Faculty of Geodesy
and Cartography
WARSAW UNIVERSITY OF TECHNOLOGY

Informatyka Geodezyjna - wykłady/ćwiczenia, rok akad. 2020-2021

Wyk. 4: Python - instrukcje warunkowe i kontrolne

Kinga Węzka
kinga.wezka@pw.edu.pl
Katedra Geodezji i Astronomii Geodezyjnej

Warsaw University
of Technology

1. Sterowanie wykonywaniem programu
2. Instrukcje warunkowe: if/elif/else
3. Operatory w instrukcjach warunkowych
4. Operatory wykorzystywane w instrukcjach warunkowych
5. Instrukcje warunkowe: zapis jednowierszowy
6. Operatory trójskładnikowe (ang. ternary operators)
7. Switch/Case w Pythonie
8. Instrukcje warunkowe – przykłady

- Instrukcje warunkowe: **if/elif/else**
- Instrukcje kontrolne: **break**, **continue** i **pass**
- Pętle: **for while**

Instrukcje warunkowe

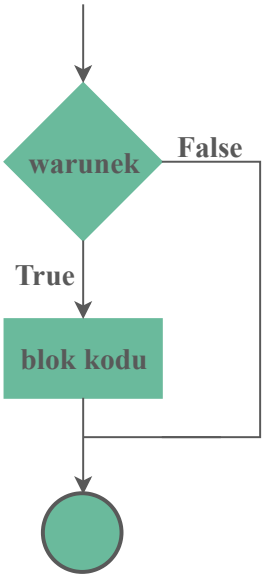
Instrukcje warunkowe inaczej sterujące to jedna z podstaw w każdym języku programowania. Działają dokładnie jak nazwa wskazuje – **instrukcje lub blok instrukcji wykonuje się tylko, gdy określony warunek (lub zestaw warunków) jest spełniony**. W uproszczeniu instrukcja `if` Pythona wybiera działanie, które należy wykonać.

- **wykonania warunkowe:** `if`
- **wykonania warunkowe alternatywne:** `if` / `else`
- **wykonania warunkowe łańcuchowe:** `if` / `elif` / `elif` /... / `else`
- **wykonania warunkowe zagnieżdżone:** *nested*

Język Python posiada tylko jedną wbudowaną instrukcję warunkową i jest nią instrukcja **if/elif/else**. Nie znajdziemy tutaj konstrukcji case/switch.

- `if` - *jeżeli* - blok instrukcji wykona się tylko wtedy, gdy **instrukcja warunku** będzie prawdziwa (`True`).
- `elif` - else if - *inaczej* - blok kodu wykona się wtedy gdy poprzednie instrukcje były fałszywe (`False`)
- `else` - *w przeciwnym wypadku* - blok instrukcji wykona się jeśli powyższe warunki nie zostały spełnione. Użycie `else` nie wymaga definicji warunku i jest opcjonalny.

```
1 if <instrukcja warunku>:
2     <blok kodu: jesli warunek = True>
3 elif <instrukcja warunku>: # jesli poprzednie False
4     <blok kodu: jeśli warunek = True>
5 else: # powyższe warunki są False
6     <blok kodu:>
```



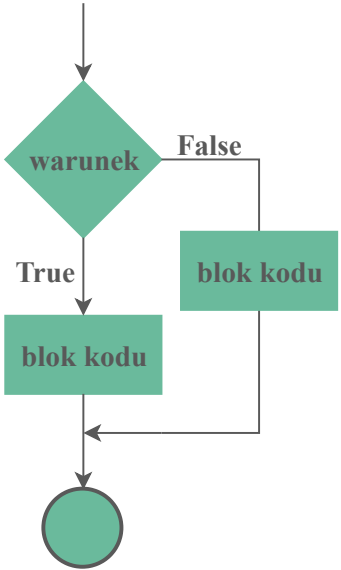

```
if <instrukcja warunku>:
    <blok kodu: jesli warunek = True>
```

In[1]:

```
a = 4
b = 0
wynik= 'Nie dziel przez zero!'
if b != 0:
    wynik = a/b # jeśli warunek True
print(wynik)
```

Out[1]:

```
Nie dziel przez zero!
```



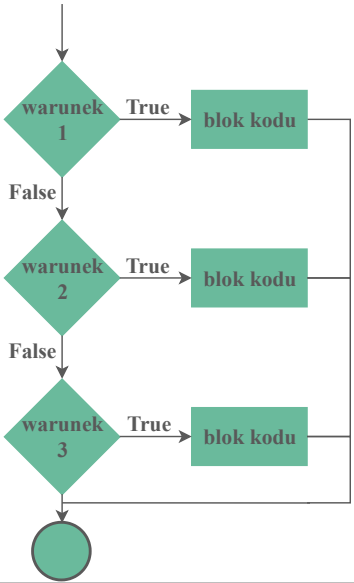

```
1 if <instrukcja warunku>:
2     <blok kodu: jesli warunek = True>
3 else: # powyższe warunki są False
4     <blok kodu:>
```

