Moduł 4 - Instrukcja warunkowa. Pętle.

1. Instrukcje warunkowe.

Do wersji 3.10 Python posiadał tylko jedną konstrukcję warunkową: **if/elif/else**. W wersji 3.10 wprowadzono również instrukcje **match/case**.

1.1 Instrukcja **if/elif/else**

Oto najprostsza postać instrukcji:

Listing 1

```
liczba1 = 1
liczba2 = 2

if liczba1 > liczba2:
    print("Pierwsza liczba jest większa")
```

Zobaczmy jak wygląda bardziej rozbudowana jej wersja wraz z obsługą danych wprowadzanych z klawiatury.

Listing 2

```
liczba = input('Podaj liczbę całkowitą ')
liczba = int(liczba)

if liczba < 10:
    print('To dość mała liczba')
elif 9 < liczba < 100:  # to jest wersja skrócona warunku
    print('To już całkiem duża liczba')
else:
    print('To musi być wielka liczba')
```

Aby budować bardziej złożone warunki używamy operatorów boolowskich, które zostały przedstawione w podrozdziale pt. 'Kilka słów o operatorach'.

Listing 3

```
if liczba < 10 and liczba > 15:
    print('Liczba nie jest z odpowiedniego przedziału')

# powyższy warunek można zapisać również tak
if 10 > liczba > 15:
    print('Liczba nie jest z odpowiedniego przedziału')
```

```
# możemy również sprawdzić warunek zawierania się elementu w kolekcji
zbior_dopuszczalny = [1, 3, 5, 7, 9]
if liczba not in zbior_dopuszczalny:
    print('Podana liczba nie znajduje się w zbiorze')
```

Jak zostało już wspomniane wcześniej Python posiada typ **None**, który jest tym samym co typ **Null** w wielu innych językach programowania oraz bazach danych. Ponownie odsyłam do podrozdziału *Kilka słów o operatorach* gdzie znajduje się informacja co jest traktowane jako None w Pythonie.

listing 4

```
liczba1 = 1
liczba2 = 2

if liczba1 > liczba2:
    print('Pierwsza liczba jest większa')

liczba = input('Podaj liczbę całkowitą ')
liczba = int(liczba)

if liczba < 10:
    print('To dość mała liczba')
elif 9 < liczba < 100:  # to jest wersja skrócona warunku
    print('To już całkiem duża liczba')
else:
    print('To musi być wielka liczba')

if liczba < 10 or liczba > 15:
    print('Liczba nie jest z odpowiedniego przedziału')

# możemy również sprawdzić warunek zawierania sie elementu w kolekcji
zbior_dopuszczalny = [1, 3, 5, 7, 9]
if liczba not in zbior_dopuszczalny:
    print('Podana liczba nie znajduje się w zbiorze')

nic = None
pusty_ciag = ''

if not nic:
    print('None to False')
if not pusty_ciag:
    print('Pusty ciąg to False')
if nic == pusty_ciag:
    print('None i pusty ciąg to boolowskie False, ale nie są sobie równe')
# jeżeli chcemy sprawdzić, czy ciąg jest pusty
if pusty_ciag == '':
    print('To jest pusty ciąg')
```

Instrukcję if można również znaleźć w wielu skryptach Pythona sprawdzającą dość tajemniczą własność if **name** == '**main**'. Poniżej wyjaśnienie stosownym przykładem.

Listing 5

```
# Jest również specjalne zastosowanie instrukcji if
# poniższy zapis powoduje, że kod umieszczony wewnątrz tego bloku
# zostanie uruchomiony tylko w przypadku, gdy plik zostanie
# uruchomiony bezpośrednio, tak jak w tym przypadku.
# Jeżeli plik zostanie zaimportowany, to kod nie zostanie uruchomiony

if __name__ == '__main__':
    pass

# możemy też sprawdzić jaką wartość ma zmienna specjalna __name__
print(__name__)
# instrukcja pass nie robi nic, ale jeżeli wymagany jest tutaj kod, żeby
# spełnić wymogi składniowe to możemy jej użyć
```

1.2 Instrukcja **match/case**

Instrukcja **match/case** (opisana w: https://peps.python.org/pep-0622/) pełni podobną funkcję jak **if/elif/else** jednak posiada znaczną przewagę nad nią jeżeli chodzi o określenie warunków. I tak wyciągając fragment z dokumentacji wzorcem (czyli tym co możemy umieścić po słowie `case`) może być:

- sekwencja, która może być rozpakowana (patrz rozpakowanie listy, krotki),
- mapowanie z konkretnymi kluczami (np. słownik),
- instancja podanej klasy (opcjonalnie z konkretnymi atrybutami)
- wartość (to jak w if)
- symbol wieloznaczny (ang. wildcard)

Listnig 6

```
user = input('Kto chce się zalogować? (admin|user)\n')

# przykład z dopasowaniem do wartości (można zastąpić if/else)
match user:
    case 'admin':
        print('Przekierowuję do panelu administratora ...')
    case 'user':
        print('Przekierowuję na stronę sklepu ...')
    case _:
        print('Błędna nazwa użytkownika!')

# przykład z rozpakowaniem sekwencji (tu listy)
command = 'remove plik.txt log.txt dane.csv'

match command.split():
    case ['show']:
```

```
            print('Wylistuj wszystkie pliki i foldery: ')
            # kod
        case ['remove', *files]:
            print('Usuwanie plików: {}'.format(files))
            # kod
        case _:
            print('Nieznane polecenie')

# output
# Usuwanie plików: ['plik.txt', 'log.txt', 'dane.csv']
```

2. Pętla `for` oraz `while`.

W Pythonie mamy do dyspozycji dwie pętle: for oraz while, przy czym ta pierwsza jest zdecydowanie bardziej „popularna". Przykład zastosowania pętli `for` znalazł się już przy okazji prezentacji funkcji `range`. Dla przypomnienia w poniższych przykładach również pojawi się jej zastosowanie.

