Liczby:
Interpreter działa jak prosty kalkulator: można wpisać wyrażenie do niego, a on wypisze jego wartość. Składnia wyrażenia jest prosta: operator +, -, * i / działają jak w wielu innych językach programowania

```
>>> 2+2
4
>>> # To jest komentarz
... 2+2
4
>>> 2+2 # to też komentarz w tej samej linii co kod
4
>>> (50-5*6)/4
5
>>> # Dzielenie całkowite zwraca liczbę zaokrągloną w dół
... 7//3
2
>>> 7//-3
-3
```

Listy:

```
>>> a = ['ziemia', 'trawa', 100, 1234]
>>> a
['ziemia', 'trawa', 100, 1234]
>>> a[0]
'ziemia'
>>> a[3]
1234
>>> a[-2]
100
>>> a[1:-1]
['trawa', 100]
>>> a[:2] + ['woda', 2*2]
['ziemia', 'trawa', 'woda', 4]
>>> 3*a[:3] + ['Acha!']
['ziemia', 'trawa', 100, 'ziemia', 'trawa', 100, 'ziemia',
'trawa', 100, 'Acha!']
>>> a[2] = a[2] + 23
>>> a
['ziemia', 'trawa', 123, 1234]
```

```
if wyrażenie_warunkowe:
    blok kodu 1
elif:
    blok kodu 2
else:
    blok kodu 3

for i in range(3):
    blok kodu

values = [3, 5, 2, 4, 1]
for value in values:
    if value == 4:
        print("Znalazłem czwórkę, kończę przeszukiwanie !")
        break
    print(value)
```

Pętla `WHILE`

```
while wyrażenie warunkowe:
    blok kodu


kontynuuj = True
while kontynuuj:


while True:
```

Własne funkcje:

Funkcja to podprogram, zwykle zawierający parametry, który może coś zwrócić:

```
def dzielenie(dzielna, dzielnik):
    if(dzielnik == 0):
        return # zakoncz funkcje nic nie zwracajac
    else:
        return dzielna / dzielnik

x=dzielenie(24,3)
print(x)
```

Słowniki.

Słownik jako nieuporządkowany zbiór par `klucz:wartość`, z założeniem, że klucze są unikalne (w jednym słowniku). Kluczem może być dowolny ciąg znaków lub liczba. Para nawiasów klamrowych tworzy pusty słownik: `{}`. Umieszczenie listy par klucz:wartość, oddzielonych przecinkami w tych nawiasach dodaje początkowe pary `klucz:wartość` do słownika:

```
>>> tel = {'Adam': 4098, 'Ola': 4139}
>>> tel['Ala'] = 4127
>>> tel
{'Ola': 4139, 'Ala': 4127, 'Adam': 4098}
```

Podając klucz jako indeks, możemy uzyskać przechowywaną wartość:

```
>>> tel['Adam']
4098
>>> del tel['Ola']
>>> tel['Jan'] = 4127
>>> tel
{'Ala': 4127, 'Jan': 4127, 'Adam': 4098}
```

Metody `keys` oraz `values` obiektu słownika pozwalają wyświetlić wszystkie klucze oraz wartości przechowywane w słowniku:

```
>>> tel.keys()
[dict_keys(['Adam', 'Ala', 'Jan'])
>>> tel.values()
dict_values([4098, 4127, 4127])
```

Metoda obiektu słownika **`keys()`** zwraca listę wszystkich kluczy używanych w słowniku, w porządku losowym (aby posortować listę kluczy, stosujemy metodę **`sorted()`** na tej liście).

```
>>> sorted(tel)
['Adam', 'Ala', 'Jan']
>>> sorted(tel.keys())
['Adam', 'Ala', 'Jan']
>>> sorted(tel.values())
[4098, 4127, 4127]
```

Do sprawdzenia obecności klucza w słowniku służy metoda **`has_key()`**.

```
>>> tel.has_key('Ala')
1
```

Podobnie jak w przypadku list, metoda `len()` zwraca liczbę par klucz-wartość w danym słowniku:

```
>>> len(tel)
4
```

Czytanie z plików i pisanie do plików:
`open()` zwraca obiekt pliku i powszechnie używana jest z dwoma argumentami:
`open(nazwa_pliku, tryb)`
tryb:
`'r'` - plik będzie tylko czytany,
`'w'` - będzie odbywało się wyłącznie pisanie do pliku (istniejący plik o podanej nazwie zostanie wyczyszczony),
`'a'` - otwiera plik, do którego można dodawać dane dopisane na jego koniec,
`'r+'` - otwiera plik zarówno do czytania jak i do pisania.
Argument `tryb` jest opcjonalny, przy jego braku plik zostanie otwarty w trybie `'r'`.

