Wstęp do programowania
semestr zimowy 2024/2025

Dr Anna Muranova
UWM w Olsztynie

Wykład 2

Python 2. vs Python 3.

- Utworzenie drugiej gałęzi rozwoju 3.x. Początkowe obie gałęzie były rozwijane niezależnie, lecz wsparcie Pythona 2.x zostało zakończone w roku 2020.
- Python 2.x cały czas jest wykorzystywany w niektórych narzędziach, np. w ArcGis Desktop
  `https://support.esri.com/en/technicalarticle/000013224`

Polecenie	Python 2	Python 3
`3/2`	1	1.5
`3//2`	1	1
`3/2.0`	1.5	1.5
`3//2.0`	1.0	1.0

Spróbować Python 2 można tu:
`https://onecompiler.com/python2`

Potęgowanie

```
2**2 #4
2**.5 #1.4142135623730951
2**1/2 #1.0
2**(1/2) #1.4142135623730951
4**(1/2) #2.0
-4**(1/2) #-2.0
(-4)**(1/2) #(1.2246467991473532e-16+2j)
(-4)**(-1/2) #(3.061616997868383e-17-0.5j)
16**(1/4) # 2.0
type(16**(1/4)) #<class 'float'>
16**0.5  #4.0
16**0.25 #2.0
type(16**(0.25)) #<class 'float'>
```

Typ `str`

Kolejną ważną zmienną są string'i czyli napisy
`str` – string, napisy, łańcuchy znaków.
Obecnie odchodzi się od określenia "tablica znaków".
Definiujemy je za pomocą apostrofów ('...') lub cudzysłowów ("..."). Inaczej `Hello World!` nie zadziała:

```
Hello World
"Hello World"
'Hello World'
"napis"
'napis'
"to jest 'napis'"
'i to tez jest "napis"'
'doesn't'
"doesn't"
'doesn\'t'
```

Każdy string musi zaczynać się i kończyć tym samym znakiem – nie ważne czy użyjemy apostrofów ('...') czy cudzysłowów ("...").

Uwaga!

Stringi:

- można używać pojedynczych apostrofów jak i podwójnych cudzysłowów
- ważne, aby stosować wybraną notację konsekwentnie
- jedyny wyjątek to gdy wewnątrz `stringu` chcemy użyć cudzysłów np. `print('Oglądam film "Player One"')`

Co jeszcze możemy zrobić ze stringami?

Ciągi znaków możemy dodawać (inaczej łączyć, inaczej konkatenować) czy mnożyć:

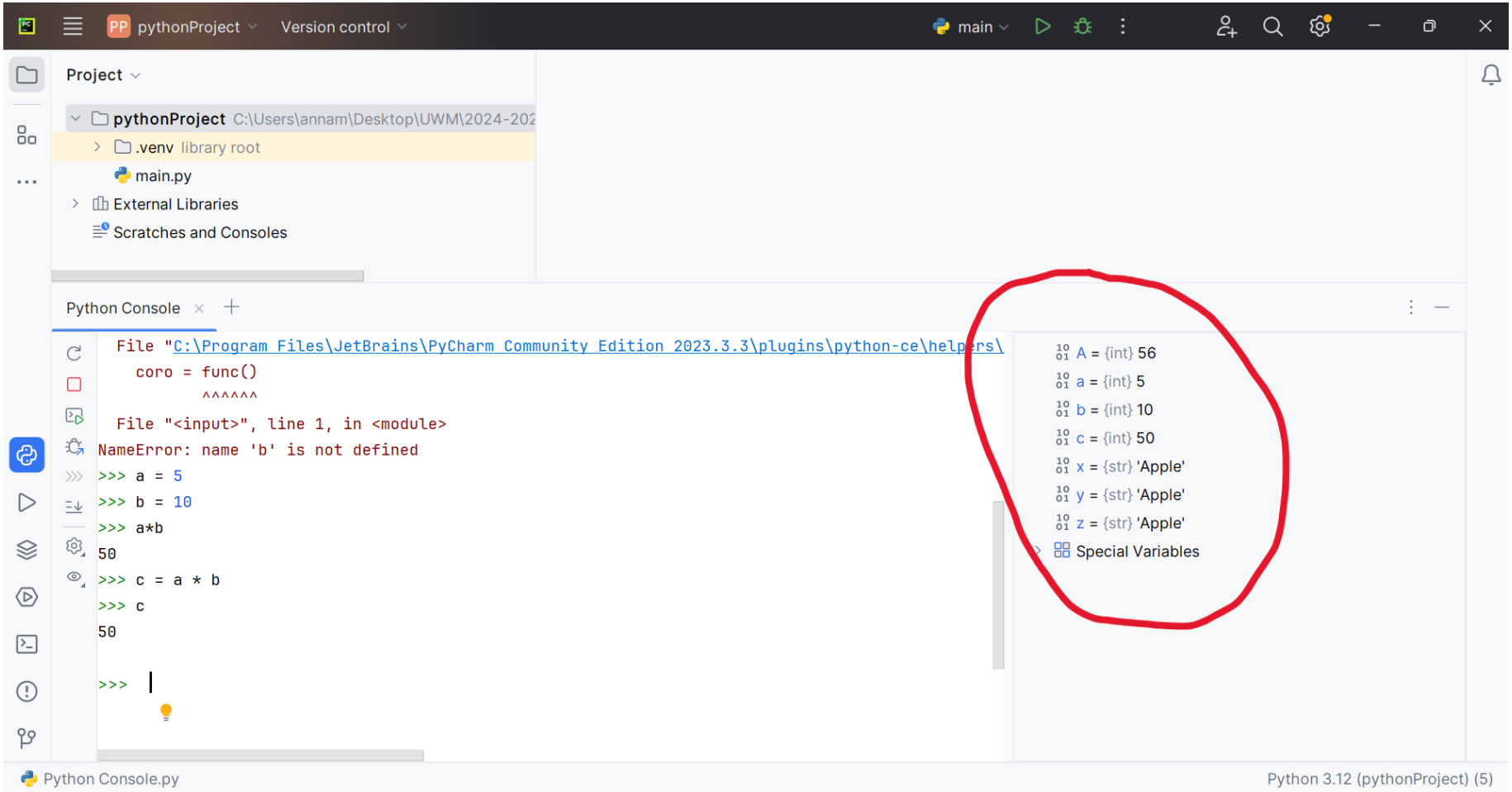
```
'Kto to jest?'+'To Adam'
'Kto to jest?'+'"To Adam" odpowiedziała Magda'
'herbata' * 3
'herbata' + 3
'herbata' + "3"#'herbata3'
'herbata' + str(3)
len('herbata')#7
len('herbata'*3)
```

Przypisywanie wartości do zmiennej

- Zmienna:
  najprościej: przechowuje pewną wartość
- Operator przypisania:
  = przypisuje prawą stronę do lewej (!), często mylony z operatorem logicznym równa się ==

```
x = 5
y = "Piotr"
a = 4.5
A = 56
x, y, z = "Orange", "Banana", "Cherry"
x = y = z = "Apple"
z = 'Cherry'
y#'Apple'
b#NameError: name 'b' is not defined
a = 5
b = 10
a * b
c = a * b
c
```

Przypisywanie wartości do zmiennej



Nazwy zmiennych

Programiści zazwyczaj wybierają dla swoich zmiennych nazwy, które są sensowne i opisują, do czego dana zmienna jest używana.
Nazwy zmiennych mogą być dowolnie długie. Mogą zawierać zarówno litery, jak i cyfry, ale nie mogą zaczynać się od cyfry. Użycie dużych liter jest poprawne, ale lepiej jest zacząć nazwy zmiennych od małej litery. Co prawda w nazwach zmiennych można też używać np. polskich znaków diakrytycznych, ale lepiej trzymać się przyjętej w środowisku programistów konwencji i ich nie używać.
Znak podkreślenia (_) może pojawić się w nazwie. Jest on często używany w nazwach zawierających wiele słów, takich jak `moje_imie` lub `predkosc_jaskolki_bez_obciazenia`. Nazwy zmiennych mogą zaczynać się od znaku podkreślenia, ale generalnie tego unikamy (chyba że piszemy kod biblioteki, która docelowo będzie używana przez inne osoby).
Jeśli nadasz zmiennej niepoprawną nazwę, otrzymasz błąd składniowy (SyntaxError):

```
76 trombones = ' big parade '
more@ = 1000000
class = ' Advanced Theoretical Zymurgy '
```

