекурсивним чином відповідно до формули

$$GCD(x, y) = \begin{cases} x, & \text{якщо } x = y \\ GCD(x - y, y), & \text{якщо } x > y \\ GCD(x, y - x), & \text{якщо } x < y \end{cases}$$

18. *0,4 бала* Напишіть функцію, яка перевірятиме номер кредитної картки на валідність. Наприклад, для номеру картки 4578 4230 1376 9219 маємо окремі цифри:

4-5-7-8-4-2-3-0-1-3-7-6-9-2-1

Починаючи з першої цифри (нумерація з 1) і до передостанньої, домножаємо на 2 кожен цифру, що стоїть на місці з непарним номером:

8-5-14-8-8-2-6-0-2-3-14-6-18-2-2

Якщо отримується двоцифрове значення, додайте його цифри та замініть на їх суму: 8-5-5-8-8-2-6-0-2-3-5-6-9-2-2

Нарешті, додайте всі отримані цифри:

$$8 + 5 + 5 + 8 + 8 + 2 + 6 + 0 + 2 + 3 + 5 + 6 + 9 + 2 + 2 = 71$$

Сума отриманого числа з останньою цифрою картки повинна давати число, кратне 10: 71 + 9 = 80

Інакше номер картки некоректний. Введіть номер картки як рядок з клавіатури та виведіть результат роботи функції: Correct або Incorrect.

19. *0,4 бала* Щоб зашифрувати повідомлення за допомогою шифру частоколу, переписуємо його у відповідному вигляді. Наприклад, для слова «криптографія» з висотою частоколу 2 отримаємо таку схему: к^Ри^Пт^Ог^Ра^Фі^Я. Далі зчитуємо текст рядками, почавши з верхнього. В результаті отримуємо криптотекст «рпорфякитгаі». "Висоту" частоколу (секретний ключ шифру) можна вибирати, що й буде запропоновано користувачу. Напишіть функції, що будуть зашифровувати та розшифровувати введений користувачем текст за допомогою введеного ключа – висоти частоколу.

20. *0,4 бала* Трикутник Паскаля – це трикутник чисел, який містить біноміальні коефіцієнти, що знаходяться за формулою

$$a_{nr} = \frac{n!}{r!(n-r)!}$$

Напишіть функцію, яка будуватиме трикутник Паскаля заданої висоти (аргумент функції). Наприклад,

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

Зверніть увагу на форматування виводу трикутника: врахуйте відповідні відступи.

21. *0,4 бала* Секретна інформація часто редагується чи видаляється з документів перед їх публікуванням. Після публікації документів відредагований текст часто замінюється чорними прямокутниками.

Напишіть функцію, яка редагує всі секретні слова, замінюючи їх зірочками, навіть якщо слово з'являється всередині іншого слова. Передбачено існування списку секретних слів. Виведіть відредаговану версію початкового тексту на екран.

Корисним може бути метод `replace()` для роботи з рядками. Доповніть програму незалежністю від регістру літер у тексті, тобто слова `exam`, `Exam`, `ExaM` та `EXAM` будуть вважатись однаковими.

22. *0,3 бала* Напишіть програму, яка зчитує числа від користувача, поки не буде введено порожній рядок. Далі вона повинна знайти середнє значення із введених чисел та вивести 2 списки: введених чисел, що не перевищують середнє значення та, відповідно, більше або рівні за нього.

23. *0,3 бала* Нехай грає музика! Запрограмуйте комп'ютер на відтворення мелодії, заданої окремими списком частот та списком тривалостей відтворення відповідної частоти. Відповідні пари значень наведені нижче:

(659,250), (659,250), (659,300), (523,250), (659,250), (784,300), (392,300), (523,275), (392,275), (330,275), (440,250), (494,250), (466,275), (440,275), (392,275), (659,250), (784,250), (880,275), (698,275), (784,225), (659,250), (523,250), (587,225), (494,225).

