



Wrocław  
University  
of Science  
and Technology

# Wstęp do programowania

INP003203L

Semestr zimowy 2020/2021

**Poniedziałek, 7:30 - 9:00**

**sala wirtualna**

**– zajęcia online**

**Sylwia Majchrowska**

**[sylwia.majchrowska@pwr.edu.pl](mailto:sylwia.majchrowska@pwr.edu.pl)**

***<https://majsylw.netlify.app/teaching/>***  
**pokój 213, budynek L-1**



# Plan na dziś

1. Operatory relacji
2. Operatory logiczne
3. Instrukcje warunkowe:
  - if
  - if-else
  - if-elif-else
4. Tablica ASCII
5. Porównywanie łańcuchów znaków



# Sprawdzanie warunków

## - pytania i odpowiedzi

- Zazwyczaj programista pisze program, który umożliwia zadanie pewnych pytań. Komputer wykonuje program i dostarcza konkretnych odpowiedzi. Najczęściej na podstawie tych odpowiedzi planowane są kolejne akcje (choćby wyświetlenie stosownych komunikatów). Jedynymi odpowiedziami jakie może udzielić komputer są: tak (**True**) i nie (**False**).
- Instrukcje warunkowe lub inaczej instrukcje sterujące pozwalają na wykonanie pewnej instrukcji (lub bloku instrukcji) gdy określony warunek (lub zestaw warunków) jest spełniony.

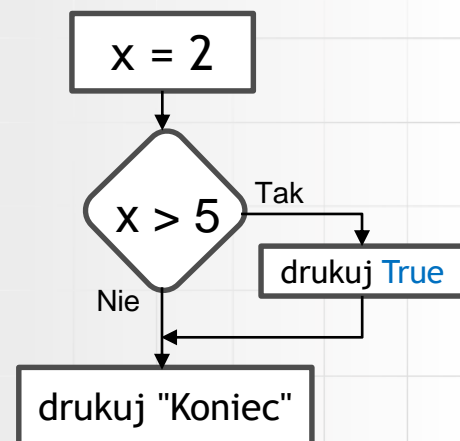
```
x = 2
if x > 5:
    print(x > 5)
print("Koniec")
```

słowo kluczowe

dwukropek

warunek

wnętrze instrukcji warunkowej (wydzielone przez 4 spacje lub 1 tabulator)





# Typ danych: bool

## - możliwe odpowiedzi

- bool lub boolean, wartość logiczna, tak/nie, prawda(True)/fałsz(False)  
*<class 'bool'>*
- Algebra boolowska – część algebry, w której używane są tylko dwie wartości True (zapisywana jako 1) oraz False (zapisywana jako 0) – nazwa pochodzi od matematyka Georga Boole

```
>> print(True > False) # True  
>> print(True < False) # False
```

Jest jeszcze jeden specjalny literał używany w Pythonie: literał **None**. Ten literał jest tak zwanym obiektem **NoneType** i jest używany do reprezentowania braku wartości.



# Operatory relacji: operator ==

## - stawiane pytania

- Pytanie: *Czy te dwie wartości są równe?*
- Zadając tego typu pytanie używamy operatora == (równe równe)
- Zapamiętaj:
  - operator =: operator przypisania, pozwala nadać pewnej zmiennej jakąś wartość, np.

```
>> a = 2 # w zmiennej a zapisano wartość 2  
>> b = a # w zmiennej b zapisano wartość 2
```

- operator ==: operator równości, sprawdza czy wartości są równe, np.

```
>> b == a # czy wartość zapisana w b równa się wartości zapisanej w a? Odp. True
```

Ćwiczenie 6.1 Co jest wynikiem zapytań?

```
>> 2 = 2 # SyntaxError  
>> 2 == 2 # True  
>> 2 == 2. # True  
>> 1 == 2 # False  
>> 2 == '2' # False  
>> 2 == int('2') # True
```

### Priorytet operatorów

```
a = 22  
b = 11  
a == 2 * b # True  
a == (2 * b) # True
```



# Operatory relacji: operator !=

## - stawiane pytania

- Pytanie: *Czy te dwie wartości różnią się?*
- Zadając tego typu pytanie używamy operatora != (różne)
- Zapamiętaj: operator !=: operator różności, sprawdza czy wartości są różne

```
>> a = 2 # w zmiennej a zapisano wartość 2
>> b = a # w zmiennej b zapisano wartość 2
>> b != a # czy wartość zapisana w b różni się od wartości zapisanej w a? Odp. False
```