Python ma zarezerwowanych 35 słów kluczowych

and	del	from	None	True
as	elif	global	nonlocal	try
assert	else	if	not	while
break	except	import	or	with
class	False	in	pass	yield
continue	finally	is	raise	async
def	for	lambda	return	await

Uwaga:

```
None = 1000000
none = 1000000
```

Instrukcja `print()`

`print()` – polecenie wyświetlania.

```
print("Hello World!")
```

Możesz połączyć dwa lub więcej napisów w jednym poleceniu oddzielając je przecinkiem, a spacja między nimi doda się automatycznie:

```
print("Witaj", "co u Ciebie?")
```

`print()` pozwala na wyświetlanie różnych typów, nie tylko napisów.

```
print(3)
print(3, 6, 8)
print("Ania ma:", 3, "lata")
print()
print('doesn\'t')
print('Hello \nworld')
print('Hello' , 'world')
print('Hello' , 'world' , sep=' ')
print('Hello' , 'world', sep='\n')
```

Znaki specjalne

Znak specjalny	Znaczenie
\n	znak nowej linii, dodanie „entera”
\t	dodanie tabulacji ”=8 spacji
\’	apostrof
\"	cudzysłów
\\	ukośnik

```
print("To jest\nnowe")
print("To jest\n\tnowe")
```

Instrukcja `print()`

`print()` – polecenie wyświetlania.

```
print("Hello World!")
```

Możesz połączyć dwa lub więcej napisów w jednym poleceniu oddzielając je przecinkiem, a spacja między nimi doda się automatycznie:

```
print("Witaj", "co u Ciebie?")
```

`print()` pozwala na wyświetlanie różnych typów, nie tylko napisów.

```
print(3)
print(3, 6, 8)
print("Ania ma:", 3, "lata")
print()
print('doesn\'t')
print('Hello \nworld')
print('Hello' , 'world')
print('Hello' , 'world' , sep=' ')
print('Hello' , 'world', sep='\n')
```

Instrukcja `print()`

```
print(5+3)
print(4*5.2)
print(9-7)
print(25%7)
print(4/5)
print(4//5)
print(4/5.0)
print(4//5.0)
print(3**0)
print(0**0)
print(4/0)
```

print() ze zmiennymi

```
a = 5
b = 7
print(a + b)
print(s = 3)
c, d = 2, 4
print(c/d**2)
print(c/(d**2))
print((c/d)**2)
slowo = 'Ala ma'
print(slowo + 'kota')
print(slowo, 'kota')
print(slowo, 'kota', sep= '*')
print(slowo*2)
print(slowo, 3)
print(slowo+3)
```

Przykład zadania

Obliczyć BMI dla wagi – 63, wzrost – 1.60.

Wzór na BMI:

$$BMI = \frac{waga}{wzrost^2}$$

```
wzrost = 1.60
waga = 53
BMI = waga / (wzrost ** 2)
BMI
print("Twoje bmi wynosi:",BMI)
```

co, oczywiście, można zrobić w jednej linii:

```
print("Twoje bmi wynosi:", 53.0 / (1.6 ** 2))
```

Zaawansowane opcje `print()`

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush =False)
```

- `objects` – to co ma być wyświetlone
- `sep` – separator, domyślnie znak spacji
- `end` – co co ma być wyświetlone na końcu, domyślnie znak końca linii
- `file` – określa gdzie mają być `objects` wyświetlone, domyślnie `sys.stdout` (domyślny ekran)
- `flush` – określa czy „wyjście" ma być buforowane przed przekazaniem do `file`, domyślne `False`

Przykłady

```
print(1, 2, 3, 4)
print(1, 2, 3, 4, sep = '*')
print(1, 2, 3, 4, sep = '*', end = '&')
print('x', 'y', 'z', sep='', end='')
print('a', 'b', 'c', sep='', end='')
print('a', 'b', '\n', 'c')
print('sdf', 3456, -2, sep='\t')
```

Skróty dla operacje arytmetycznych na zmiennych

+	+ =	dodawanie
−	− =	odejmowanie
*	* =	mnożenie
/	/ =	dzielenie
**	** =	potęgowanie
//	// =	dzielenie liczb całkowitych lub iloraz przy dzieleniu z resztą
%	% =	reszta