Для підключення біперу потрібно імпортувати модуль `winsound` і викликати функцію `winsound.Beep(частота у Гц, тривалість у мс)`

24. *0,2 бала* Якщо Вам сподобалась попередня задача, спробуйте тепер іншу мелодію, збережену в *двовимірному* масиві (списку списків):
 (440,500), (440,500), (440,500), (349,350), (523,150), (440,500), (349,350), (523,150), (440,1000), (659,500), (659,500), (659,500), (698,350), (523,150), (415,500), (349,350), (523,150), (440,1000).

25. *0,4 бала* Звичайна колода карт має 52 карти, кожна з яких описується мастю і значенням. Мастями можуть бути піки (spades), черви (hearts), трефи (diamonds) та бубни (clubs), а значеннями – числа від 2 до 10, валет (Jack), дама (Queen), король (King) і туз (Ace). Приклади позначень карт описані в таблиці.

Карта	Абревіатура
Піковий валет (Jack of spades)	Js
Трефова четвірка (Four of clubs)	4c
Бубнова дев'ятка (Nine of diamonds)	9d
Чирвова десятка (Ten of hearts)	Th

Напишіть програму, яка буде створювати колоду карт у вигляді списку. Перетасуйте колоду відповідно до алгоритму Сатолю, представленого [тут](#). Виведіть початкову колоду та перетасовану на екран.

26. *0,3 бала* Створивши та перетасувавши колоду в попередніх задачах, прийшов час здати їх гравцям. Кількість гравців та карт для кожного з них залежить від гри, тому користувач повинен після перетасовки ввести кількість карт на одного гравця та кількість гравців. За цими даними потрібно вивести інформацію зі списку про гравців та карти, що їм були здані. Забезпечте коректну роботу програми, коли карт для гравців потрібно більше, ніж є в колоді.

27. *0,3 бала* Напишіть програму, яка симулюватиме 1000 кидків пари костей та виводитиме статистику сум, які випадають у відсотках. Оформити результат можна так:

Total	Simulated Percent	Expected Percent
2	2.90	2.78
3	6.90	5.56
4	9.40	8.33
5	11.90	11.11
6	14.20	13.89
7	14.20	16.67
8	15.00	13.89
9	10.50	11.11
10	7.90	8.33
11	4.50	5.56
12	2.60	2.78

Очікуваний відсоток – стандартне значення, отримане з розподілу Гауса.

28. *0,4 бала* Решето Ератосфена – алгоритм, який дозволяє знайти прості числа, що не перевищують деякого n . Він був придуманий понад 2000 років тому і описується так:
 Запишіть всі числа від 0 до n
 Викресліть 0 і 1, тому що вони не є простими

Задайте $p = 2$

While $p < n$ do

Викреслити всі числа, кратні p (проте не сам p)

Перейти (змінна p) до наступного числа в списку, яке не буде закресленим

Вивести всі незакреслені числа.

Сформуйте список усіх чисел від 2 до введеного значення n , а потім видаляйте з нього всі числа, які викреслює алгоритм. У результаті повинен отриматись список із простих чисел для введеного діапазону.

Підказка. Пришвидшити роботу початкового алгоритму можна за рахунок неповного перебору чисел. Насправді, достатньо перевірити числа не до n , а лише до \sqrt{n} .

29. *0,4 бала* У грі Scrabble™ кожна буква оцінюється балами, як показано в таблиці. Загальна оцінка слова – це сумарна кількість очків за кожну букву:

One point	A, E, I, L, N, O, R, S, T and U
Two points	D and G
Three points	B, C, M and P
Four points	F, H, V, W and Y
Five points	K
Eight points	J and X
Ten points	Q and Z

Напишіть програму, яка обчислює та виводить Scrabble™-оцінку для слова. Створіть словник, який відображає букви на бальні значення, а потім з його допомогою аналізуйте введені слова, виводячи їх оцінку в балах.

У грі Scrabble™ включено деякі squares, які домножають значення букви чи всього слова на певні значення. Це правило в задачі ігнорується.

