1 Liczby i operatory:

```
2 # liczba całkowita
2

2.33 # liczba typu float
2.33

type(2) # funkcja wbudowana sprawdzająca typ
int

type(2.33)
float
```

```
2 + 3 # dodawanie, odejmowanie
5

2 - 3
-1

2 * 2 # mnożenie / dzielenie
4

5/2
2.5
```

```
5 // 2 # dzielenie 'flor division'
2

5 % 2 # dzielenie: reszta z dzielenia
1

2 ** 3 # potęgowanie
8

9 ** 0.5
3.0
```


```
5 // 2 # dzielenie 'flor division'
2

5 % 2 # dzielenie: reszta z dzielenia
1

2 ** 3 # potęgowanie
8

9 ** 0.5
3.0
```


2 Stringi

```
'abc xxxx' # pojedynczy cudzysłów
'abc xxxx'

"abc xxxx" # podwójny cudzysłów
'abc xxxx'

"abc 'zzz' xxxx" # cytat wewnątrz
"abc 'zzz' xxxx"

'abc "zzz" xxx' # cytat wewnątrz
'abc "zzz" xxx'
```

```
# potrójny cudzysłów - tekst wieloliniowy

'''abc
xxx
   zzzz ccccc'''
'abc\nxxx\n   zzzz ccccc'

"""abc
xxx
   zzzz ccccc"""
'abc\nxxx\n   zzzz ccccc'
```

```
In [63]: # \n znak nowej linii i funkcja print()

In [64]: print('aaa\nbbbb')
aaa
bbbb

In [65]: print('''abc
    ...: xxx
    ...:    zzzz ccccc''')
abc
xxx
   zzzz ccccc
```

3 Zmienne

```
x = 3 # przypianie wartości

x
3

type(x) # sprawdzenie typu
int

x = 2.3 ** 3 # nowe przypisanie wartości

x
12.166999999999998

type(x) # sprawdzenie typu
float
```

```
# różne nazwy zmiennych

xx = 100

imie = 'Adam'

a01 = 'xxx lll'

a02 = 20

imie_nazwisko = 'A. Nowak'
```

4 Użycie zmiennych

```
In [90]: # przypisywanie wartości

In [91]: x, y, imie = '21', 21, 'Adam'

In [92]: # print() - wyświetlenie w jednej linii

In [93]: print(x, y, imie)
21 21 Adam

In [94]: # sprawdzenie typu

In [95]: print(type(x), type(y), type(imie))
<class 'str'> <class 'int'> <class 'str'>
```

```
In [96]: # operatory '+', '-', '*', '/'

In [97]: x,y = 9, 5

In [98]: print(x+y, x-y, x*y, x/y)
14 4 45 1.8

In [99]: # przypisanie wyniku do zmiennej

In [100]: z = x*y

In [101]: z
Out[101]: 45

In [102]: print(z, (z+x)/y)
45 10.8
```

5 Działania na stringach

```
# dodawanie i mnożenie

x = 'a'

x + x
'aa'

y = x + x + x

y
'aaa'

y = 5 * x

y
'aaaaa'
```

```
In [117]: # dodawanie i mnożenie

In [118]: x,y = 'aaa', 'zzzzz'

In [119]: x + y
Out[119]: 'aaazzzzz'

In [120]: x * y
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-120-e32109319f52> in <module>
----> 1 x * y

TypeError: can't multiply sequence by non-int of type 'str'
```

```
# indeksowanie / wybieranie

x = '0123456789'

x[0]  # pierwszy element
'0'

x[8]  # ósmy element
'8'

x[-1] # ostatni, licząc od tyłu
'9'
```

```
# wycinki: start, stop, step

x
'0123456789'

x[1:] # od indeksu '1' do końca
'123456789'

x[:-1] # od indeksu '0' do przedostatniego
'012345678'

x[3:6] # wszystkie z zakresu (bez liczby 6)
'345'

x[:6:2] # co drugi z zakresu od '0' do '6'
'024'

x[::3] # co trzeci z całego zakresu
'0369'

x[2::2] # co drugi od '2' do końca
'2468'
```

6 Formatowanie stringów - porównanie 3 metod

Metody:

- *'%s' % (obj)*: Python 2, 3 - stara metoda
- *str.format()*: Python 3
- *f'{}'* (f-string): Python od wersji 3.6

1. Proste wstawianie danych jako 'string':

```
# proste wstawienie stringa

ss,ii,ff = 'abc', 99, 3.12345

'%s' % ss  # stary: s - konwersja na string
'abc'

'{}'.format(ss) # str.format()
'abc'

f'{ss}'   # f-string
'abc'
```

```
'%s' % ii  # stary: s - konwersja na string
'99'

'{}'.format(ii) # str.format()
'99'

f'{ii}'   # f-string
'99'

'%s' % ff
'3.12345'

'{}'.format(ff)
'3.12345'

f'{ff}'
'3.12345'
```