Ćwiczenie 6.2 Co jest wynikiem zapytań?

```
>> 2 != 2 # False
>> 2 != 2. # False
>> 1 != 2 # True
>> 2 != '2' # True
>> 2 != float('2') # False
>> 0.1 + 0.2 != 0.3 # True
```

~~help("<>")~~



# Operatory relacji: >, >=, <, <=

## - stawiane pytania

- Pytanie: *Czy pierwsza wartość jest większa od drugiej?*
  - operator >: operator większości, sprawdza czy pierwsza wartość jest większa do drugiej
- Pytanie: *Czy pierwsza wartość jest większa lub równa drugiej?*
  - operator >=: operator większe lub równe, sprawdza czy pierwsza wartość jest większa lub równa drugiej
- Pytanie: *Czy pierwsza wartość jest mniejsza od drugiej?*
  - operator <: operator mniejszości, sprawdza czy pierwsza wartość jest mniejsza do drugiej
- Pytanie: *Czy pierwsza wartość jest mniejsza lub równa drugiej?*
  - operator <=: operator mniejsze lub równe, sprawdza czy pierwsza wartość jest mniejsza lub równa drugiej

```
>> x = 2 > 5
>> print(x)           # False
>> print(x * 1)       # 0
>> print(x + 1)       # 1
```

```
>> x = 2 < 5
>> print(x)           # True
>> print(x * 1)       # 1
>> print(x + 1)       # 2
```



# Operatory relacji

## - przykład 6.3

Napisz dwulinijkowy program który pozwoli na pobranie całkowitego parametru  $n$  z klawiatury, a następnie wyświetli True lub False w zależności czy  $n$  będzie mniejsze od 100 albo większe lub równe 100.

```
n = int(input("Podaj n: "))  
print("Czy n jest mniejsze od 100?", n < 100)
```





# Operatory logiczne: and, or i not

## - stawiane pytania

- Pytanie: *Czy jakieś wyrażenie lub/i jakieś drugie wyrażenie jest prawdziwe?*
  - and: operator koniunkcji – sprawdza czy oba wyrażenia są prawdziwe
  - or: operator alternatywy – sprawdza czy którekolwiek wyrażenie jest prawdziwe
- Pytanie: *Czy jakieś wyrażenie jest nieprawdziwe?*
  - not: operator negacji – sprawdza czy jakieś wyrażenie nie jest prawdziwe

### Koniunkcja - and

<wyrażenie A>	<wyrażenie B>	A and B
True	True	True
True	False	False
False	True	False
False	False	False

### Alternatywa - or

<wyrażenie A>	<wyrażenie B>	A or B
True	True	True
True	False	True
False	True	True
False	False	False

### Negacja - not

<wyrażenie>	not <wyrażenie>
True	False
False	True



# Wyrażenia równorzędne

## - stawiane pytania

- Zaprzeczanie

- Podwójne zaprzeczenie (potrójne, poczwórne ect.)

```
>> i = 1  
>> j = not not i      # True
```

- Operatory mniejszości/większości <-> większe lub równe/mniejsze lub równe

```
>> print(not (var <= 0))  
>> print(var > 0)
```

- Operator równości <-> nierówności

```
>> print(var != 0)  
>> print(not (var == 0))
```

- Prawa De Morgana

```
not (p and q) == (not p) or (not q)  
not (p or q) == (not p) and (not q)
```



# Testy przynależności i tożsamości

## - stawiane pytania

- Operator przynależności **in** oraz jego zaprzeczenie **not in**
  - Wyrażenie `x in s` jest prawdziwe, jeśli `x` jest elementem zbioru `s`, a fałszywe w przeciwnym wypadku.

```
>> 'i' in 'napis'           # True
>> 'i' not in 'napis'      # False
>> 1 in 123                 # Error
>> 1 in [1, 2, 'txt']      # True
```

- Operator tożsamości **is** oraz jego zaprzeczenie **not is**
  - `x is y` jest prawdziwe wtedy i tylko wtedy, gdy `x` i `y` są tymi samymi obiektami.

```
>> i = 1
>> i is int                 # False
>> type(i) is int          # True
```

- Czy `is` oraz `==` to te same operatory? – Nie, znak `==` sprawdza czy po obu jego stronach widnieje ta sama wartość, a `is` sprawdza czy obie zmienne wskazują na ten sam obiekt

```
>> 'aa' == 'aa'           # True
>> 'aa' is 'aa'           # True
```

```
>> 'aa' == 'a' * 2        # True
>> 'aa' is 'a' * 2        # True
```

```
>> x = 'a'
>> 'aa' == x * 2          # True
>> 'aa' is x * 2          # False
```



# Operatory w instrukcjach warunkowych

## - hierarchia - przykład 6.4

Operator	Opis
or	Logiczne OR (lub)
and	Logiczne AND (i)
not <i>x</i>	Logiczne NOT (nie)
in, not in	Testy przynależności
is, is not	Testy tożsamości
<, <=, >, >=, <>, !=, ==	Porównania

help("|")

help("&")

help("~")

help("^")

**Operatory bitowe**

Zadanie dla  
chętnych:  
opisz działanie  
operatorów na  
przykładach

Dla przejrzystości  
zapytań staraj się  
wstawiać nawiasy!