```
a = 3
a*=2
print(a) #6
#print(a*=2)
#b*=a**2
b = 2
b*=a**2
print(b) #72
#a+b*=2
b/=a
print(b)#12.0
#3*=2
```

Typ `bool`

Typ logiczny `bool` to dwie stałe: `True` (dawniej równe 1) i `False` (dawniej równe 0). `True` i `False` pojawiły się w Py2.3. Jednak ogólnie w Pythonie jako fałsz logiczny traktuje się:

- liczbę zero (0, 0.0, 0e0, 0j, itp.),
- stałe `False` i `None` (null),
- puste kolekcje (pusta lista, tuple/krotka, słownik, itp.),
- puste napisy,
- obiekty posiadające metodę `__nonzero__()`, jeśli zwraca ona `False` lub 0.
- obiekty posiadające metodę `__len__()`, jeśli zwraca ona 0

Wszystko inne jest prawdą logiczną

Typ bool

```
print(True)
print(False)
a = True
print(a) # True
print(5 + True) #6
print(True + True) #2
a += 2
print(a) #3
b = False
print(type(b)) #<class 'bool'>
print(type(b + 3)) #<class 'int'>
print(bool(0)) #False
print(bool(5)) #True
print(bool('')) #False
print(bool(' ')) #True
```

Styl PEP8

- wymowa: pi-i-pi-ejt
- standaryzacja kodu używana m.in. przy rozwijaniu nowych funkcjonalności
- używanie daje lepszą organizację i czytelność kod
- pełna wersja `https://www.python.org/dev/peps/pep-0008/`

Znaki odstępu

- we wcięciach stosujemy spacje (a nie tabulatory)
- każdy poziom wcięcia powinien składać się z 4 spacji
- wiersz powinien składać się z maksymalnie 79 znaków

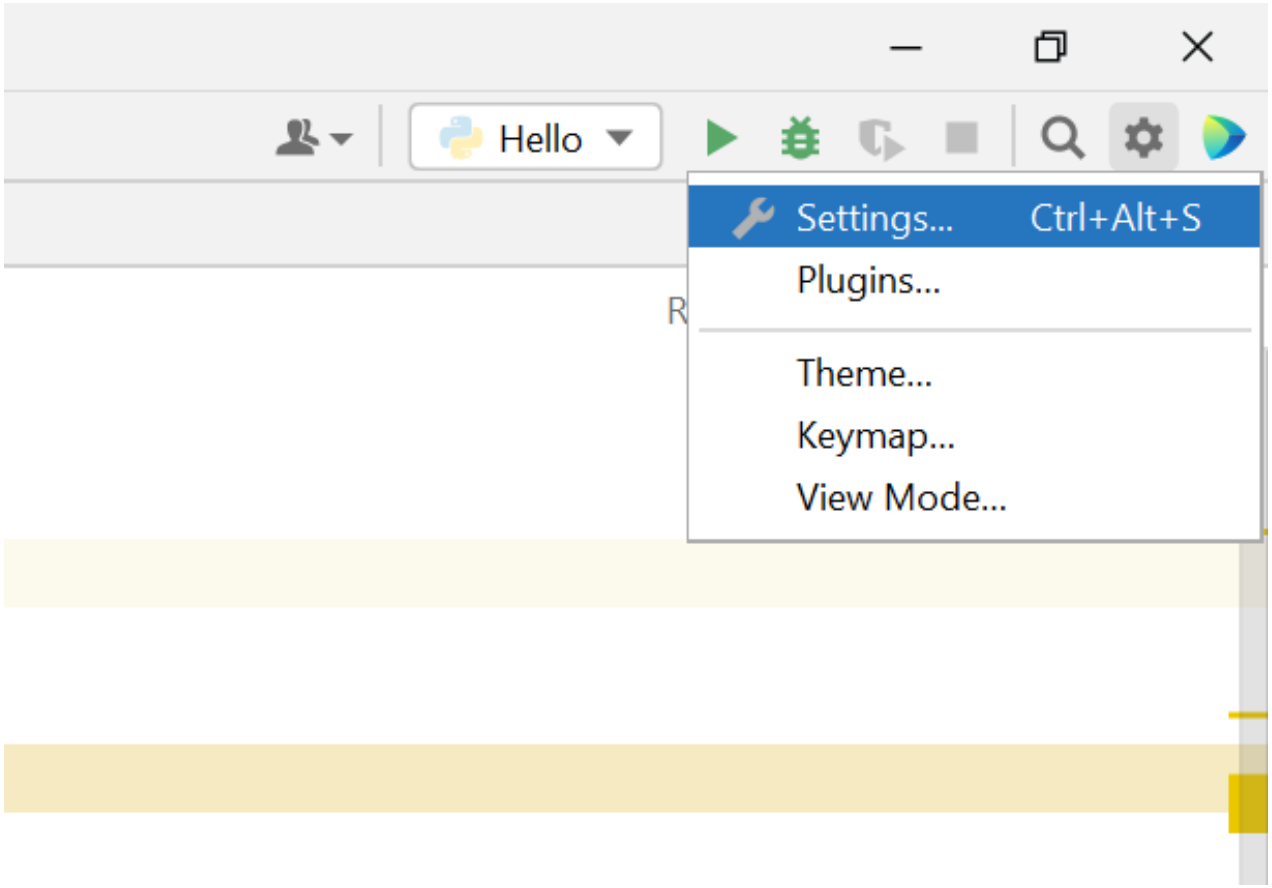
Puste linie:

- dwie linie między funkcjami najwyższego poziomu i między klasami.
- pojedyncza linia między funkcjami w klasie

Kodowanie:

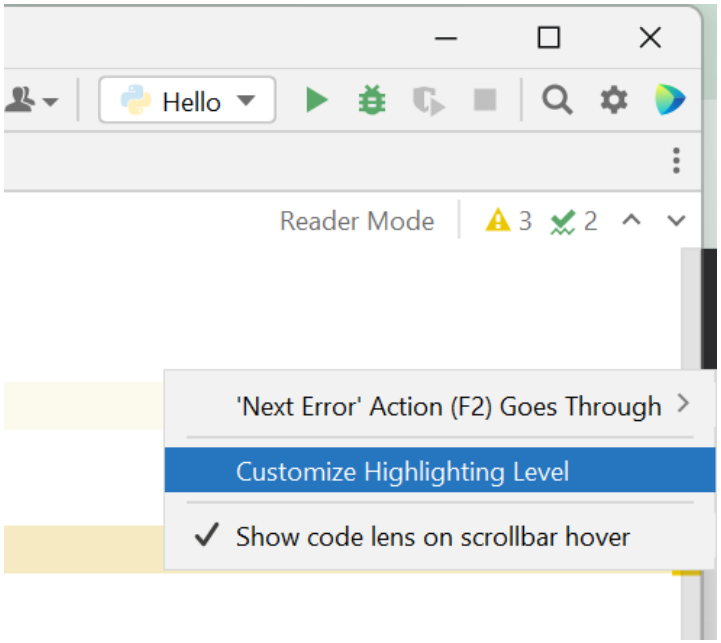
- dla Pythona 3 sugerowane i domyślne to UTF-8.

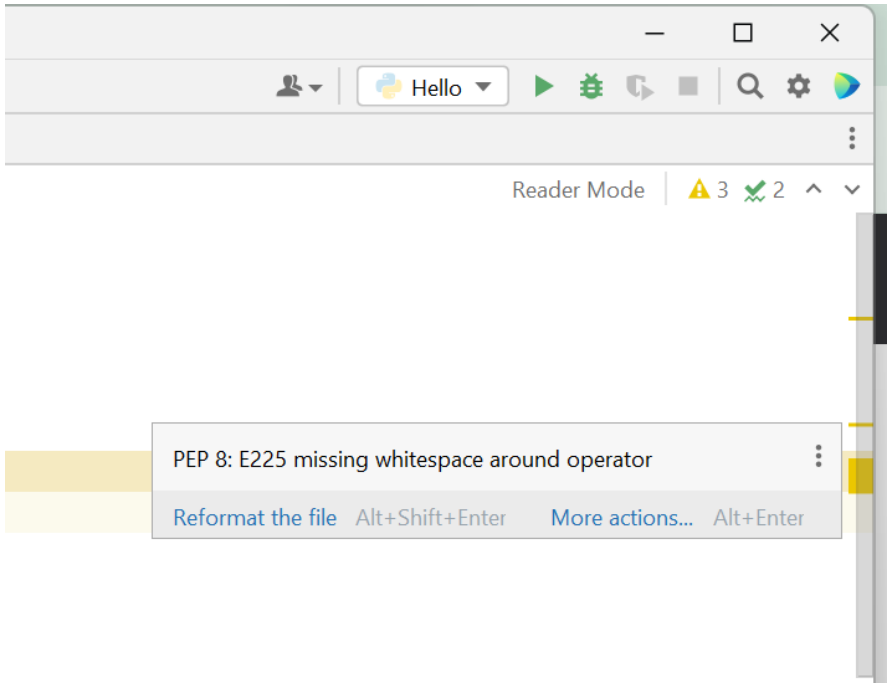
Jak sprawdzić, by kod spełniał standard PEP8?



Settings –> Editor –> Inspections –> Python –> PEP8 . . .

Dodać podpowiedzi:





Przykład