30. *0,5 бала* Припустимо, на залізничному вокзалі присутня автоматизована система з управління камер зберігання для багажу пасажирів. Інформація про камери зберігається у вигляді переліку «Прізвище пасажир-час здачі багажу-час видачі багажу». На вокзалі присутні 12 комірок, пасажирів відкривається вільна комірка з мінімальним номером. Вважаємо, що робота програми обмежується одним днем, а час здачі та видачі багажу вводить сам пасажир. Для роботи з програмою спочатку треба ввести кількість пасажирів, а потім їх дані, наприклад, так:

3

Іваненко 09:45 12:00

Петренко 10:00 11:00

Сидоренко 12:00 13:12

На екран має бути виведено, яку комірку відкривати кожному з пасажирів.

Іваненко 1

Петренко 2

Сидоренко 1

31. *0,4 бала* Введіть з клавіатури деякий текст та підрахуйте в ньому кількість слів, букв, цифр, малих літер, великих літер, пробілів.

32. *0,4 бала* Лінійна регресія дозволяє побудувати пряму, яка найкраще наближає набір з n точок даних (data points). Це дає можливість, наприклад, встановити тренд цін на біржі в деякому проміжку часу. Кожна точка з набору має координати $(x; y)$ – хвилину з початку роботи біржі та ціну за акцію. Символами \bar{x} та \bar{y} позначаємо середнє значення по окремим координатам. Лінія тренду представляється рівнянням $y = ax + b$, де a і b обчислюються за формулами

$$a = \frac{\sum xy - \frac{(\sum x)(\sum y)}{n}}{\sum x^2 - \frac{(\sum x)^2}{n}}$$

$$b = \bar{y} - m\bar{x}$$

Напишіть програму, яка зчитує колекцію точок від користувача по координатно, поки не буде введено порожній рядок. У результаті потрібно вивести формулу для прямої, наприклад, $y = 0.95x + 0.1$ для координат $(1, 1)$, $(2, 2.1)$ та $(3, 2.9)$.

33. *0,6 бала* Виграшна картка лото містить лінію з 5 чисел, які до цього були витягнуті з мішка. Гравці закривають витягнуті числа фішками. У нашому додатку будемо замінити число на 0 у лото-словнику, якщо воно було оголошено.

Напишіть програму, яка працює зі словниками, що представляють картки лото. Якщо картка містить рядок з 5 нулів, вона вважається виграшною. Програма має формувати 2 картки лото (9 стовпців чисел, кожен з яких відповідає десятку числа, 3 рядка, по 5 чисел у кожному), генерувати випадкове число та заново виводити картку після цього. Як тільки якась із карток виграє, програма завершує роботу з повідомленням про виграш відповідного гравця.

6		28		47		63		89
		20	34		53		70	86
	11		32	45	51		77	

Підказка. Оскільки всі числа додатні, знаходження 5 нулів на картці відповідає задачі рівності суми 5 входжень нулю. Можливо, Вам так буде простіше моделювати гру.

34. *0,7 бала* Для гри в зонк потрібно 6 шестигранних кубиків, які будуть кидатись одночасно. Доступні наступні комбінації:

Категория	Название	Очки	Пример	Примечания
	Единицы	100 за каждую		Комбинация из трех единиц дает 1000 очков. Однако в различных вариациях правил четвертая единица дает как 100, так и 1000 очков.
	Пятерки	50 за каждую		Комбинация из трех пятерок дает 500 очков. Однако в различных вариациях правил четвертая пятерка дает как 50, так и 500 очков.
Три одинаковых				
	Единицы	1000		
	Двойки	200		
	Тройки	300		
	Четверки	400		
	Пятерки	500		
	Шестерки	600		
Специальные				
	Четвертая/пятая/шестая кость	Приносит столько же, сколько три предыдущих кости	 + = 300 + 300	Это касается и пятой, и шестой кости. В некоторых вариантах правил четвертая кость не обладает этим свойством.
	Три пары	750 и призовой бросок		Если выпали 4 кости одного достоинства и 2 другого (например,), эта комбинация 750 очков не принесет. Кроме того, комбинацию "три пары" можно получить только за один бросок.
	Шесть разных	1500 (1000) и призовой бросок		В других вариантах правил эта сумма может быть несгораемой или за эту комбинацию дается 1500 очков.