Jaki będzie wynik poniższego kodu?

```
x, z, y = 1, 2, 1
print(x == z == y) # x == z and z == y -> False
print(x != z != y) # x != z and z != y -> True

# k = x == y and not x == y or not x == y
k = (x == y) and (((not x) == y) or (not x) == y)
print(not k) # True
```



# Instrukcja warunkowa if

- sterowanie wykonaniem kodu

```
x = 2
if x > 5:
    print(x > 5)
    x = 1
print("Koniec")
```

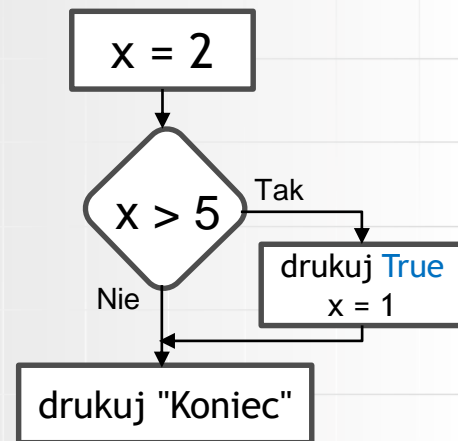
słowo  
kluczowe

dwukropek

warunek

wnętrze instrukcji  
warunkowej (wcięcie)

```
if true_or_not:
    wykonaj_gdy_true
```



- Instrukcja warunkowa rozpoczyna się słowem kluczowym if, następnie występuje warunek i znak :
- Wewnątrz instrukcji if (wydzielonej poprzez wcięcie) znajdują się instrukcje, które zostaną wykonane, gdy warunek będzie spełniony

Przykład 6.5a)

Podaj fragment pseudokodu, w którym zaprezentujesz wyrażenie:

*Jeśli pogoda jest dobra, idę na spacer*

*Po południu jem obiad.*

```
If dobra_pogoda:
    ide_na_spacer()
jem_obiad()
```



# Instrukcja warunkowa if-else

- sterowanie wykonaniem kodu

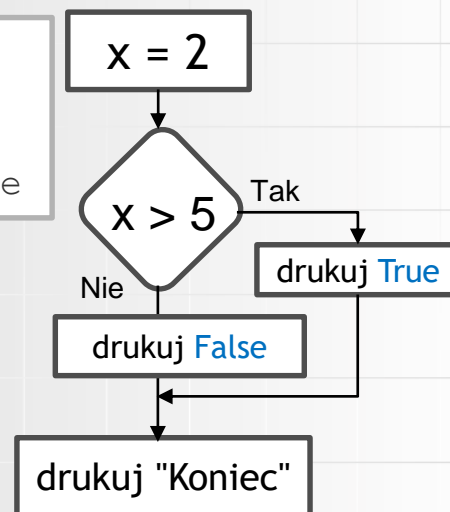
```
x = 2  
  
if x > 5:  
    print(x > 5)  
else:  
    print(x > 5)  
print("Koniec")
```

słowa  
kluczowe  
warunek

wnętrze instrukcji  
warunkowej (wcięcie)

wnętrze instrukcji  
else (wcięcie)

```
if true_or_not:  
    wykonaj_gdy_true  
else:  
    wykonaj_gdy_false
```



- Kontynuacja instrukcji warunkowej rozpoczyna się słowem kluczowym else, a następnie występuje : (**nie ma else bez if**)
- Wewnątrz instrukcji else (wydzielonej poprzez wcięcie) znajdują się instrukcje, które zostaną wykonane, gdy warunek przy if nie będzie spełniony

Przykład 6.5b)

Podaj fragment pseudokodu, w którym zaprezentujesz wyrażenie:

*Jeśli pogoda jest dobra, idę na spacer*

*w przeciwnym razie idę na kręgle.*

*Po południu jem obiad.*

```
If dobra_pogoda:  
    ide_na_spacer()  
else:  
    ide_na_kregle  
jem_obiad()
```



