Розділ 3

Умовне виконання

3.1 Логічні вирази

Логічним (булевим) називається вираз, який має значення «хибне» або «істинне» (True або False). Нижче зображені приклади з оператором «==», який порівнює два операнди і повертає значення True, якщо вони рівні, чи False, якщо ні:

```
>>> 5 == 5
True
>>> 5 == 6
False
```

True та False – значення, що належать до класу bool, вони не є рядками:

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

Оператор «==» є одним з *операторів порівняння*. До них також належать:

```
x != y               # x не дорівнює y
x > y                # x більше за y
x < y                # x менше за y
x >= y               # x більше або дорівнює y
x <= y               # x менше або дорівнює y
x is y               # x тотожне y
x is not y           # x не тотожне y
```

Зауважте, незважаючи на те, що вони можуть бути вам знайомими, для виконання однакових дій у Python та математиці використовуються різні знаки. Поширеною помилкою є використання одного знаку дорівнює (=) замість подвійного (==). Запам'ятайте, «=» – оператор присвоєння, «==» – оператор порівняння. Операторів «=<» та «=>» не існує.

3.2 Логічні оператори

Існує три *логічні оператори*: and, or та not. Семантика (значення) цих операторів подібна до значення цих слів в англійській мові: and – і, or – або, not – не. Наприклад,

x > 0 and x < 10

має значення True, тільки якщо x більше за 0 ***і*** менше pf 10.

n % 2 == 0 or n % 3 == 0

має значення True, якщо виконується ***будь-яка*** з умов, тобто числа діляться або на 2, або на 3 без остачі.

І останнє, оператор not заперечує логічний вираз, тож

not (x > y)

має значення True, якщо вираз x > y хибний; тобто, якщо x менше або дорівнює y.

Власне кажучи, операнди логічних операторів мали б бути логічними виразами, але у Python не так суворо щодо цього. Будь-яке ненульове число тлумачиться як «True».

```
>>> 17 and True
True
```

Така властивість може бути корисною, проте існують певні особливості, які можуть збити з пантелику. Краще, мабуть, поки уникайте цього, доки не будете повністю впевненими у ваших діях.

3.3 Умови з одним шляхом

Для того, щоб писати корисні програми, нам майже завжди потрібна можливість перевіряти умови і відповідно змінювати поведінку програми. У цьому нам допомагають *умовні інструкції.* Найпростіша з них – це інструкція if:

```
if x > 0 :
    print('x – додатне число')
```

Логічний вираз, який іде після інструкції if називається *умовою*. Інструкція if завершується знаком двокрапки (:), після неї робляться відступи в рядку (-ах).

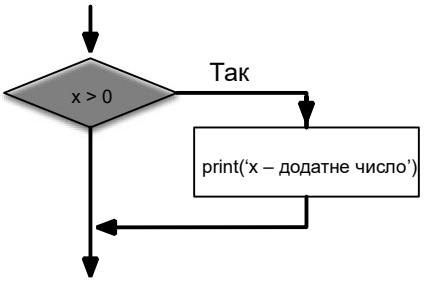

Рисунок 3.1: Логіка If

Інструкція в рядку(-ах) з відступом виконується, якщо логічна умова має значення True, в іншому разі, якщо умова має значення False, вона пропускається.

В інструкціях if така сама структура, як у визначеннях функцій чи циклах for[1]. Інструкція складається з рядка заголовка, який закінчується двокрапкою (:), після цього слідує блок з відступом. Такі інструкції називаються *складеними*, оскільки вони займають більше одного рядка.

Інструкцій в тілі може бути необмежена кількість, головне, аби була наявна принаймні одна. Іноді, зручно, коли в тілі інструкції відсутні (зазвичай це місце для ще не написаного коду). У такому разі, можна скористатися інструкцією pass, яка нічого не робить.

```
if x < 0 :
    pass          # потрібно обробити негативні значення!
```

Після введення інструкції if в інтерпретатор Python, знак підказки >>> зміниться на три крапки, це вказуватиме на те, що ви знаходитесь в середині блоку інструкцій. Нижче наведено приклад:

```
>>> x = 3
>>> if x < 10:
...     print('Малий')
...
Малий
>>>
```

Під час роботи в інтерпретаторі Python, у кінці блоку слід залишити порожній рядок, бо інакше виникне помилка:

```
>>> x = 3
>>> if x < 10:
...     print('Малий')
...     print('Готово')
  File "<stdin>", line 3
    print('Готово')
        ^
SyntaxError: invalid syntax
```

Під час написання та виконання скрипта, не обов'язково залишати порожній рядок в кінці блоку інструкцій, однак це полегшить прочитання вашого коду.