```
var1 = 2
var2 = 3
var1 += 2
var3 = var1
var1 *= 3
var4 = var1/var2
var2 += 2


#wyświetlenie wartości:
print('var2 = ', var2)
```

oznacza komentarz

3 typy instrukcji

Istnieje **tylko** trzy typa instrukcje:

- Instrukcja prosta
- Instrukcja warunkowa
- Pętla

Program z instrukcją prostą

```
a = int(input ("Wprowadź pierwszą liczbę : "))
b = int(input ("Wprowadź drugą liczbę : "))
print(a/b)
```

A jeżeli chcemy sprawdzić , ze $b \neq 0$?

Instrukcja warunkowa

Instrukcja warunkowa – element języka programowania, który pozwala na wykonanie różnych instrukcji w zależności od tego czy zdefiniowane przez programistę wyrażenie logiczne jest prawdziwe, czy fałszywe.

Jeżeli $b \neq 0$, wyświetl a/b.
Inaczej – wyświetl komunikat o błędzie (opcjonalne).

```
if <expr>:
    <statement(s)>
```

Jeżeli $b \neq 0$, wyświetl a/b, inaczej....

```
if <expr>:
    <statement(s)>
else:
    <statement(s)>
```

Warunek musi być typu `bool`.

Typ `bool`

Typ logiczny `bool` to dwie stałe: `True` (dawniej równe 1) i `False` (dawniej równe 0). `True` i `False` pojawiły się w Py2.3. Jednak ogólnie w Pythonie jako fałsz logiczny traktuje się:

- liczbę zero (0, 0.0, 0e0, 0j, itp.),
- stałe `False` i `None` (null),
- puste kolekcje (pusta lista, tuple/krotka, słownik, itp.),
- puste napisy,
- obiekty posiadające metodę `__nonzero__()`, jeśli zwraca ona `False` lub 0.
- obiekty posiadające metodę `__len__()`, jeśli zwraca ona 0

Wszystko inne jest prawdą logiczną

Operatory porównania

Wynikiem działania operatora porównania jest wartość typu `bool`

Symbol	Znaczenie	Przykład
>	Większe niż	`x > y`
<	Mniejsze niż	`x < y`
==	Równe	`x == y`
!=	Nie równa się	`x != y`
>=	Większe lub równe	`x >= y`
<=	Mniejsze lub równe	`x <= y`

if...

```
a = int(input ("Wprowadź pierwszą liczbę : "))
b = int(input ("Wprowadź drugą liczbę : "))
if b != 0:
    print(a,":",b, "=", a/b)
```

Bez wcięć będzie błąd

```
a = int(input ("Wprowadź pierwszą liczbę : "))
b = int(input ("Wprowadź drugą liczbę : "))
if b != 0:
print(a,":",b, "=", a/b)
```

if... else...