In[2]:

```
1 a = 4
2 b = 0
3 if b != 0:
4     wynik = a/b # jeśli warunek True
5 else:
6     wynik= 'Nie dziel przez zero!'
7 print(wynik)
```

Out[2]:

```
1 Nie dziel przez zero!
```



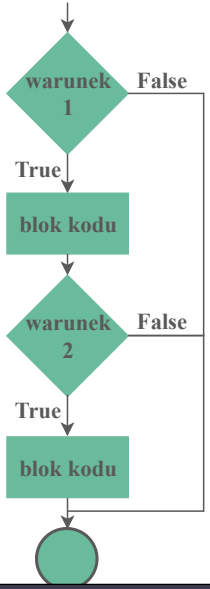

```
1 if <instrukcja warunku>:
2     <blok kodu: jesli warunek = True>
3 elif <instrukcja warunku>: # poprzednie False
4     <blok kodu: jeśli warunek = True>
5 else: # powyższe warunki są False
6     <blok kodu:>
```

In[3]:

```
1 a, b = 4, 0
2 if b > 0:
3     print(a / b) # jeśli warunek True
4 elif b < 0:
5     print(a + b) # jeśli warunek True
6 else:
7     print('Nie dziel przez zero!)
```

Out[3]:

```
1 Nie dziel przez zero!
```




```
if <instrukcja warunku>:
    <blok kodu: jesli warunek = True>
    if <instrukcja warunku>:
        <blok kodu: jesli warunek = True>
```

In[4]:

```
x, y = 4, 3
if x == y:
    print('x i y są równe')
else:
    if x < y:
        print('x jest mniejsze niż y')
    else:
        print('x jest większe niż y')
```

Out[4]:

```
x jest większe niż y
```

operator	znaczenie
`==`	równe co do wartości
`!=`	nie jest równe co co wartości
`> , <`	większe , mniejsze
`>= ,<=`	większe lub równe, mniejsze lub równe
`and, or, not`	*i, lub, nie* – operatory logiczne
`in, not in`	*w, nie w* – operatory przynależności, sprawdzenie elementów w sekwencji
`is, is not`	*jest, nie jest* – operatory tożsamości, sprawdza wskazanie obiektu

- Operator `==`: **równość**: porównuje wartości wartości dwóch obiektów. `==`, jeśli obiekty, do których odwołują się zmienne, są równe co do wartości.
- Operator `is`: **tożsamości**, `id(a) == id(b)` : porównuje czy obiekty są tożsame tzn. `is` zwróci True, jeśli dwie zmienne wskazują na ten sam obiekt.

In[5]:

```
1 i = 5
2 j = 5.0
3 if (i == j):
4     print('i == j')
5 if (i is j):
6     print ('i is j!')
```

Out[5]:

```
1 i == j
```

In[6]:

```
1 val1 = 1000
2 val2 = val1
3 if(val1 == val2):
4     print('val1 == val2')
5 if(val1 is val2):
6     print('val1 is val2')
```

Out[6]:

```
1 val1 == val2
2 val1 is val2
```

```
1 x == y # rowne co do wartosci
2 x != y # x nie jest równe y
3 x > y  # x jest większe niż y
4 x < y  # x jest mniejsze niż y
5 x >= y # x jest większe niż y lub równe y
6 x <= y # x jest mniejsze niż y lub równe y
```

Zwracają warość: True/False

```
x or y      # Logiczna operacja OR (lub)
x and y     # Logiczna operacja AND (i)
not x       # Logiczna negacja
```

Przykłady:

```
In [1]: a = ''
In [2]: bool(a)
Out[2]: False
In [3]: b = 1
In [4]: bool(b)
Out[4]: True
In [5]: b and a
Out[5]: ''
In [6]: a or b
Out[6]: 1
```

```
is  # tożsamość obiektów, porównuje wynik funkcji id() - adres w pamieci
is not # sprawdza czy obiekty nie sa tożsame
```

Przykłady:

```
In [1]: a = 7
In [2]: b = 7.0
In [3]: a == b
Out[4]: True
In [5]: a is b
Out[5]: False
```

- Operator równości == sprawdza równość co do wartości. Python wykonuje test równości, porównując rekurencyjnie wszystkie zagnieżdżone obiekty (Lutz, 2011, p.286) .
- Operator tożsamości `is` sprawdza identyczność obiektów. Python sprawdza, czy dwa obiekty są tak naprawdę jednym (to znaczy znajdują się pod jednym adresem w pamięci).

```
in      # czy obiekt jest zawarty w innym obiekcie (np. wartość w liście)
not in  # czy obiekt nie jest zawarty w innym obiekcie
```

Przykłady:

```
In [1]: 'a' in 'ala'
Out[1]: True
In [2]: 'a' not in 'ala'
Out[2]: False
```

Testy przynależności wykorzystywane są często **dla sekwencji lub kolekcji** (ciągi tekstowe, krotki, listy) w celu sprawdzenia czy dana wartość znajduje się w zbiorze.