# Instrukcja warunkowa - zagnieżdżanie

- sterowanie wykonaniem kodu

```
x = 2          warunki          słowa
                                kluczowe

if x >= 2:
    print('x >= 2')
    if x > 5:
        print('x > 5')
    else:
        print('x <= 5')
else:
    print('else')
print("Koniec")
```

wnętrze instrukcji

warunkowej (wcięcie)

wnętrze kolejnej  
instrukcji warunkowej  
(wcięcie)

wnętrze instrukcji  
else dla drugiego ifa  
(wcięcie)

wnętrze instrukcji  
else dla pierwszego ifa  
(wcięcie)

```
if true_or_not_1:
    wykonaj_gdy_true_1
    if true_or_not_2:
        wykonaj_gdy_true_2
    else:
        wykonaj_gdy_false_2
else:
    wykonaj_gdy_false_1
```

- Wewnątrz instrukcji if (podobnie else) można umieścić kolejne instrukcje warunkowe, które będą sprawdzane, gdy prawdziwe jest pierwsze wyrażenie – jest to tak zwane zagnieżdżanie instrukcji warunkowych.
- **UWAGA:** opcjonalny else dołączony jest do swojego ifa i musi znajdować się na tym samym poziomie wcięcia



# Instrukcja warunkowa if-elif-else

- sterowanie wykonaniem kodu

warunki

```
x = 2
```

```
if x > 5:
```

```
    print('x > 5')
```

```
elif x > 2:
```

```
    print('x > 2')
```

```
else:
```

```
    print('else')
```

```
print("Koniec")
```

słowa

kluczowe

wnętrze instrukcji

warunkowej (wcięcie)

wnętrze kolejnej

instrukcji warunkowej  
(wcięcie)

wnętrze instrukcji

else (wcięcie)

```
if true_or_not_1:
```

```
    wykonaj_gdy_true_1
```

```
elif true_or_not_2:
```

```
    wykonaj_gdy_true_2
```

```
else:
```

```
    wykonaj_gdy_false
```

- Poprzez słowo kluczowe elif zaznaczamy, że chcemy sprawdzić więcej niż jeden warunek i zatrzymać się gdy pierwszy z nich (w kolejności od góry do dołu) będzie prawdziwy.
- Dodatkowe wykorzystanie else pozwala na wykonanie pewnych instrukcji, gdy wszystkie kolejno postawione warunki były fałszywe (jeśli występuje else to zawsze jest na końcu).
- Instrukcje typu if-elif-else także można zagnieżdżać.





# Instrukcje warunkowe

## - przykład 6.6 a) i b)

Napisz program który pozwoli na pobranie n liczb całkowitych i wyświetli największą z nich dla:

a)  $n = 2$

b)  $n = 3$

```
a = int(input("Podaj liczbę: "))
b = int(input("Podaj liczbę: "))
if a > b:
    print("Największą liczbą jest", a)
else:
    print("Największą liczbą jest", b)
```

```
a = int(input("Podaj liczbę: "))
b = int(input("Podaj liczbę: "))
c = int(input("Podaj liczbę: "))
if a > b and a > c:
    print("Największą liczbą jest", a)
elif b > a and b > c:
    print("Największą liczbą jest", b)
else:
    print("Największą liczbą jest", c)
```



# Instrukcje warunkowe

## - przykład 6.6 b) – mniej porównań

Napisz program który pozwoli na pobranie  $n$  liczb całkowitych i wyświetli największą z nich dla:

a)  $n = 2$

b)  $n = 3$

```
a = int(input("Podaj liczbę: "))
b = int(input("Podaj liczbę: "))
c = int(input("Podaj liczbę: "))
najwieksza = a
if b > najwieksza:
    najwieksza = b
if c > najwieksza:
    najwieksza = c
print("Największą liczbą jest", najwieksza)
```



# Operatory w instrukcjach warunkowych

## - operowanie na napisach

- Porównywanie obiektów zależy od ich typu - liczby są porównywane arytmetycznie a napisy leksykograficznie znak po znaku.

```
'ala' < 'kot' # True  
'Ala' < 'ala' # True
```

- Symbole znakowe są przechowywane w pamięci komputera jako liczby, a najpopularniejszy system kodowania to system ASCII (American Standard Code for Information Interchange):
  - literom od „A” do „Z” odpowiadają liczby od 65 do 90,
  - literom od „a” do „z” odpowiadają liczby od 97 do 122,
  - cyfrom od „0” do „9” odpowiadają liczby od 48 do 57,
  - znakowi spacji odpowiada liczba 32,
  - znakowi nowej linii odpowiada liczba