Напишіть підпрограму, яка аналізує кидок одного гравця. При цьому генерується 6 випадкових чисел (від 1 до 6). Виведіть на екран доступні комбінації за результатом кидка та очки, які за них набере користувач.

35. *0,4 бала* Нижче наведена таблиця описує октави музичних нот, починаючи з middle C (C4) разом з їх частотами.

Нота	Частота (Гц)
C4	261,63
D4	293,66
E4	329,63
F4	349,23
G4	392,00
A4	440,00
B4	493,88

Програма має коректно виводити частоти для нот всіх октав – від C0 до C8. Для того, щоб їх знайти, потрібно враховувати наступне правило: частота ноти в октаві n – половина частоти ноти в октаві n+1. Додаток має підтримувати 2 варіанти роботи:

1) На вхід програми має надходити нота відповідної октави, а на виході виводиться її частота.

2) За введеною частотою визначити ноту.

Підказка: для виконання завдання потрібно виділити окремо символи з двосимвольної назви ноти. Далі можна визначити частоту ноти в четвертій октаві, а потім поділити її на 2^{4-x} , де x – номер октави, введеної користувачем.

36. *0,5 бала* (Пил Кантора) Множину Кантора отримують шляхом поділу відрізка на 3 частини та видалення середньої з них. Напишіть рекурсивну функцію, що дозволить відобразити аналогічне зображення в консольному додатку. Для візуалізації n -того рівня множини Кантора використовують правило $n = 0$: на початку для $[0, 1]$; $n > 0$: застосувати правило n разів

37. *0,3 бала* Опишіть рекурсивну функцію для піднесення числа до цілої додатної степені

$$x^n = \begin{cases} 1, & \text{якщо } n = 0 \\ x \cdot x^{n-1}, & \text{якщо } n > 0 \end{cases}$$

38. *0,3 бала* Більш ефективне рекурсивне визначення x^n , що охоплює випадок $n < 0$:

$$x^n = \begin{cases} 1, & \text{якщо } n = 0 \\ \left(x^{\frac{n}{2}}\right)^2, & \text{якщо } n > 0, \quad n - \text{парне} \\ x \cdot x^{n-1}, & \text{якщо } n > 0, \quad n - \text{непарне} \\ \frac{1}{x^n}, & \text{якщо } n < 0 \end{cases}$$

39. *0,4 бала* **Градова послідовність (Hailstone Sequence)**. Градова послідовність (градові числа, послідовність Колатца) називається так тому, що її значення зазвичай зростають та зменшуються подібно до форми градових хмар.

Створіть програму, яка рекурсивно генеруватиме градову послідовність чисел, починаючи з деякого введенного цілого числа n . Правила формування послідовності:

Якщо $n = 1$, послідовність закінчується.

Якщо n – парне, наступне число послідовності буде $n/2$

Якщо n – непарне, наступне число послідовності $= (3 * n) + 1$

Використайте програму, щоб показати, що для числа 27 послідовність складатиметься з 112 елементів, починаючись з 27, 82, 41, 124 та закінчуючись числами 8, 4, 2, 1

40. *0,4 бала* Створіть програму, яка підраховує суму всіх введених користувачем чисел, пропускаючи некоректно введені (нечислові) дані. Програма має виводити поточну суму після кожного введенного числа. При вводі нечислових даних виводиться повідомлення про помилку, проте робота програми продовжується. Вихід з програми здійснюється після вводу порожнього рядка. Переконайтесь, що програма працює коректно як для цілих, так і дробових значень.

Підказка: вправа вимагає використання винятків.

41. *0,3 бала* Один з перших відомих прикладів шифрування використовував Юлій Цезар. Йому потрібно було надсилати письмові інструкції своїм генералам, проте він не хотів, щоб вороги дізнались про його плани, якщо повідомлення потрапить їм у руки. Тому Цезар запропонував просту ідею: кожна літера первинного повідомлення зсувалась на 3 позиції в алфавіті. Замість А записували D, замість В – Е, С ставало F, D – G і т. д. Останні 3 букви в алфавіті мали замітники з його початку ($X = A$, $Y = B$, $Z = C$). Небуквенні символи не зашифровувались.

