**Лабораторна робота № 18**. РЯДКИ в мові PYTHON

*Рядки* - це впорядковані послідовності символів, що використовують для зберігання і подання текстової інформації (символів і слів, наприклад, ваше ім'я, змісту текстових файлів, завантажених в пам'ять, адрес в Інтернеті, програми на Python тощо). Рядки володіють потужним набором засобів для їх обробки. У Python відсутній спеціальний тип для подання одного символу, тому в разі необхідності використовуються односимвольні рядки.

Рядки відносять до категорії незмінних послідовностей, в тому сенсі, що символи, які вони містять, мають певний порядок розміщення зліва направо і самі рядки неможливо змінити. Рядки – це представник великого класу об'єктів, які називають послідовностями. Зверніть увагу на операції над послідовностями, наведені тут, тому що вони схожим чином працюють і з іншими типами послідовностей, такими як списки і кортежі, які ми будемо розглядати пізніше. У табл. 1 наведені найбільш типові літерали рядків і операцій

Таблиця1

*Типові літерали рядків та операції над ними*



Порожній рядок має вигляд пари лапок (або апострофів), між якими нічого немає. Для роботи з рядками підтримуються операції над виразами, такі як конкатенація (об'єднання рядків), виділення підрядка, вибірка символів за індексами (за зсувом від початку рядка) тощо. Python пропонує ряд методів, які реалізують різні завдання роботи з рядками.

Приклад 1. Викориcтання рядків.

s1 = "Рядок 1"; s2 = 'Рядок 2'

print(s1, s2)

# формування рядка з іншого значення

s3 = str(8);

print(s3)

# рядки, які складаються з багатьох рядків

s4 = """ Lesson2. Variables and Data Types Some data types explained in this lesson:

- int, - bool, - float, - complex, - str """

print(s4)

# символ \ використовують, щоб продовжити рядок

# або будь-який вираз в Python з наступного рядка кода

s5 = "started\

continued"

print(s5)

## II. ЦИКЛ WHILE

**1. Інструкція while***.* Алгоритм, в якому передбачено неодноразове виконання певної послідовності дій, називається алгоритмом циклічної структури або циклом. Цикл дозволяє істотно скоротити розмір запису алгоритму, зобразити його компактно шляхом відповідної організації пропонуємих дій. Повторювати певні дії має сенс при різних значеннях параметрів, які змінюються. Такі параметри називаються *параметрами циклу*. Блок повторюваних операторів називають *тілом циклу* (це послідовність дій, які виконується багаторазово).

Циклічний процес називається ітераційним, якщо заздалегідь невідома кількість повторень циклу, а кінець обчислення визначається при досягненні деякою величиною заздалегідь заданої точності обчислення.

При програмуванні ітераційних процесів прийнято їх розділяти на цикли з «передумовою» і з «післяумовою». Їх відмінність полягає в тому, що перевірка досягнення деякою величиною заданої точності обчислення здійснюється або на початку циклу, або наприкінці циклу відповідно. Особливість циклу з «післяумовою» полягає в тому, що повторювана ділянка алгоритму виконається хоча б один раз, у той час як в циклі з «передумовою» ця ділянка може не виконатися жодного разу. Процес ініціалізації включає в себе визначення (введення) початкових значень змінних, які використовуються в тілі циклу.

Для запису ітераційних процесів в Python використовують лише один тип операторів циклу *while* - оператор з попередньою умовою (передумовою).

Для всіх операторів циклу характерні такі особливості:

1. Повторювані обчислення записуються лише один раз.
2. Вхід в цикл можливий тільки через його початок.
3. Змінні оператора циклу повинні бути визначені до входу в цикл.
4. Потрібно передбачити вихід з циклу. Якщо цього не зробити, то обчислення будуть тривати нескінченно довго.

Нескінченний цикл – це циклічна ділянка в програмі, в якій не передбачені засоби виходу з циклу при досягненні деякого умови.

Інструкція *while* організує цикл з передумовою (перевірка виконується перед початком чергової ітерації), складається з рядка заголовка з умовним виразом, тіла циклу, що містить одну або більше інструкцій з відступами, і необов'язкової частини *else*, яка виконується, коли управління передається за межі циклу без використання інструкції *break*. Інтерпретатор продовжує обчислювати умовний вираз в рядку заголовка і виконувати вкладені інструкції в тілі циклу, поки умовний вираз не поверне значення «хибність»:

while <test>: # Умовний вираз test

<statements1> # Тіло цикла else: # Необов‘язкова частина else

<statements2> # Виконується, якщо вихід із цикла

# виконується не інструкцією break

Оператор *while* дозволяє багаторазово виконувати певні дії в залежності від деякої <умови> (виразу логічного типу, який приймає тільки значення True або False). Цикл виконується поки <Умова>=«істина». Як тільки <Умова> порушується, виконання циклу завершується. Блок-схема циклу *while* наведена на рис. 5.1.