Kody ASCII nie obejmują polskich znaków diakrytycznych

```
'a' > 'A' # True  
ord('a') > ord('A') # True, bo 97 > 65  
chr(97) > chr(65) # True, bo 97 > 65
```



# Instrukcje warunkowe

- przykład 6.7 – różnice między typami instrukcji

Co jest wynikiem wywołań skryptów?

```
x = 5
if x == 5:
    print(x == 5)
if x > 2:
    print(x > 2)
if x < 5:
    print(x < 5)
else:
    print("else")
```

True  
True  
else

```
x = "11"

if x == 10 + 1:
    print("raz")
elif x == "1" * 2:
    if ord(x[0]) > 1:
        print("dwa")
    elif int(x) < 12:
        print("trzy")
    else:
        print("cztery")
if int(x) == 11:
    print("pięć")
else:
    print("sześć")
```

dwa  
pięć



# Rzeczy do zapamiętania!

1. Wynikami porównań są wartości logiczne: True oznaczająca prawdę (podobnie jak niezerowa liczba całkowita czy niepusta kolekcja lub napis) a False oznaczająca fałsz (tak jak liczba zero czy pusta kolekcja lub napis).
2. Operatory `<`, `>`, `==`, `>=`, `<=` oraz `!=` porównują wartości dwóch obiektów. Obiekty nie muszą być tego samego typu. Jeśli oba są liczbami, następuje ich konwersja do wspólnego typu. W przeciwnym wypadku, obiekty różnych typów są zawsze uznawane za nierówne, a ich porządek musi być ustalony arbitralnie np. przy porównywaniu liczb całkowitych i znaku musi się odbyć za pośrednictwem funkcji `ord` (dla zakodowania znaku na liczbę całkowitą) lub funkcji `chr` (dla zakodowania liczby na znak) – w innym przypadku rzuci błędem.
3. Operatory `in` oraz `not in` dokonują sprawdzenia przynależności. Wyrażenie `x in s` jest prawdziwe, jeśli `x` jest elementem zbioru `s`, a fałszywe w przeciwnym wypadku. Tradycyjnie sprawdzanie przynależności jest związane z typami złożonymi.
4. Operatory `is` oraz `is not` sprawdzają tożsamość obiektu: `x is y` jest prawdziwe wtedy i tylko wtedy, gdy `x` i `y` są tymi samymi obiektami.



# Rzeczy do zapamiętania!

5. Operator not zwraca 1, jeśli jego argumentem jest fałsz, zaś 0 w przeciwnym wypadku.
6. Wyrażenie  $p$  and  $q$  sprawdza czy oba wyrażenia (tzn.  $p$  oraz  $q$ ) są prawdziwe i jeśli tak jest zwraca true.
7. Wyrażenie  $p$  or  $q$  sprawdza czy którekolwiek z wyrażień (tzn.  $p$  oraz  $q$ ) jest prawdziwe i jeśli tak jest zwraca true.
8. Porównania mogą być dowolnie łączone, np.  $x < y \leq z$  jest równoważne zapisowi  $x < y$  and  $y \leq z$ , z wyjątkiem tego, że  $y$  jest wartościowane tylko raz (jednak w obu przypadkach  $z$  nie jest wartościowane w ogóle, jeśli warunek  $x < y$  okaże się być fałszywy). Nie jest to zapis zbyt elegancki i lepiej posługiwać się nawiasami.
9. Porównywanie obiektów zależy od ich typu - liczby są porównywane arytmetycznie a napisy leksykograficznie znak po znaku.
10. Symbole znakowe są przechowywane w pamięci komputera jako liczby, a najpopularniejszy system kodowania to system ASCII.



# Rzeczy do zapamiętania!

11. Jeśli chcesz wykonać jakąś część instrukcji wtedy i tylko wtedy, gdy zostanie spełniony jakiś warunek możesz wykorzystać instrukcje warunkowe:

- Pojedyncze if - do sprawdzenia jednego warunku i odpowiedniej reakcji, gdy będzie on prawdziwy
- Wielokrotne if - do sprawdzenia wielu warunków i odpowiedniej reakcji za każdym razem, gdy którykolwiek z nich będzie prawdziwy.
- Wyrażenia if-else - do reakcji gdy sprawdzany warunek jest prawdziwy oraz możliwości reakcji, gdy jest on nieprawdziwy (po else). Tu znów możliwa jest seria takich wielokrotnych zapytań.
- Wyrażenia if-elif-else - daje możliwość kaskadowego sprawdzania warunków i odpowiedniej reakcji na pierwszy prawdziwy z nich.