Напишіть програму, яка реалізує шифр Цезаря. Дозвольте користувачеві ввести повідомлення та розмір зсуву, а потім виведіть «зсунуте» повідомлення. Переконайтесь, що Ваша програма кодує як великі, так і маленькі букви. Додаток також має підтримувати від'ємний зсув, щоб мати можливість як зашифрувати, так і розшифрувати повідомлення.

42. *0,6 бала* Реалізуйте гру «Хрестики-нолики». Крім головної функції, виділіть окремі підпрограми для

- ініціалізації матриці 3x3 «порожніми» символами,

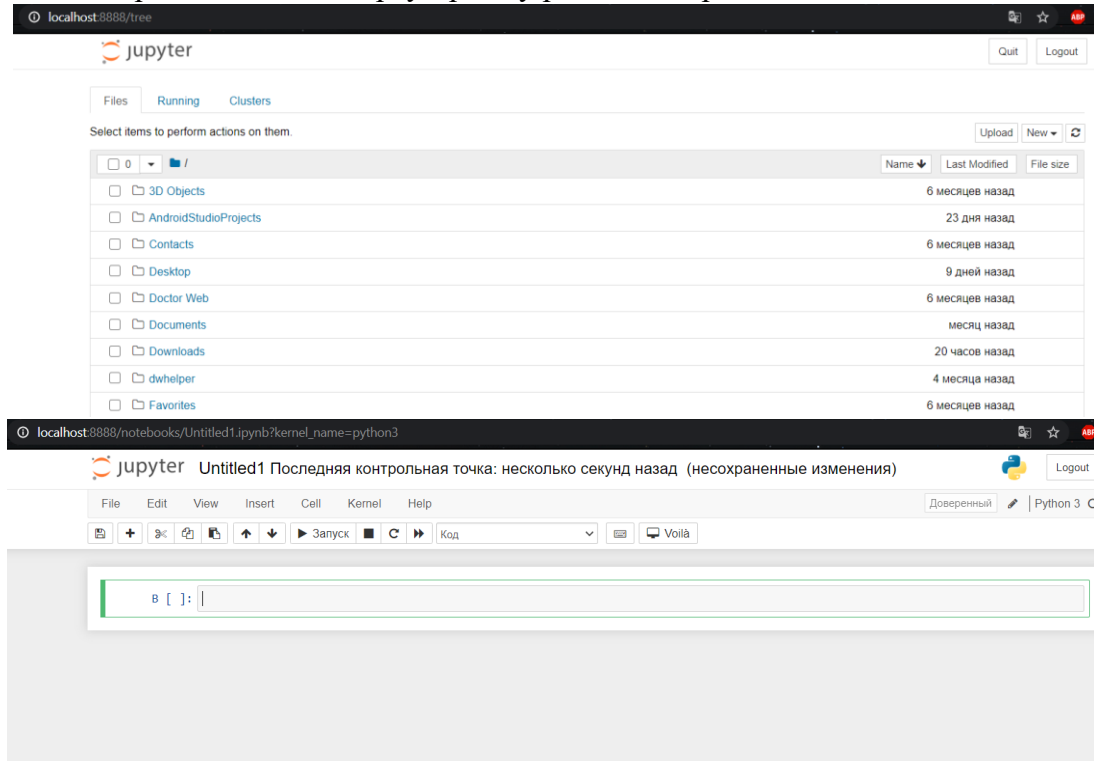
- обробки ходу гравця,
- обробки ходу комп'ютера,
- перевірки виграшу,
- виводу результатів.

Нехай комп'ютер під час свого ходу заповнює першу «порожню» клітинку, а гравець — задає координати (X,Y), які треба перевірити на валідність.