Інструкція *while* продовжує виконувати блок інструкцій (зазвичай з відступами), поки умовний вираз продовжує повертати значення «істина». Вона називається «циклом», тому що управління циклічно повертається до початку інструкції, поки умовний вираз не поверне значення «хибність». Як тільки в результаті перевірки буде отримано значення «хибність», управління буде передано першій інструкції, яка розташована відразу ж за вкладеним блоком тіла циклу *while*. Розглянемо приклади простих циклів *while.*

Перший приклад (який містить інструкцію *print*, вкладену в цикл *while*), нескінченно виводить повідомлення (True – це особлива версія цілого числа «1», вона позначає значення «істина», тому результатом цього умовного виразу завжди буде «істина», і інтерпретатор нескінченно буде виконувати тіло циклу, поки ви не скасуєте його виконання). Такі цикли зазвичай називають нескінченними:

>>> while True:

print(‗Type Ctrl-C to stop me!‘)

Наступний фрагмент продовжує видаляти з рядка перший символ, поки

він не стане порожнім, в результаті умова прийме значення «хибність». Перевірка об‘єктів на значення «істина» здійснюється безпосередньо замість використання еквіваленту (*while* x != ‘’:).

>>> x = ‘spam'

>>> while x: # Поки x – це не пустий рядок

... print(x, end=’ ‘)

... x = x[1:] # Видалити перший символ з x ...

spam pam am m

Аргумент *end*= ‘ ’ забезпечує виведення значень в один рядок через пробіл. Наступний фрагмент перебирає значення від *a* до *b*, не включаючи значення *b*:

>>> a=0; b=10

>>> while a < b:

# Один зi способів організації циклів перебору

... print(a, end=’ ‘)

... a += 1 # або a = a + 1 ...

0 1 2 3 4 5 6 7 8 9

В Python відсутній цикл «*do until*», однак його можна імітувати, додавши в кінець тіла циклу умовну інструкцію та інструкцію *break*:

while True:

... <тіло цикла>...

if exitTest(): break

**2. Вreak, continue, pass і else.** Розглянемо дві прості інструкції, які можна використати тільки усередині циклів – інструкції *break* і *continue*. У Python:

*Break* виконує перехід за межі циклу (всієї інструкції циклу).

*Continue* виконує перехід на початок циклу (в рядок заголовка).

*Pass* нічого не робить: це пуста інструкція.

Блок *else* виконується, тільки якщо цикл завершився звичайним чином (без використання інструкції *break*).

З урахуванням інструкцій *break* і *continue* цикл *while* має такий загальний вигляд:

while <test1>:

<statements1> # тіло циклу

if <test2>: break # Вийти з циклу, пропустивши частину else

if <test3>:continue #Перейти на початок циклу, до вираз.

test1

else:

<statements2> # Виконується, якщо не була використана

# інструкція ‗break‘

Інструкції *break* і *continue* можуть з'являтися в будь-якому місці всередині тіла циклу *while* (або for), але як правило, їх використовують в умовних інструкціях *if*, щоб виконати необхідну дію у відповідь на деяку умову.

**Pass.** Інструкція *pass* не виконує ніяких дій, її використовують у випадках, коли синтаксис мови вимагає наявності інструкції, але ніяких корисних дій в цій точці програми виконати не можна. Вона часто використовується в якості порожнього тіла складної інструкції. Наприклад, створити нескінченний цикл, який нічого не робить, можна таким чином:

**while 1: pass # Натисніть Ctrl-C, щоб припинити цикл!**

Цей приклад нескінченно робить «ніщо». Ця інструкція може використовуватися, наприклад, для того, щоб ігнорувати виключення в інструкції *try*. Іноді інструкцію *pass* використовують як заповнювач, замість того, «що буде написано пізніше», і в якості тимчасового фіктивного тіла функцій:

**def func1():**

**pass # Реалізація функції буде додана пізніше**

**def func2(): pass**

Порожнє тіло функції викличе синтаксичну помилку, тому в подібних ситуаціях можна використати інструкцію *pass*. У Python 3.0 замість будь-якого виразу допускається використовувати три крапки «...», які самі по собі не виконують жодної дії, тому їх можна використовувати як альтернативу інструкції *pass*, зокрема замість програмного коду, який буде написано пізніше (примітка: To Be Done – слід реалізувати):

**def func1():**

**... # Альтернатива інструкції pass def func2():**

**...**

**func1() # Під час виклику не виконає жодних дій**

Три крапки може бути присутнім в одному рядку із заголовком інструкції і використовуватися для ініціалізації змінних, коли не потрібно вказувати значення якогось певного типу:

**def func1(): ... #Може бути присутнім в тому ж рядку**

**def func2(): ...**

**>>> X = ... # Альтернатива об'єкту None**

**>>> X**

**Ellipsis**

**Continue**. Інструкція *continue* виконує перехід на початок циклу. Вона іноді дозволяє уникнути використання вкладених інструкцій. У наступному прикладі цю інструкцію використовують для пропуску непарних чисел (цей фрагмент виводить парні числа менше «10» і більше або рівні «0»). Число «0» означає «хибність», а оператор % обчислює залишок від ділення, тому даний цикл виводить числа в зворотному порядку, пропускаючи значення, кратні 2 (він виводить 8 6 4 2 0):

**x = 10 while x:**

**x = x-1 # x -= 1**

**if x%2!=0: continue # непарне, пропустить вивеедення print(x, end= ' ')**

Останній приклад виглядав би зрозуміліше, якби інструкція *print* була складовою інструкції *if*: x = 10 while x:

**x = x-1**

**if x % 2 == 0: # парне? - вывести print(x, end=’ ‘)**

**Вreak. Інструкція *break* виконує негайний вихід** з циклу. Програмний код, розташований в циклі за цією інструкцією не виконується, якщо ця інструкція запущена. Нижче наведено інтерактивний цикл, який виконує введення даних за допомогою функції *input* і вихід з циклу, якщо у відповідь на запит імені буде введена рядок «*stop*»:

**>>> while 1:**

**... name = input('Enter name:')**

**... if name == 'stop‘: break**

**... age = input(‘Enter age: ')**

**... print(‘Hello', name, ‘=>‘, int(age) \*\* 2)**

**...**

**Enter name:mel**

**Enter age: 40**

**Hello mel => 1600**

**Enter name:bob**

**Enter age: 30**

**Hello bob => 900**

**Enter name:stop**

Цей приклад виконує перетворення рядка *age* в ціле число за допомогою функції *int*, перед тим як звести його до другого степеня (це необхідно, тому що функція *input* повертає результат введення користувача у вигляді рядка).

**Else**. При об'єднанні з частиною *else* інструкція *break* дозволяє позбутися від необхідності зберігати прапор стану пошуку. Наступний фрагмент визначає, чи є додатне ціле число простим числом, виконуючи пошук дільників більше за значення «1»:

**x = y // 2 # Для значень y>1 while x > 1:**

**if y % x == 0: # залишок**

**print(y, 'has factor', x)**

**break # Переступити блок else**

**x -= 1**

**else: # Нормальне завершення цикла print(y, ‘is prime’)**

Замість того щоб встановлювати прапор, який буде перевірений після закінчення циклу, досить вставити інструкцію *break* в місці, де буде знайдений дільник. При такій реалізації управління буде передано блоку *else*, тільки якщо інструкція *break* не була виконана, тобто коли з упевненістю можна сказати, що число є простим. Блок *else* циклу виконується також у разі, коли тіло циклу жодного разу не виконувалося, оскільки в цій ситуації інструкція *break* також не виконується. У циклах *while* це відбувається, коли перша ж перевірка умови в заголовку дає значення «хибність». Внаслідок цього в попередньому прикладі буде отримано повідомлення «is prime» (просте число), якщо спочатку *x* менше або дорівнює «1» (тобто, коли y=2). Блок else в циклах дозволяє обробити «інший» спосіб виходу з циклу, без необхідності встановлювати і перевіряти прапори або умови.

Цей приклад визначає прості числа, але недостатньо точно. Числа, менші «2», не вважають простими у відповідності із суворим математичним визначенням. Якщо бути більш точним, цей код також буде терпіти невдачі при від‘ємних значеннях і виконуватися успішно при використанні дійсних чисел без дробової частини. В Python замість оператора ділення / використовують оператор //, тому що тепер оператор / виконує операцію «істинного ділення» (початкове ділення необхідно, щоб відсікти залишок!).

Нехай ми створюємо цикл пошуку деякого значення в списку і після виходу з циклу необхідно дізнатися, чи було знайдено це значення. Це завдання можна вирішити таким чином:

**found = False**

**# прапор, щоб визначити, чи закінчився пошук успіхом while x and not found:**

**if match(x[0]): # шукане значення є першим?**

**рrint('Ni‘); found = True**

**else:**

**x = x[1:] # Видалити перше значення й повторити**

**if not found:**

**print('not found‘)**

Нижче наводиться еквівалентний фрагмент, де використано блок *else* в циклі:

while x: # Вийти, коли x спорожніє

if match(x[0]):

print('Ni‘); break # Вихід, в обхід блоку else

x = x[1:]

else:

print('Not found') #цей блок працює, якщо рядок x вичерпанo

**Приклад.** Функція *range*

1. Параметр i приймає значення в діапазоні [0, 10)