```
a = int(input ("Wprowadź pierwszą liczbę : "))
b = int(input ("Wprowadź drugą liczbę : "))
if b != 0:
    print(a,":",b, "=", a/b)
else:
    print("Błąd: dzielnik nie może być 0")
```

if... else...

Minimum dwóch liczb:

```
a = int(input ("Wprowadź pierwszą liczbę : "))
b = int(input ("Wprowadź drugą liczbę : "))
if b > a:
  mmin  = a
else:
  mmin = b
print(mmin)
```

elif

```
a = int(input ("Wprowadź liczbę : "))
if a > 0:
  print("a is positive")
elif a < 0:
  print("a is negative")
else:
  print("a is 0")
```

Można używać bez else:

```
a = int(input ("Wprowadź liczbę : "))
if a > 0:
  print("a is positive")
elif a < 0:
  print("a is negative")
```

Przykład

```
day = int(input("Enter a number from 1 to 7: "))

if day == 1:
    print(day, "is Sunday")
elif day == 2:
    print(day, "is Monday")
elif day == 3:
    print(day, "is Tuesday")
elif day == 4:
    print(day, "is Wednesday")
elif day == 5:
    print(day, "is Thursday")
elif day == 6:
    print(day, "is Friday")
elif day == 7:
    print(day, "is Saturday")
else:
    print("Wrong input! Please enter a number from 1 to 7.")
```

Krotka pisownia warunku

```
a = int(input ("Wprowadź liczbę : "))
if a > 0: print("a is positive")
```

```
a = int(input ("Wprowadź liczbę : "))
print("a is positive") if a > 0 else print("a is not positive")
```

```
a = int(input ("Wprowadź liczbę :  "))
print(‘‘a is positive") if a > 0 else print(‘‘a is zero") if a == 0 else print(‘‘a is negative")
```

Warunki

A jeżeli chcemy sprawdzić, ze *a* jest pomiędzy 0 a 100?

```
a = int(input ("Wprowadź liczbę : "))
if a > 0:
  if a < 100: print("Tak")
  else: print("Nie")
else:
  print("Nie")
```

```
a = int(input ("Wprowadź liczbę : "))
if (a > 0 and a < 100):
    print("Tak")
else:
  print("Nie")
```

```
a = int(input ("Wprowadź liczbę : "))
if 0 < a < 100:
    print("Tak")
else:
  print("Nie")
```

Operatory logiczne

Wynikiem działania operatora logicznego jest wartość typu `bool`

Symbol	Znaczenie	Przykład
`and`	i	`x and y`
`or`	lub	`x or y`
`not`	negacja	`x not y`

Operatory logiczne: przykłady

```
a = int(input ("Wprowadź liczbę : "))
if (a > 0 and a < 100):
    print("Tak")
else:
  print("Nie")


a = int(input ("Wprowadź liczbę : "))
if (a <= 0 or a >= 100):
    print("Nie")
else:
  print("Tak")


a = int(input ("Wprowadź pierwszą liczbę : "))
b = int(input ("Wprowadź drugą liczbę : "))
if not b == 0:
    print(a,":",b, "=", a/b)
else:
    print("Błąd: dzielnik nie może być 0")
```

Operatory logiczne: jeszcze przykład

```
x , y = 0 , 5

if x < y:
    print('yes 1')
if y < x:
    print('yes 2')
if x:
    print('yes 3')
if y:
    print('yes 4')
if x or y:
    print('yes 5')
if x and y:
    print('yes 6')
```

```
yes 1
yes 4
yes 5
```

Włożone if ...

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

Above ten,
and also above 20!

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
else print ("Below 10")
```

Włożone if ...

Porównaj (i = 22, -2, 21, -11):

```
if i > 0:
    print("Liczba jest dodatnia")
    if i % 2 == 0:
        print("Liczba jest dodatkowo parzysta")
```

oraz

```
if i > 0:
    print("Liczba jest dodatnia")
if i % 2 == 0:
    print("Liczba jest parzysta")
```

oraz

```
if i % 2 == 0:
    print("Liczba jest parzysta")
    if i > 0:
        print("Liczba jest dodatkowo dodania")
```

pass

Jeżeli w `if` nic nie wykonujemy to musimy tam wpisać pass

```
a = 200
b = 20

if b > a:
  pass
else:
    print("Hello")
```