Listing 6

```
# for z funkcją range
for i in range(3):
    print(i)

# for dla listy
lista = [4, 5, 6]
for i in lista:
    print(i)

# a gdybyśmy chcieli zwracać również index elementów listy ?
for index, wartosc in enumerate(lista):
    print(f'{index} -> {wartosc}')

# a mozna jeszcze tak, gdyż funkcja enumerate wypakowuje każdy element listy
# w postaci krotki (index, wartość_z_tablicy)
for krotka in enumerate(lista):
    print(f'{krotka[0]}-> {krotka[1]}')

print(type(krotka[0]))
print(type(krotka[1]))
print(type(krotka))

# pętla for i słowniki
# jeżeli nie wskażemy pętli for czy chcemy iterować po kluczach czy wartościach
# to domyślnie zostaną wybrane klucze
slownik = {'imie': 'Marek', 'nazwisko': 'Kowalski', 'plec': 'mezczyzna'}
for key in slownik:
    print(key)

for val in slownik.values():
    print(val)
```

```
for key, value in slownik.items():
    print(f'{key} -> {value}')

for key in slownik:
    print(f'{key} -> {slownik[key]}')
```

Postać Pythonowej pętli while nie różni się od jej sposobu działania w innych językach.

Listing 7

```
# pętla while
counter = 0
while True:
    counter += 1
    if counter > 10:
        break

counter = 0
while counter < 5:
    print(f'{counter} mniejsze od 5')
    counter += 1

# pętla while nadaje się dobrze w sytuacji, kiedy nie wiemy kiedy (nie
# znamy liczby iteracji) się ona zakończy, np. przy pobieraniu danych
# wejściowych w oczekiwaniu na podanie komendy równej warunkowi stopu pętli

lista = []
print('Podaj liczby całkowite, które chcesz umieścić w pętli.')
print('Wpisz "stop" aby zakończyć')
while True:
    wejscie = input()
    if wejscie == 'stop':
        break
    lista.append(int(wejscie))

print('Twoja lista -> ' + repr(lista))
```

W tym miejscu należy wspomnieć o instrukcji `break` oraz `continue`, które możemy umieszczać wewnątrz pętli. `Break` powoduje zakończenie pętli (tylko tej, w bloku której znalazła się instrukcja) natomiast `continue` kończy przebieg aktualnej iteracji pętli (czyli to, co jest za `continue` się nie wykona) i rozpoczyna kolejną iterację.

3. Wprowadzanie danych ze standardowego wejścia.

Do wprowadzania danych możemy użyć funkcji **input()**.

Listing 8

```
a = input("Tu jest jakiś komunikat np. Podaj liczbę\n")
print(a)

# Możemy użyć też komend readline() i write(s), które są w module sys
import sys

print("Podaj jakiś tekst")
s = sys.stdin.readline() #Wczytuje wiersz
print("Twój tekst to: " + s)
# Do wydruku można użyć też komendy write np.
sys.stdout.write(s)
```

> Zadania

Zadanie 1
Napisz skrypt, który pobiera od użytkownika zdanie i liczy w nim spacje. Wynik wyświetla na ekranie (użyj instrukcji input).

Zadanie 2
Napisz skrypt, który pobiera od użytkownika dwie wartości i mnoży je przez siebie. Wynik wyświetla na ekranie (ale użyj instrukcji `readline()` i `write()` z modułu `sys`).

Zadanie 3
Napisz skrypt, który pobiera od użytkownika trzy liczby a, b i c. Sprawdza następujące warunki:

- czy a zawiera się w przedziale (0,10]
- oraz czy jednocześnie a>b lub b>c.

W obu przypadkach (jeżeli warunki są spełnione lub nie są spełnione) należy wyświetlić odpowiedni komunikat na ekranie.

Zadanie 4
Napisz pętlę, która wyświetla liczby podzielne przez 5 z zakresu [0,50]

Zadanie 5
Napisz pętle (while), która pobiera liczby od użytkownika i wyświetla ich kwadraty na ekranie. Liczby pobierane są w postaci oddzielonej spacjami. Pętla kończy działanie po wpisaniu słowa `quit`.

Zadanie 6
Napisz skrypt, który odczytuje liczby od użytkownika i umieszcza je na liście. Liczby dodajemy do momentu wpisania słowa 'stop' zamiast liczby. Wykorzystaj pętle while. Po wpisaniu stop wyświetl listę liczb.

Zadanie 7
Napisz skrypt, który odczytuje od użytkownika liczbę wielocyfrową i sumuje jej cyfry. Wynik wyświetla na ekranie. Napisz dwa rozwiązania: jedno z uzyciem pętli `for` a drugie z użyciem pętli `while`.

Zadanie 8
Napisz skrypt, który rysuje wieżę z literek. Użytkownik podaje wysokość wieży, ale nie więcej jak 10.

```

```

```
A
AA
AAA
AAAA
AAAAA
AAAAAA
```

Zadanie 9
Napisz skrypt, który wyświetla i oblicza tabliczkę mnożenia od 1 do 100 w formie znanej z lekcji matematyki w szkole podstawowej.

Zadanie 10
Napisz skrypt, który rysuje diament. Użytkownik podaje wysokość nie mniej jak 3 i nie więcej jak 9, ale dopuszczamy tylko nieparzystą wysokość.

Przykład wyjścia dla `wysokosc = 3`

```
  o
 ooo
  o
```

oraz `wysokosc = 5`

```
    o
   ooo
  ooooo
   ooo
    o
```

itd.