**for i in range(10):**

**print('i =', i)**

1. Параметр i приймає значення в діапазоні [5, 10)

**for i in range(5, 10):**

**print('i =', i)**

1. Параметр i приймає значення в діапазоні [5, 10) з кроком 2

**for i in range(5, 10, 2):   
 print('i =', i)**

1. Цикл буде повторюватися 3 рази, якщо користувач не завершить його раніше

for i in range(3):

response=input('Введіть stop, щоб зупинити цикл (інакше

що завгодно): ')

if response=='stop': break   
else:

# цю гілку виконують тільки якщо цикл не був перерваний print('Цикл сам був завершений')

print('Кінець програми')

5)for x in range(1, 11):

if x == 5: # пропускаємо число 5

continue

print('Поточне число дорівнює ', x)

6) for i in range(10):

for j in range(30):

print('\*', end='')

print()

**Завдання до лабораторної роботи:**

**Завдання 1. Одновимірні масиви (вектори)**

1. Використовуючи генератор випадкових чисел, заповнити список [*a*1, …, *an*] елементами:

*а*) дійсними числами, які лежать в діапазоні від 0 до 1;

*б*) цілими додатними та від‘ємними числами, які лежать в діапазоні від –

10 до 10 включно;

*в*) цілими додатними числами, які лежать в діапазоні від 0 до 50 включно.

Нехай задано список різних випадкових чисел [*a*1, …, *an*], значення *n* визначає користувач програми.

2. Задано список (*б*). Написати програму формування іншого списку, в якому усі елементи, які передують найбільшому від‘ємному елементу, замінити на значення їх квадратів.

1. Задано список (*б*). Написати програму формування іншого списку, в якому, якщо еленти заданого списку не утворюють послідовності, яка зменшується, то замінити його від‘ємні елементи одиницями.
2. Задано список (*б*). Написати програму формування іншого списку, в

якому переставити елементи таким чином, щоб спочатку були розташовані всі невід‘ємні елементи, а вкінці - від‘ємні елементи.

1. Задано список (*б*). Написати програму формування іншого списку, в якому елементи сформовані таким чином [*a*1, *an*+1, *a*2, *an*+2, …, *an*, *a*2*n*].
2. Задано список (*б*). Перевірити чи утворюють елементи заданого масиву послідовність, яка чітко зменшується або збільшується. Вивести відповідне повідомлення.
3. Задано список (*а*). За заданими дійсними числами *a0*, *a*1, *…*,*an*, *t* обчислити значення багаточлена *anxn + an-1 xn-1 +…+ a1 x + a0* і його похідної в точці *t*.
4. Задано список (*б*). Написати програму визначення суми всiх елементів, розміщених до останнього додатного елемента включно.
5. Задано список (*в*). Написати програму визначення суми лише тих елементів, які є непарними числами.
6. Задано список (*б*). Написати програму визначення добутку елементів, розміщених між максимальним за модулем та мінімальним за модулем елементами.
7. Задано список (*б*). Написати програму формування іншого списку, в якому елементи сформовані таким чином, що нульові елементи перенесено у хвіст списку.
8. Задано список (*б*). Написати програму визначення суми модулів елементів, розміщених після першого нульового.
9. Задано список (*б*). Написати програму визначення суми елементів, розміщених між першим та другим від‘ємними елементами.
10. Задано список (*б*). Написати програму формування іншого списку, в якому елементи сформовані таким чином: якщо хоча б одне значення елементів належить проміжку [*x, y*], то всі елементи, які не належать цьому проміжку, замінити на *z.*
11. Задано список (*б*). Написати програму визначення суми чисел цієї послідовності, розташованих між максимальним и мінімальним числами (до суми включити й обидва цих числа).
12. Задано різні два списки різних цілих випадкових чисел [*a*1, *a*2, …, *a3n*] (0.б). Написати програму визначення найменшого серед тих чисел першого списку, які не входять до другого (вважаючи, що хоча б одне таке число існує).
13. Задано список (*б*). Написати програму формування іншого списку, в якому всі від‘ємні елементи списку перенести в його початок, а всі інші – в кінець, сбурігаючи початкове взаємне розміщення як серед від‘ємних елементів, так и серед інших элементі.
14. Задано список (б). Написати програму, яка визначити максимальний елемент третього стовпчика та суму непарних елементів першого рядка.
15. Задано список (г). Написати програму, яка визначить рядок, сума елементів якого мінімальна. Вивести на екран початкову матрицю та визначений рядок. 12.
16. Задано список (г). Написати програму, яка визначить суму елементів кожного стовпчика (результат записати в інший список).
17. Задано список (в). Написати програму, яка визначить суму елементів тих рядків, у яких на побічної діагоналі стоять невід‘ємні числа.