[1] Детальніше про: *функції* в розділі 4; *цикли* в розділі 5.

3.4 Умови з двома шляхами

Другою формою інструкції if є *альтернативне виконання, або умови з двома шляхами*, у якіх є два варіанти вибору. Умова визначає, який з них виконувати. Синтаксис виглядає так:

```
if x%2 == 0 :
    print('x – парне число')
else :
    print('x – непарне число')
```

Якщо остача ділення x на 2 дорівнюватиме 0, буде зрозуміло, що x – парне, тому програма виведе відповідне повідомлення. Якщо умова має значення False, виконується другий набір інструкцій.

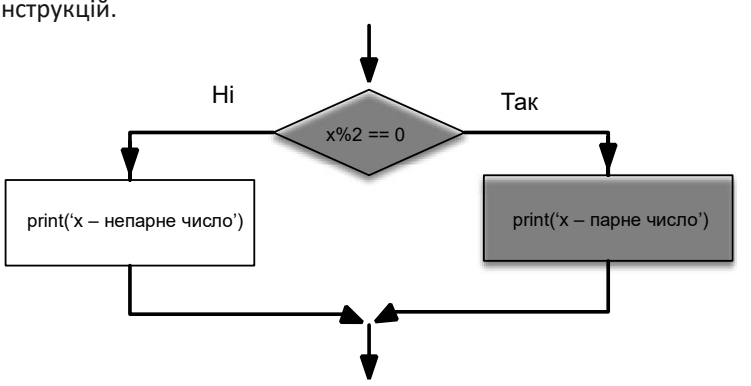

Рисунок 3.2: Логіка If-Then-Else

Оскільки умова може бути або True або False, лише одна альтернатива буде виконана, тобто обрано один з шляхів. Альтернативи називаються *гілками або шляхами*, оскільки вони розгалужуються в процесі виконання умов.

3.5 Множинне розгалуження

Бувають ситуації, у яких зустрічається більше двох варіантів вибору, у такому разі потрібно більше двох гілок. Одним зі способів вираження подібного обчислення – використання *ланцюжків*:

```
if x < y:
    print('x менше за y')
elif x > y:
    print('x більше за y')
else:
    print('x рівне y')
```

elif – це скорочення від «else if». І знову ж таки, виконуватиметься лише одна гілка.

Інструкції elif можна використовувати безліч разів. Інструкція else повинна розташовуватись в кінці, проте її наявність не обов'язкова.

```
if choice == 'a':
        print('Невдала спроба')
elif choice == 'b':
        print('Влучна спроба')
elif choice == 'c':
        print('Близько, проте неправильно')
```