2. Wstawianie liczb:

```
# konwersja liczb: d -int, f - float

'int: %d,  float: %f' % (ii,ff)  # stary
'int: 99,  float: 3.123450'

'int: {:d},   float: {:f}'.format(ii,ff)
'int: 99,   float: 3.123450'

f'int: {ii:d},   float: {ff:f}'
'int: 99,   float: 3.123450'
```

```
ii,ff = 9999, 10000/3

# określenie precyzji wyświetlania liczb

'float: %f,  float: %f' % (ii,ff)  # domyślna precyzja
'float: 9999.000000,  float: 3333.333333'

'float: %.1f,  float: %.3f' % (ii,ff)  # ii: 1 cyfra, ff: 3 cyfry
'float: 9999.0,  float: 3333.333'

'int: {:.1f},   float: {:f}'.format(ii,ff) # str.format()
'int: 9999.0,   float: 3333.333333'

f'int: {ii:.1f},   float: {ff:.3f}'
'int: 9999.0,   float: 3333.333'
```

3. Wstawianie wg. indeksów, nazw, kluczy

```
# wstawianie wg pozycji argumentu - niedostępne w '%s % ()' i f-string

'ff: {1:.1f}, ii: {0:d},  ii: {0:.1f}, ff: {1:.1f}'.format(ii,ff) # str.format()
'ff: 3333.3, ii: 9999,  ii: 9999.0, ff: 3333.3'

# wstawienie wg nazwy argumentu:

'ff: %(ff).1f,  ii: %(ii)d, ii: %(ii).1f,  ff: %(ff).1f'%({'ii':ii,'ff':ff}) # stary sposób
'ff: 3333.3,  ii: 9999, ii: 9999.0,  ff: 3333.3'

'ff: {ff:.1f}, ii: {ii:d},  ii: {ii:.1f}, ff: {ff:.1f}'.format(**{'ii':ii,'ff':ff})  # str.format()
'ff: 3333.3, ii: 9999,  ii: 9999.0, ff: 3333.3'

f'ff: {ff:.1f}, ii: {ii:d}, ii: {ii:.1f},  ff: {ff:.1f}'   # f-string
'ff: 3333.3, ii: 9999, ii: 9999.0,  ff: 3333.3'
```

```
# wstwianie wg kluczy

'%(k3).1f %(k1)s %(k2)d' % s   # stary styl
'333.7 abc 999'

'{k3:.1f} {k1} %{k2:d}'.format(**s)  # str.format()
'333.7 abc %999'

f"{s['k3']:.1f} {s['k1']} {s['k2']:d}"  # f-string
'333.7 abc 999'
```

4. Wyrównywanie tekstu

```
# wyrównywanie: lewo znak '-', prawo domyślnie, szerokości: 10, 15, 12

'%(k3)10.1f %(k1)15s %(k2)12d' % s   # stary: domyślne wyrównanie
'     333.7             abc          999'

'%(k3)-10.1f %(k1)-15s %(k2)-12d' % s   # stary: do lewej
'333.7      abc             999          '

'%(k3)-10.1f %(k1)-15s %(k2)12d' % s   # stary: lewa, lewa, prawa
'333.7      abc                        999'

# str.format() - znaki wyrównania:< lewo, ^środek, >prawo

'{k3:10.1f} {k1:15s} {k2:12d}'.format(**s)  # domyślnie
'     333.7 abc                        999'

'{k3:<10.1f} {k1:<15s} {k2:<12d}'.format(**s)  # lewo
'333.7      abc             999          '

'{k3:^10.1f} {k1:^15s} {k2:^12d}'.format(**s)  # środek
'  333.7          abc           999     '

'{k3:<10.1f} {k1:^15s} {k2:>12d}'.format(**s)  # < ^ >
'333.7            abc                 999'

# f-string: tak jak str.format()

f"{s['k3']:10.1f} {s['k1']:15} {s['k2']:12d}" # domyślnie
'     333.7 abc                        999'

f"{s['k3']:<10.1f} {s['k1']:^15} {s['k2']:>12d}" # < ^ >
'333.7            abc                 999'
```

5. Wypełnienie wyrównania

```
In [244]: # wypełnienie wyrównania różnymi znakami np. _,0,.

In [245]: '{k3:_<10.1f} {k1:0^15s} {k2:.>12d}'.format(**s)  # < ^ >
Out[245]: '333.7_____ 000000abc000000 .........999'

In [246]: '{k3:_<10.1f} {k1:0^15s} {k2:.>12d}'.format(**s)  # < ^ >
Out[246]: '333.7_____ 000000abc000000 .........999'

In [247]: for i in range(1,11,2): print(f'{i:0>2d}\t{i**2:*>10}\t{i*i/3:.^12.2f}')
01      *********1      ....0.33....
03      *********9      ....3.00....
05      ********25      ....8.33....
07      ********49      ...16.33....
09      ********81      ...27.00....
```

6. F-stringi - obliczenia:

```
# tylko f-stringi: operacje na zmiennych

a, b = 3, 7

f'{a/b:^25},{a/b:^10.3f}'
'   0.42857142857142855   ,  0.429   '

l = [i for i in range(8)]

l
[0, 1, 2, 3, 4, 5, 6, 7]

f'sum: {sum(l)},\tsr: {sum(l)/len(l)}'
'sum: 28,\tsr: 3.5'

f'sum: {sum(l)},{"sr":>6}: {sum(l)/len(l)}'
'sum: 28,    sr: 3.5'

l = [str(i) for i in l]

l
['0', '1', '2', '3', '4', '5', '6', '7']

print(f'\t{l[::2]}\n\t{l}')
['0', '2', '4', '6']
['0', '1', '2', '3', '4', '5', '6', '7']

f'{"*".join(l)}'
'0*1*2*3*4*5*6*7'
```