ІНДЗ

1. ^{2 бали} (*Jupyter Notebook*) Ознайомтесь з можливостями роботи з інструментом Jupyter Notebook. Для початку роботи Ви маєте кілька варіантів його інсталяції:

- Встановити [окремий інструмент JupyterLab](#). Відповідно до цього способу Ви працюватимете в браузерному режимі з окремими блокнотами:



- [налаштувати](#) підтримку JupyterLab у PyCharm Professional (безкоштовна версія Community Edition не підтримує роботу з блокнотами Jupyter, починаючи з 2019.x). Майте на увазі, що версія Professional працюватиме в trial-режимі протягом визначеного періоду. Або розгляньте можливість отримання [безкоштовної ліцензії](#) інструментів JetBrains для студентів, зокрема за програмою GitHub Student Developer Pack;
- [Налаштувати](#) підтримку Jupyter Notebook в Anaconda Navigator.

Дослідіть питання практичної потреби та використання даного інструменту, опишіть у звіті переваги від його використання, а також продемонструйте інсталяцію та інтерактивну роботу з блокнотом Jupyter (файл .ipynb) на власній машині. За основу можете взяти [туторіал](#).

2. ^{2 бали} (*Налагодження коду*) Існує набагато потужніший спосіб розібратись в тому, що робить програма, ніж вводити print-інструкції в код: використовувати відладник (debugger).

- 1) Розпочніть з простої програми, код якої записано нижче.

```
name = 'Ваше ім'я та прізвище'  
greeting = 'Привіт, ' + name
```

```
print(greeting)
```

Збережіть код у файлі `pdb_exercise_1.py`.

Викличте скрипт з використанням відладника за допомогою командного рядка:

```
python -m pdb pdb_exercise_1.py
```

Відладник `pdb` (Python Debugger) покаже наступний рядок коду, який буде виконуватись (`-> name = 'Ваше ім'я та прізвище'`), а також запрошення до взаємодії з відладником (`Pdb`).

Введіть `n` (скорочено від `next`), щоб відобразити на екрані наступний рядок для виконання.

Тепер перевіримо значення змінної `name`. Введіть у командний рядок `p name` (`p` – скорочено від `print`). Повинно вивестись Ваше ім'я та прізвище. Знову введіть `n` для переходу до наступної команди. У змінній `greeting` має бути рядок `'Привіт, Ваше ім'я та прізвище'`.

При налагодженні коду дуже просто загубитись у тому, який рядок зараз налагоджується. Для того, щоб у цьому розібратись, введіть `l` (від `list`). На екрані з'явиться код поточного файлу та рядок, який налагоджується, вказаний стрілкою `->`. Знову введіть `n`, щоб виконати останню команду. Тепер можна ввести `q` та вийти з відладника. *Зробіть скриншоти Ваших дій та додайте їх у звіт.*

- Потренуємось налагоджувати роботу функцій. Створіть новий файл `pdb_exercise_2.py` та скопіюйте у нього відповідний код:

```
def greet(name):
```

```
    greeting = 'Привіт, ' + name
```

```
    return greeting
```

```
greeting = greet('Ваше ім'я та прізвище')
```

```
print(greeting)
```

Запустіть відладник, ввівши в командному рядку

```
python -m pdb pdb_exercise_2.py
```

Після запуску введіть `c` (від `continue execution`). Результат виводу наприкінці матиме вигляд

```
-> def greet(name):
```

Тричі введіть `n`, щоб було виведено всі три рядка тексту основної функції (не заходячи в `greet()`). Натисніть `c`, щоб перезапустити програму та один раз `n`, щоб отримати рядок, де викликається функція `greet()`. Цього разу введемо `s` (від `step`), щоб зайти у функцію `greet()` та ще 5 разів введемо `n`. Зверніть увагу на відмінності у виводі та відобразіть їх за допомогою скриншотів у звіті.

Наприкінці цієї вправи розглянемо команду `r` (від `return`). Вона схожа до команди `c`, проте замість налагодження до кінця програми відбувається запуск програми лише до кінця функції.

Перезапустимо програму, ввівши спочатку *c*, а потім *n* та *s*. Тепер Ви маєте знаходитись у функції `greet()`. Для перевірки правильності використайте *l*.

Тепер натисніть *r* ("return"). Зауважте, що відбулося негайне переміщення у кінець функції, де має відбутись повернення значення.