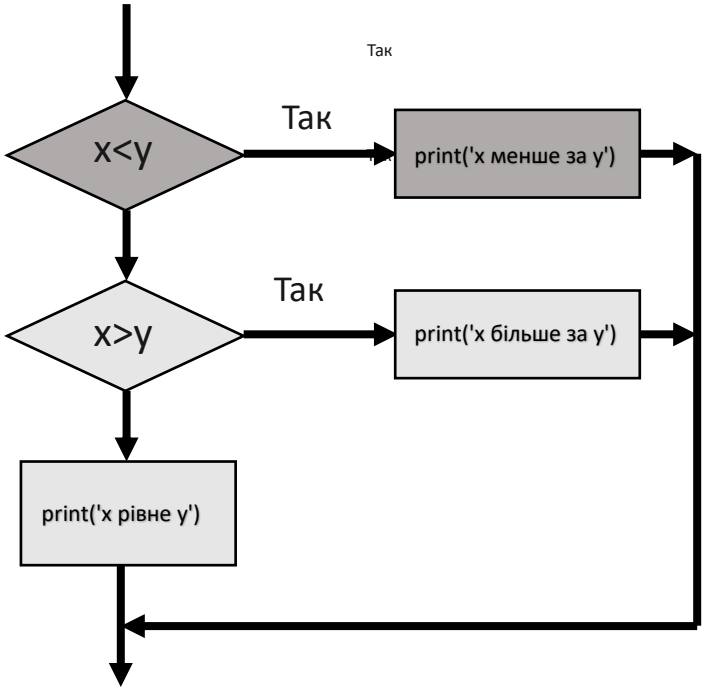

Рисунок 3.3: Логіка If-Then-ElseIf

Кожна умова перевіряється по порядку. Якщо перша умова має значення False, відбувається перевірка наступної і так далі. Якщо одна з них True, відповідно, виконується її гілка й умовна інструкція завершується. Навіть якщо є кілька умов зі значенням True, виконується лише перша гілка з них.

3.6 Вкладені умови

Одне умовне речення можна вкладати в інше. Приклад з трьома гілками можна записати таким чином:

```
if x == y:
        print('x рівне y')
else:
```

```
if x < y:
    print('x менше y')
else:
    print('x більше y')
```

Зовнішня умова складається з двох гілок. У першій гілці маємо просту інструкцію, а в другій гілці інструкцію if, у якій так само є дві гілки. У цьому прикладі, це прості інструкції, та ця властивість доступна і для умовних.

Незважаючи на те, що відступи між інструкціями роблять структуру коду зрозумілою, *вкладені умови* все одно важкі для прочитання. За можливості, радимо уникати їх.

Часто, за допомогою логічних операторів можна спростити вкладені умовні інструкції. Наприклад, наведений нижче код можна переписати всього лиш однією умовою:

```
if 0 < x:
    if x < 10:
        print('x – додатне одноцифрове число')
```

Інструкція print виконується лише після двох умов, тож те саме можна записати за допомогою оператора and:

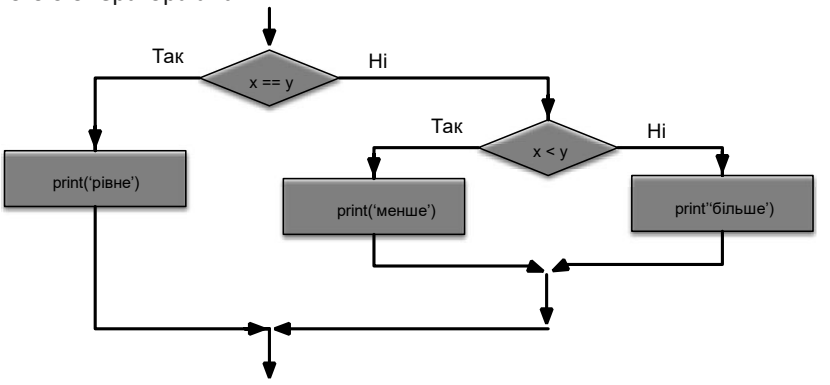

Рисунок 3.4: Вкладені інструкції If

```
if 0 < x and x < 10:
    print('x – додатне одноцифрове число')
```

3.7 Обробка винятків за допомогою конструкції try та except

Раніше ми зустрічали фрагмент коду, в якому використовувалися функції input та int для зчитування та парсування цілого числа, введеного користувачем. І також відразу побачили, яке підводне каміння там може бути:

```
>>> prompt = "What is the air velocity of an unladen swallow?\n"
>>> speed = input(prompt)
```

```
What is the air velocity of an unladen swallow?
What do you mean, an African or a European swallow?
>>> int(speed)
ValueError: invalid literal for int() with base 10:
>>>
```

Коли ми виконуємо ці інструкції в інтерпретаторі Python, ми отримуємо новий запит від інтерпретатора, думаємо «ой» і переходимо до наступної інструкції.

Проте, якщо під час роботи з цим кодом в скрипті Python виникне така сама помилка, скрипт відразу ж зупиниться з помилкою трасування і не виконає наступну інструкцію.