Odczytywanie treści z pliku:

- najprostszy sposób:

```
f = open('test.txt', 'r')
tekst=f.read()
print(tekst)
```

- zastosowanie `with ...as`

```
with open('test.txt', 'r') as f:
    for linia in f:   # odczytywanie linia po linii
        print(linia, end="")  # aby pozbyć się dodatkowych
                                odstępów
```

Pisanie do pliku

```
f = open('test.txt', 'w')
f.write("tekst wpisany")
f.close()
```

W przypadku gdy mamy listę (lub inną sekwencję) napisów to wszystkie możemy zapisać do pliku poprzez metodę `writelines`:

```
lista = ["aaa ", "bbb ", "cc "]
f = open('test.txt', 'w')
f.writelines(lista)
f.close()
```

- zastosowanie `with ...as`

```
with open('test.txt', 'w') as f:
    f.write("Zapisujemy do pliku\n")
```

Tak wygląda realizacja w terminalu:
Zawartość otwartego pliku możemy wczytać od razu w całości, bądź czytać go linia po linii:

```
>>> f = open('test.txt', "r")
>>> f.read()
'Pierwsza linia\nDruga linia\nTrzecia linia\n'
```

Możemy przeczytać konkretną liczbę znaków podaną przez nas jako parametr metody:

```
>>> f = open('test.txt', "r")
>>> f.read(8)
'Pierwsza'
```

Po odczytaniu zawartości pliku zostaje ona "wykasowana" ze zmiennej plikowej:

```
>>> f = open('test.txt', 'r')
>>> f.read(8)
'Pierwsza'
>>> f.read()
' linianDruga linia\nTrzecia linia\n'
```

po odczytaniu całego pliku następne wywołanie metody `read` zwróci pusty napis:

```
>>> f = open('test.txt', 'r')
>>> tekst = f.read()
>>> f.read()
''
```

plik został wcześniej odczytany i zapisany pod `tekst`

```
>>> print(tekst)
Pierwsza linia
Druga linia
Trzecia linia
```

Odczytanie pojedyczej linii z pliku:

```
>>> f = open('test.txt', 'r')
>>> print(f.readline())
>>> f.readline()
'Pierwsza linia\n'
```

odczytana została cała pierwsza linia pliku (wraz ze znakiem końca linii).

Ponowne użycie metody pozwoli na wczytanie następnej linii:

```
>>> f.readline()
'Druga linia\n'
```

Możemy wczytać wszystkie linie pliku do listy metodą `readlines`.

```
>>> f = open('test.txt', 'r')
>>> lines = f.readlines()
>>> lines
['Pierwsza linia\n', 'Druga linia\n', 'Trzecia linia\n']
```

Kodowanie

```
>>> f = open('test.txt', 'r', encoding="utf-8")
>>> f.read()
'żółć'
>>> f = open('test.txt', 'w', encoding='cp1250')
```

Zmiana koloru diody oraz uśpienie układu:

```
import pycom
import time

pycom.heartbeat(False)

for cycles in range(3): # stop after 3 cycles
    pycom.rgbled(0xFF0000)  # Red
    time.sleep(5)
    pycom.rgbled(0x00FF00)  # Green
    time.sleep(2)
    pycom.rgbled(0x0000FF)  # Blue
    time.sleep(1)
```

Uśpienie układu i wybudzenie z wykorzystaniem akcelerometru:

```
from pysense import Pysense
from LIS2HH12 import LIS2HH12
import time

print("Powod uspienia: " + str(py.get_wake_reason()))

# włączanie aktywności, a także przerwania nieaktywności
py.setup_int_wake_up(True, True)

# ustawia próg przyspieszenia na 2000 mG (2G), a minimalny czas
trwania na 200 ms
li.enable_activity_interrupt(2000, 200)

# Uśpienie maksymalnie na 10 sekund, jeśli nie nastąpi przerwanie
akcelerometrem
py.setup_sleep(10)
py.go_to_sleep()
```

Formatowanie łańcucha znaków:

```
print("Cisnienie: " + str(mpp.pressure())+ '\n')
```

lub

```
print("Cisnienie: " + str(mpp.pressure()))
print(' ')
```

```
napis1 = 'zielony'
napis2 = 'szorstka'

tekst = "Kolor: " + napis1 + ", faktura: " + napis2
lub

tekst = "Kolor: %s, faktura: %s” % (napis1, napis2)
lub

tekst = "Kolor: {}, faktura: {}”.format(napis1, napis2)
lub

tekst = "Kolor: {1}, faktura: { 0 }”.format(napis2, napis1)


print( tekst )
Kolor: zielony, faktura: szorstka
```