Blok kodu w instrukcji jest zazwyczaj wyodrębniony **indentacją**. W specjalnych przypadkach ciało instrukcji może pojawić się w tym samym wierszu co jej nagłówek (Lutz, 2011):

- ciało instrukcji nie zawiera żadnych instrukcji złożonych, mogą się tam znajdowac jedynie proste instrukcje np. `print()`, `return()`
- ciało instrukcji jest krótkim zapisem

```
1 if <warunek> : <rob_cos>
```

Standardowy zapis:

```
1 if a > 8:
2     b = a + a
3 elif a < 8 :
4     b = a - a
5 else:
6     b = a
```

Zapis jednowierszowy

```
1 if a > 8 : b = a + a
2 elif a < 8: b = a - a
3 else: b = a
```

Operatory trójskładnikowe w Pythonie są zwięzłymi wyrażeniami warunkowymi. Są to operatory, które testują warunek i na tej podstawie oceniają wartość (działają od Pytona 2.4). Właściwe wykorzystanie operatora trójskładnikowe pozwala zmniejszyć rozmiar kodu i poprawić czytelność. **PEP 308 – Conditional Expressions**: (https://www.python.org/dev/peps/pep-0308/) .

```
<rób jeśli warunek True > if <warunek> else <rób jesli warunek False >
```

Na przykład porównanie dwóch liczb całkowitych:

Standardowy zapis:

```
a,b = 2,3
if a > b:
    print(a)
else:
    print(b)
```

Zapis operatorem trójskładnikowym

```
print(a if a > b else b)
# odpowiednik w C++ : max=(a>b)?a:b
```

Język Python posiada tylko jedną wbudowaną instrukcję warunkową i jest nią instrukcja **if/elif/else**. Nie znajdziemy tutaj konstrukcji case/switch.

In[7]:

```
def week(i):
    switcher={0:'Sunday',
        1:'Monday',
        2:'Tuesday',
        3:'Wednesday',
        4:'Thursday',
        5:'Friday',
        6:'Saturday'}
    return switcher.get(i,"Invalid day of week")

print(week(3)) # wywołanie funkcji
```

In[8]:

```
a = 2
if a > 1:
    a = a+1
if a > 2:
    a = a+1
print(a)
```

Out[8]:

```
4
```

In[9]:

```
a = 2
if a > 1:
    a = a+1
elif a > 2:
    a = a+1
print(a)
```

Out[9]:

```
3
```

- Przypadek po lewej jest równoważny dwóm różnym **if** z pustymi instrukcjami **else** (else: pass); w drugim przypadku **elif** jest częścią pierwszej instrukcji **if**.
- W przykładzie w przebiegu programu dwa warunki zostana spełnione.
- W przykładzie po prawej **elif** jest realizowany tylko jeśli warunek **if** nie zostanie spełniony

- `str.isdigit()` - wbudowana (*build-in*) – metoda typu tekstowego, zwraca `True` jeśli sprawdzany obiekt jest cyfrą (inne: `isalnum`, `isalpha`, `isdecimal`, `isnumeric` etc.)

In[10]:

```
a = '12tak'
if a.isdigit(): # cyfrą?
    print('a jest liczbą')
    if int(a) > 5:
        print('if a > 5: ')
    elif int(a) < 5:
        print('elif a < 2:')
    else:
        print('else')
else:
    print('a - tekst ')
```

Out[10]:

```
a - tekst
```

In[11]:

```
a = '12'
if a.isdigit(): # cyfrą?
    print('a jest liczbą')
    if int(a) > 5:
        print('if a > 5: ')
    elif int(a) < 5:
        print('elif a < 2:')
    else:
        print('else')
else:
    print('a - tekst')
```

Out[11]:

```
a jest liczbą, if a > 5:
```

In[12]:

```
imie = "Jan"
wiek = 23

# sprawdza czy zmienna imie=Jan i zmienna wiek=23.
if imie == "Jan" and wiek == 23:
    print("Nazywasz sie Jan i masz 23 lata.")

# sprawdza czy zmienna imie=Jan lub imie=Robert.
if imie == "Jan" or imie == "Robert":
    print("Nazywasz sie Jan lub Robert")
```

Out[12]:

```
Nazywasz sie Jan i masz 23 lata
Nazywasz sie Jan lub Robert
```

In[13]:

```
imie = "Robert"
lista = ["Jan", "Robert", "Kasia", "Ola"]
if imie in lista: # True of False: sprawdza czy zmienna imię
    jest w zbiorze (lista)
    print("Imię Robert znajduje się na liście (lista)")
```

Out[13]:

```
Imię Robert znajduje się na liście (lista)
```

In[14]:

```
klucz = input('podaj klucz: ')
s1 = {'k1':2, 'k2':32, 'k3':15}
lista_kluczy = s1.keys()
if klucz in lista_kluczy:
    print("Podany klucz znajduje się w słowniku: s1")
```

Out[14]:

```
Podany klucz znajduje się w słowniku: s1
```

M. Lutz. *Python. Wprowadzenie*. Helion, 2011.

Dziękuje za uwagę

Kinga Węzka
Gmach Główny PW – pok 38/6
kinga.wezka@pw.edu.pl