Розгляньте приклад програми для переведення температури зі шкали Фаренгейта до шкали Цельсія:

```
inp = input('Введіть температуру за шкалою Фаренгейта: ')
fahr = float(inp)
cel = (fahr - 32.0) * 5.0 / 9.0
print(cel)

# Код: http://www.py4e.com/code3/fahren.py
```

Якщо запустити цей код і ввести неправильні дані, він просто вийде з ладу і залишить недружелюбне повідомлення про помилку:

```
python fahren.py
Введіть температуру за шкалою Фаренгейта: 72
22.22222222222222

python fahren.py
Введіть температуру за шкалою Фаренгейта: fred
Traceback (most recent call last):
  File "fahren.py", line 2, in <module>
    fahr = float(inp)
ValueError: could not convert string to float: 'fred'
```

У Python вбудована структура умовного виконання для обробки подібних типів передбачуваних і непередбачуваних помилок, яка називається «try/except». Принцип дії try та except полягає у тому, що ви передбачаєте виникнення проблем при певній послідовності інструкцій, і додаєте кілька інструкцій, які виконуватимуться, якщо виникне помилка. Ці додаткові інструкції (блок except) ігноруються за відсутності помилки.

У Python, властивості try та except можна використовувати «для перестраховки» під час роботи з послідовністю інструкцій.

Програму для перетворення температури можна переписати таким чином:

```
inp = input('Введіть температуру за шкалою Фаренгейта: ')
try:
```

```
    fahr = float(inp)
    cel = (fahr - 32.0) * 5.0 / 9.0
    print(cel)
except:
    print('Введіть число')

# Код: http://www.py4e.com/code3/fahren2.py
```

Python починає з виконання послідовності інструкцій у блоці try. Якщо все гаразд, блок except пропускається, і програма продовжує роботу. Якщо у блоці try виникає передбачений заздалегідь виняток, який може викликати помилку, Python виходить з блоку try і виконує послідовність інструкцій блоку except.

```
python fahren2.py
Введіть температуру за шкалою Фаренгейта: 72
22.22222222222222

python fahren2.py
Введіть температуру за шкалою Фаренгейта: fred
Введіть число
```

Обробка винятку за допомогою інструкції try називається *перехопленням* виняткової ситуації. У цьому прикладі except виводить повідомлення про помилку. Загалом, перехоплення виняткової ситуації надає можливість розв'язати проблему, або спробувати ще раз, чи, принаймні, завершити програму коректно.

3.8 Обчислення логічних виразів за короткою схемою

Обчислення виразів, таких як x >= 2 and (x/y) > 2, у Python відбувається зліва направо. Згідно визначенню and, якщо x менше за 2, то вираз x >= 2 має значення False і, відповідно, весь вираз має значення False, незважаючи на відповідь виразу (x/y) > 2, чи то True, чи то False.

Коли Python встановлює, що обчислення решти логічного виразу не принесе жодного результату, він зупиняє його виконання і не обчислює іншу частину логічного виразу. Зупинка обчислення логічного виразу внаслідок встановлення загального значення називається обчисленням *за короткою схемою*.

Це може здаватись дрібницею, однак завдяки обчисленню за короткою схемою зумовлюється поява ефективної техніки шаблона *запобіжника*. Розгляньмо подану послідовність коду в інтерпретаторі Python:

```
>>> x = 6
>>> y = 2
>>> x >= 2 and (x/y) > 2
True
>>> x = 1
>>> y = 0
>>> x >= 2 and (x/y) > 2
False
```

```
>>> x = 6
>>> y = 0
>>> x >= 2 and (x/y) > 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>>
```

Обчислення третього прикладу завершилося збоєм, оскільки Python обчислював (x/y), де *y* дорівнює нулю, що спричинило появу помилки. А от другий приклад завершився вдало, оскільки перша частина виразу x >= 2 отримала значення False, тому, через правило *короткої схеми* програма не перейшла до наступної дії (x/y), і в результаті, ми уникнули помилки.

Побудувати логічний вираз для стратегічного розміщення запобіжника безпосередньо перед обчисленням, яке може призвести до помилки, можна наступним чином:

```
>>> x = 1
>>> y = 0
>>> x >= 2 and y != 0 and (x/y) > 2
False
>>> x = 6
>>> y = 0
>>> x >= 2 and y != 0 and (x/y) > 2
False
>>> x >= 2 and (x/y) > 2 and y != 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>>
```

У першому логічному виразі x >= 2 має значення False, тому обчислення зупиняється на and. У другому логічному виразі x >= 2 - True, проте y != 0 - False, тому ми так і не дійдемо до (x/y).

