

### Typy proste

**integer, float, boolean, string, bytes**

```
int 783 0 -192 0b010 0o642 0xf3
l. całkowita zero binarny ósemkowy
float 9.23 0.0 -1.7e-6
l. zmiennoprzecinkowa
bool True False
typ logiczny
str "One\nTwo"
łańcuch znaków Przejście do nowej linii
"'\n'" Wyświetlenie znaku '
"'\t'" Wyświetlenie tabulacji
bytes b"toto\xfe\775"
bajt szesnastkowy ósemkowy niezmiennie
```

### Rodzaje kontenerów

- uporządkowane sekwencje, szybki dostęp do indeksów, powtarzalne wartości
- list [1,5,9] ["x",11,8.9] ["mot"]
- tuple (1,5,9) 11,"y",7.4 ("mot",)
- Wartości niemodyfikowalne (niezmienne) wyrażenie z tylko przecinkami → tuple
- str bytes (uporządkowane sekwencje znaków / bajtów)
- kontenery na klucze, brak kolejności a priori, szybki dostęp do kluczy, każdy klucz jest unikalny
- dictionary dict {"key": "value"} dict(a=3,b=4,k="v")
- (pary klucz/wartość) {1: "one", 3: "three", 2: "two", 3.14: "pi"}
- collection set {"key1", "key2"} {1,9,3,0} set
- klucze = wartości, które można mieszać (typy podstawowe, niezmiennie, itd) frozenset niezmienny zestaw pusty

### Identyfikatory

dla nazw zmiennych, funkcji, modułów, klas ...

a...zA...Z po którym mogą być a...zA...Z\_0...9

- znaki diakrytyczne dozwolone, ale należy ich unikać
- zabronione słowa kluczowe w języku
- rozróżnianie małych/DUŻYCH liter

© a toto x7 y\_max BigOne  
 © 8y and for

### Konwersje typów

type (expression)

określenie podstawy liczby całkowitej w drugim parametrze

określenie części dziesiętnej

zaokrąglenie do 1 miejsca po przecinku (0 liczba dziesiętna → liczba całkowita)

bool(x) False dla zera x, pusty kontener x, None lub False x; True dla innych x

str(x) → "..." reprezentuje łańcuch x do wyświetlenia (por. formatowanie z tyłu)

chr(64) → '@' ord('@') → 64 kod ↔ znak

repr(x) → "..." dosłowny ciąg reprezentujący x

bytes([72,9,64]) → b'H\t@'

list("abc") → ['a','b','c']

dict([(3,"three"),(1,"one")]) → {1:'one',3:'three'}

set(["one","two"]) → {'one','two'}

separator str i sekwencja str → złożony str

':'.join(['toto','12','pswd']) → 'toto:12:pswd'

str wycięcie białych znaków → list utworzona z str

"words with spaces".split() → ['words','with','spaces']

str rozdzielenie na separatorze str → list utworzona z str

"1,4,8,2".split(",") → ['1','4','8','2']

sekwencja jednego typu → list innego typu

[int(x) for x in ('1','29','-3')] → [1,29,-3]

### Przypisania do zmiennych

zadanie ↔ wiązanie nazwy z wartością

- ocena wartości wyrażenia po prawej stronie
- przypisanie w kolejności z nazwami po lewej stronie

x=1.2+8+sin(y)

a=b=c=0 przypisanie do tej samej wartości

y,z,r=9.2,-7.6,0 równoczesne przypisania

a,b=b,a zamiana wartości

a,\*b=seq sekwencja rozpakowywania w \*a,b=seq pozycji i w listach

x+=3 inkrementacja ↔ x=x+3

x-=2 dekrementacja ↔ x=x-2

x=None «niezdefiniowany» stała wartość

del x usuwa nazwę x

### Indeksowanie kontenerów sekwencji

dla list, krotek, łańcuchów znaków, bajtów...

indeks ujemny	-5	-4	-3	-2	-1	
indeks dodatni		0	1	2	3	4
		10	20	30	40	50
wycinek dodatni	0	1	2	3	4	5
wycinek ujemny	-5	-4	-3	-2	-1	

Liczba elementów len(lst) → 5

index od 0 (tutaj od 0 do 4)

Indywidualny dostęp do elementów przez lst[index]

lst[0] → 10 ⇒ pierwszy lst[1] → 20

lst[-1] → 50 ⇒ ostatni lst[-2] → 40

Na modyfikowalnych sekwencjach (list), usuwanie za pomocą del lst[3] i według przypisania lst[4]=25

Dostęp do sekwencji podrzędnych za pośrednictwem lst[start slice: end slice: step]

lst[:-1] → [10,20,30,40] lst[:: -1] → [50,40,30,20,10] lst[1:3] → [20,30] lst[:3] → [10,20,30]

lst[1:-1] → [20,30,40] lst[:: -2] → [50,30,10] lst[-3:-1] → [30,40] lst[3:] → [40,50]

lst[::2] → [10,30,50] lst[:] → [10,20,30,40,50] płytka kopia sekwencji

Brak wskazania wycinka → od początku / do końca.