- 3) Працюючи з цим інструментом нечасто, дуже просто забути назви команд та їх суть. Проте можна використовувати команду `help`, яка відновить інформацію в пам'яті.

(Pdb) `help`

Викличте `help` для команд, які розглядались до цього:

(Pdb) `help n`

(Pdb) `help s`

(Pdb) `help c`

(Pdb) `help r`

(Pdb) `help l`

(Pdb) `help help`

- 4) До цього моменту не було помилок у скриптах, які довелося би виправляти. Скопіюйте представлений нижче код у файл з назвою `pdb_exercise_4.py`.

```
import sys

def magic(x, y):
    return x + y * 2

x = sys.argv[1]
y = sys.argv[1]

answer = magic(x, y)
print('The answer is: {}'.format(answer))
```

Нехай скрипт запускається з числами 1 та 50 (очікуємо результат 101).

`python pdb_exercise_4.py 1 50`

Чи вивело відповідь 101?

Тепер замість вставки `print`-інструкцій по всьому коду, перевіримо код у відладнику коду:

`python -m pdb pdb_exercise_4.py 1 50`

Доберемось до точки, де відбувається доступ до змінних *x* та *y*.

(Pdb) `n`

(Pdb) `n`

(Pdb) `n`

(Pdb) `n`

Спочатку переглянемо, які атрибути доступні в скоупі програми. Це можна зробити за допомогою *p* (відповідає `print`).

(Pdb) `p dir()`

Також є **pp** (pretty print).

```
(Pdb) pp dir()
```

Чому дорівнює *x*?

```
(Pdb) p x
```

Зауважте, що можна використовувати Python-код у відладнику. Визначимо тип *x*:

```
(Pdb) type(x)
```

Виконуючи Python-код, наприклад, можна динамічно змінити вхідні змінні:

```
(Pdb) x = int(x)
```

```
(Pdb) y = int(y)
```

Перевіримо тепер значення до запуску програми.

```
(Pdb) p x, y
```

Чому *y* дорівнює 1, а не 50? Переглядаючи код, було встановлено, що забули оновити індекс при копіюванні вхідного parsing-рядка.

Змінімо значення *y* на 50 у відладнику до перевірки на відповідність роботи коду очікуванням, дозволивши пропрацювати йому до кінця.

```
(Pdb) y = 50
```

```
(Pdb) c
```

- 5) До цього часу ми проходили по скриптах від початку до кінця. Проте працюючи у більших програмах це часто непрактично. Для моделювання такої ситуації скопіюйте нижче наведений код у файл з назвою `pdb_exercise_5.py`.

```
import time

def slow_subtractor(a, b):
    """Return a minus b."""
    time.sleep(5)
    return a - b

some = slow_subtractor(12, 8)
crazy = slow_subtractor(12, 78)
scientific = slow_subtractor(56, 31)
experiment = slow_subtractor(101, 64)

total = some + crazy + scientific + experiment

experimental_fraction = experiment / total
```

Після запуску коду отримаємо `ZeroDivisionError`. Прохід по коду у відладнику може бути annoying, оскільки потрібно натискати `n` кожного разу, коли викликається функція `slow_subtraction()`. Замість цього додамо точку розриву (breakpoint) перед рядком, який генеруватиме помилку. Це виконується за допомогою імпортування модуля `pdb` та використання функції `pdb.set_trace()`.

```
import time
```

```
def slow_subtractor(a, b):  
    """Return a minus b."""  
    time.sleep(5)  
    return a - b  
  
some = slow_subtractor(12, 8)  
crazy = slow_subtractor(12, 78)  
scientific = slow_subtractor(56, 31)  
experiment = slow_subtractor(101, 64)  
  
total = some + crazy + scientific + experiment  
  
import pdb; pdb.set_trace()  
experimental_fraction = experiment / total
```

Якщо запустимо код тепер, то перейдемо в режим налагодження до виконання проблемного рядку коду.

```
python pdb_exercise_5.py  
(Pdb) p total  
(Pdb) p some, crazy, scientific, experiment
```

Зі змінною `crazy` буде робитись щось незрозуміле. Можливо, вхідні аргументи (input arguments) було задано неправильно.