У третьому логічному виразі y != 0 обчислюється після (x/y), тому вираз завершується помилкою.

У другому виразі ми вказуємо, y != 0 запобіжником, який забезпечує можливість обчислення (x/y) лише у випадку, якщо y відмінне від нуля.

3.9 Налагодження програми

У трасуванні, яке відображає Python під час виникнення помилки, міститься багато інформації, однак її важко зрозуміти. Найбільш корисними пунктами, як правило, є:

- Вид помилки
- Місце помилки

Зазвичай синтаксичні помилки легко знайти, однак є кілька «але». Проблеми виникають з помилками табуляції та пробілів, оскільки вони невидимі, тобто ніяк не позначаються, тож ми звикли не помічати їх.

```
>>> x = 5
>>>  y = 6
   File "<stdin>", line 1
     y = 6
     ^
IndentationError: unexpected indent
```

У цьому прикладі проблема стосується того, що другий рядок має відступ на один пробіл. Але повідомлення про помилку вказує на y, це може заплутати. Загалом, повідомлення про помилки вказують на місце виявлення проблеми, хоча насправді вона могла бути раніше у коді, іноді у попередньому рядку.

3.10 Словник

body (тіло) Послідовність інструкцій у складеній інструкції.
boolean expression (логічний вираз) Вираз, який має значення True або False.
branch (гілка) Одна з альтернативних послідовностей інструкцій всередині умовної інструкції.
chained conditional (множинне розгалуження) Умовна інструкція з рядом альтернативних розгалужень.
comparison operator (оператор порівняння) Один з операторів, який порівнює операнди: ==, !=, >, <, >=, та <=.
conditional statement (умовна інструкція) Інструкція, яка контролює процес виконання відповідно до заданої умови.
condition (умова) Логічний вираз в умовній інструкції, який визначає, яка гілка виконується.
compound statement (складена інструкція) Інструкція, яка складається із заголовка та тіла. Заголовок закінчується двокрапкою (:). Тіло розміщується з відступом порівняно із заголовком.
guardian pattern (запобіжник) Місце, де будується логічний вираз з додатковими порівняннями, для реалізації властивостей обчислення за короткою схемою.
logical operator (логічний оператор) Оператор, який об'єднує логічні вирази: and, or та not.
nested conditional (вкладені умови) Умовна інструкція, що з'являється в одній з гілок іншої умовної інструкції.
traceback (трасування) Список функцій, що виконуються, який виводиться у разі виникнення виключення.
short circuit (коротка схема) Схема, за якою Python зупиняє обчислення посеред логічного виразу, оскільки визначив відсутність необхідності в обчисленні наступної частини виразу.

3.11 Вправи

Вправа 1: Перепишіть розрахунок заробітної плати таким чином, щоб працівник отримував 1,5-кратну погодинну ставку за понад 40 відпрацьованих годин.

Введіть години: 45
Введіть ставку: 10
Оплата: 475.0

Вправа 2: Перепишіть програму оплати праці, використовуючи метод try та except, таким чином, аби вона коректно обробляла введення нечислових даних, виводячи повідомлення про помилку і завершуючи роботу програми після цього. Нижче наведено два варіанти виконання програми:

Введіть години: 20
Введіть ставку: дев'ять
Помилка, будь ласка, введіть числове значення

Введіть години: сорок
Помилка, будь ласка, введіть числове значення

Вправа 3: Напишіть програму, яка запитуватиме оцінку у діапазоні від 0.0 до 1.0. Якщо число оцінки знаходиться за межами діапазону, виведіть повідомлення про помилку. Якщо результат знаходиться у діапазоні від 0.0 до 1.0, виведіть відповідь, згідно таблиці нижче:

Score	Grade
>= 0.9	A
>= 0.8	B
>= 0.7	C
>= 0.6	D
< 0.6	F

Enter score: 0.95
A

Enter score: perfect
Bad score

Enter score: 10.0
Bad score

Enter score: 0.75
C

Enter score: 0.5
F

Запустіть програму кілька разів, як показано вище, щоб протестувати різні значення вхідних даних.