W sekwencjach mutowalnych (lista) (list), usuń za pomocą del lst[3:5] i modyfikuj z przydziałem lst[1:4]=[15,25]

### Logika Boole'a

Porównania: < > <= >= == != (wyniki logiczne)

<= >= == !=

a and b logiczne i oba jednocześnie

a or b logiczne lub jeden lub inny lub oba

pułapka: and i or zwraca wartość a lub b (według najkrótszej oceny).

⇒ upewnij się, że a i b są wartościami logicznymi.

not a logiczne nie

True False } stałe prawda i fałsz

### Bloki instrukcji

blok rodzica:

```
blok instrukcji 1...
:
blok rodzica:
blok instrukcji 2...
:
następna instrukcja po bloku 1
```

wcięcie!

skonfiguruj edytor tak, aby wstawiał 4 spacje w miejsce wcięcia.

### Importy modułów / nazw

modul truc ↔ plik truc.py

from monmod import nom1, nom2 as fct

→ bezpośredni dostęp do nazwy; zmiana nazwy za pomocą as

import monmod → dostęp przez monmod.nom1 ...

moduły i pakiety przeszukane w ścieżce python path (por sys.path)

### Instrukcje warunkowe

blok instrukcji wykonywany tylko jeśli warunek jest prawdziwy

if warunek logiczny: blok instrukcji

Może iść z kilkoma elif, elif... i tylko jednym końcowym else. Wykonywany jest tylko blok pierwszego prawdziwego warunku.

z var x:

if bool(x) == True: ↔ if x:

if bool(x) == False: ↔ if not x:

```
if age <= 18:
    state = "Dziecko"
elif age > 65:
    state = "Emeryt"
else:
    state = "Pracownik"
```

### Matematyka

liczby zmiennoprzecinkowe... wartości przybliżone

Operatory: + - \* / // % \*\*

Priorytet (...)

@ → macierz × python3.5+ numpy

(1+5.3)\*2 → 12.6

abs(-3.2) → 3.2

round(3.57,1) → 3.6

pow(4,3) → 64.0

zwykła kolejność operacji

### Matematyka

kąty w radianach

```
from math import sin, pi...
sin(pi/4) → 0.707...
cos(2*pi/3) → -0.4999...
sqrt(81) → 9.0
log(e**2) → 2.0
ceil(12.5) → 13
floor(12.5) → 12
```

moduły math, statistics, random, decimal, fractions, numpy, itp. (por. doc)

### Wyjątki

Sygnalizacja błędów: raise ExcClass(...)

Przetwarzanie błędów: try:

normalny blok przetwarzania

except Exception as e: blok przetwarzania błędów

finally blok do ostatecznego przetworzenia we wszystkich przypadkach

### Instrukcja pętli warunkowej

blok instrukcji wykonywany tak długo, dopóki warunek jest prawdziwy

**while warunek logiczny:**  
 → blok instrukcji

```

s = 0
i = 1
while i <= 100:
    s = s + i**2
    i = i + 1
print("sum:", s)
  
```

inicjalizacje przed pętlą  
 warunek z co najmniej jedną wartością zmiennej (tutaj i)  
 zmień zmienną warunku!

**Kontrola pętli**  
**break** natychmiastowe wyjście  
**continue** następna iteracja  
**else** blokuj dla normalne wyjście z pętli.

Algo:  $S = \sum_{i=1}^{i=100} i^2$

uważaj na nieskończone pętle!

### Instrukcja pętli iteracyjnej

blok instrukcji wykonywany dla każdego elementu kontenera lub iteratora

**for var in sekwencja:**  
 → sekwencja blok

```

s = "tekst"
cnt = 0
for c in s:
    if c == "e":
        cnt = cnt + 1
print("znaleziony", cnt, "e")
  
```

Przejrzyj wartości sekwencji  
 inicjalizacje przed pętlą  
 zmienna pętli, przypisanie zarządzane przez instrukcję for  
 Algo: zlicza liczbę e w ciągu znaków.

pętla na dict / set ⇒ pętla na sekwencjach klawiszy  
 użyj wycięć do zapętlenia podzbioru sekwencji

### Wyświetlanie

```
print("v=", 3, "cm :", x, ", ", y+4)
```

elementy do wyświetlenia: wartości literalne, zmienne, wyrażenia

**print** opcje:

- `sep=" "` separator elementów, domyślna spacja
- `end="\n"` koniec funkcji print, domyślna nowa linia
- `file=sys.stdout` drukuj do pliku, domyślne standardowe wyjście

**Wejście**

```
s = input("Instrukcje: ")
```

`input` zawsze zwraca **string**, przekonwertuj go na wymagany typ (por. ramka *Konwersje typów* po drugiej stronie).

### Przejrzyj indeks sekwencji

- modyfikuj element w indeksie
- dostęp do elementów wokół indeksu (przed / po)

```
lst = [11, 18, 9, 12, 23, 4, 17]
lost = []
for idx in range(len(lst)):
    val = lst[idx]
    if val > 15:
        lost.append(val)
        lst[idx] = 15
print("zmieniony:", lst, "- zagubiony:", lost)
```

Algo: wartości graniczne większe niż 15, zapamiętywanie utraconych wartości.

Przejdź jednocześnie przez indeks i wartości sekwencji:  
**for idx, val in enumerate(lst):**

### Ogólne operacje na kontenerach

`len(c)` → l. elementów  
`min(c)` `max(c)` `sum(c)` Uwaga: w przypadku słowników i zestawów, te operacje używają kluczy.  
`sorted(c)` → list posortowana kopia  
`sorted in c` → boolean, operator członkostwa **in** (brak **not in**)  
`enumerate(c)` → iterator na (index, wartość)  
`zip(c1, c2...)` → iterator krotek zawierających elementy w tym samym indeksie  
`all(c)` → True jeśli wszystkie elementy c mają wartość true, w przeciwnym razie False  
`any(c)` → True jeśli przynajmniej jedna pozycja c została uznana za prawdziwą, w przeciwnym razie False  
*Specyficzne dla kontenerów uporządkowanych sekwencji (listy, krotki, ciągi znaków, bajty...)*  
`reversed(c)` → odwrócony iterator `c*5` → duplikat `c+c2` → część wspólna  
`c.index(val)` → pozycja `c.count(val)` → liczba wydarzeń

`import copy`  
`copy.copy(c)` → płytka kopia kontenera  
`copy.deepcopy(c)` → głęboka kopia kontenera

### Sekwencje całkowite

`range([start, koniec], [krok])`  
 start domyślnie 0, koniec nie zawarte w sekwencji, krok podpisany, domyślny 1

`range(5)` → 0 1 2 3 4      `range(2, 12, 3)` → 2 5 8 11  
`range(3, 8)` → 3 4 5 6 7      `range(20, 5, -5)` → 20 15 10  
`range(len(seq))` → sekwencja indeksów wartości w seq  
 range zapewnia niezmienną sekwencję liczb całkowitych konstruowanych w razie potrzeby

### Operacje na listach

- `lst.append(val)` dodaj element na końcu
- `lst.extend(seq)` dodaj sekwencję elementów na końcu
- `lst.insert(idx, val)` wstaw element w indeksie
- `lst.remove(val)` usuń pierwszą pozycję z wartością val
- `lst.pop([idx])` usuń i zwróć element w indeksie idx (domyślnie ostatni)
- `lst.sort()` `lst.reverse()` sortowanie / odwracanie listy na miejscu

### Definiowanie funkcji

nazwa funkcji (identyfikator)  
 nazwane parametry

```
def fct(x, y, z):
    """dokumentacja"""
    # blok instrukcji, obliczanie res itp.
    return res
```

wartość wynikowa połączenia, jeśli nie została obliczona, wynik do zwrócenia: **return None**

parametry i wszystkie zmienne tego bloku istnieją tylko w bloku i podczas wywołania funkcji (myśl o „zamej skrzynce”)

Zaawansowane: `def fct(x, y, z, *args, a=3, b=5, **kwargs):`  
 \*args zmienne argumenty pozycyjne (→ tuple), wartości domyślne,  
 \*\*kwargs zmienna o nazwie argumenty (→ dict)

### Operacje na słownikach

`d[klucz]=wartość` `d.clear()`  
`d[klucz]` → wartość `del d[klucz]`  
`d.update(d2)` { zaktualizuj/dodaj parę }  
`d.keys()` → iterowalne poglądy na klucze / wartości / skojarzenia  
`d.values()`  
`d.items()`  
`d.pop(klucz[,domyślny])` → wartość  
`d.popitem()` → (klucz, wartość)  
`d.get(klucz[,domyślny])` → wartość  
`d.setdefault(klucz[,domyślny])` → wartość

### Operacje na zbiorach

Operatory:  
 | → suma (znak pionowej kreski)  
 & → powiązanie  
 - ^ → różnica/różnica symetryczna  
 < <= > >= → relacje zawieranie  
 Operatory istnieją również jako metody.

`s.update(s2)` `s.copy()`  
`s.add(klucz)` `s.remove(klucz)`  
`s.discard(klucz)` `s.clear()`  
`s.pop()`

### Wywołanie funkcji

`r = fct(3, i+2, 2*i)`  
 przechowywanie / użytkowanie zwrócona wartość jeden argument na parametr

jest to użycie nazwy funkcji z nawiasami, która wywołuje

Zaawansowane: `fct()` `fct()`

### Pliki

przechowywanie danych na dysku i ich odczytywanie

```
f = open("plik.txt", "w", encoding="utf8")
```

plik zmiennej do operacji    nazwa pliku na dysku (+ ścieżka ...)  
 tryb otwierania: 'r' czytaj, 'w' zapisz, 'a' dodaj  
 kodowanie: utf8, ascii, latin1 ...

**zapis**  
`f.write("test")`  
`f.writelines(lista linii)`

**wczytanie**  
`f.read([n])` → następane znaki jeśli n nie określono, czytaj do końca!  
`f.readlines([n])` → lista następnych wierszy  
`f.readline()` → następna linia

tryb tekstowy t domyślne (odczyt/zapis str), możliwy tryb binarny b (odczyt/zapis bytes). Konwertuj z / na wymagany typ!  
`f.close()` nie zapomnij zamknąć pliku po użyciu!

`f.flush()` zapisz pamięć podręczną    `f.truncate([rozmiar])` Zmień rozmiar postępowo odczytu / zapisu sekwencyjnie w pliku, modyfikowalny za pomocą:  
`f.tell()` → pozycja    `f.seek(pozycja[,źródło])`

**Bardzo częste:** otwieranie za pomocą chronionego bloku (automatyczne zamykanie) oraz pętla odczytu w wierszach pliku tekstowego:

```
with open(...) as f:
    for line in f:
        # przetwarzanie line
```

### Operacje na łańcuchach

`s.startswith(prefix[,start[,koniec]])` `s.endswith(suffix[,start[,koniec]])` `s.strip([znaki])`  
`s.count(sub[,start[,koniec]])` `s.partition(sep)` → (przed, sep, po)  
`s.index(sub[,start[,koniec]])` `s.find(sub[,start[,koniec]])`  
`s.is...()` testy na kategoriach znaków (ex. `s.isalpha()`)  
`s.upper()` `s.lower()` `s.title()` `s.swapcase()`  
`s.casefold()` `s.capitalize()` `s.center([szerokość, wypełnienie])`  
`s.ljust([szerokość, wypełnienie])` `s.rjust([szerokość, wypełnienie])`  
`s.zfill([szerokość])` `s.encode(kodowanie)` `s.split([sep])` `s.join(seq)`

dyrektywy formatujące    wartości do sformatowania

**Formatowanie**  
`"modele{} {} {}".format(x, y, r)` → str  
`"wybór: formatowanie! konwersja"`

Wybór:  
 2    "{: +2.3f}".format(45.72793)  
 nom    → '+45.728'  
 0. nom    "{1:>10s}".format(8, "toto")  
 4[klucz]    → 'toto'  
 0[2]    "{x!r}".format(x="I'm")  
           → 'I\'m'

Formatowanie:  
 wypelnij char    wyrównanie znak    min.szerokość    precyzja-maks.szerokość    rodzaj

`<>+ = - + - spacja` 0 na początku do wypełniania 0  
 l. całkowita: b dwójkowy, c znak, d dziesiętny (domyślne), o ósemkowy, x lub X hex  
 zmienionoprzecinkowa: e lub E wykładniczy, f lub F punkt stały, g lub G odpowiednie (domyślne),  
 łańcuch znaków: s ...    % procent

o **Konwersja** : s (czytelny tekst) lub r (reprezentacja dosłowna)

dobry nawyk: nie modyfikuj zmiennej pętli