Podstawy programowania w języku Python

Dzień 3

Agenda

Dzień 3

- praca poza zajęciami
- szerzej o formatowaniu i indeksowaniu
- instrukcje warunkowe
- nowe typy danych: **range**, **listy**, **tuple**
- pętle: **while**, **for**
- instrukcje: **pass**, **continue**, **break**

Formatowanie

białe znaki / znaki specjalne:

\n \t \ ” ‘

formatowanie:

r"encyklopedia"

f"wartość zmiennej nazwa = **{nazwa}**"

"wartość to: **%s**" **%** zmienna

"wartość to: **%f**" **%** zmienna

"wartość to: **%d**" **%** zmienna

"wartość to: **{}, {}**".**format**(**zmienna1, zmienna2**)

"wartość to: **{}, {a}**".**format**(**zmienna1, a=zmienna2**)

Indeksowanie

zdanie = "encyklopedia"

zdanie[4]

zdanie[-3]

zdanie[2:8]

zdanie[:7]

zdanie[8:]

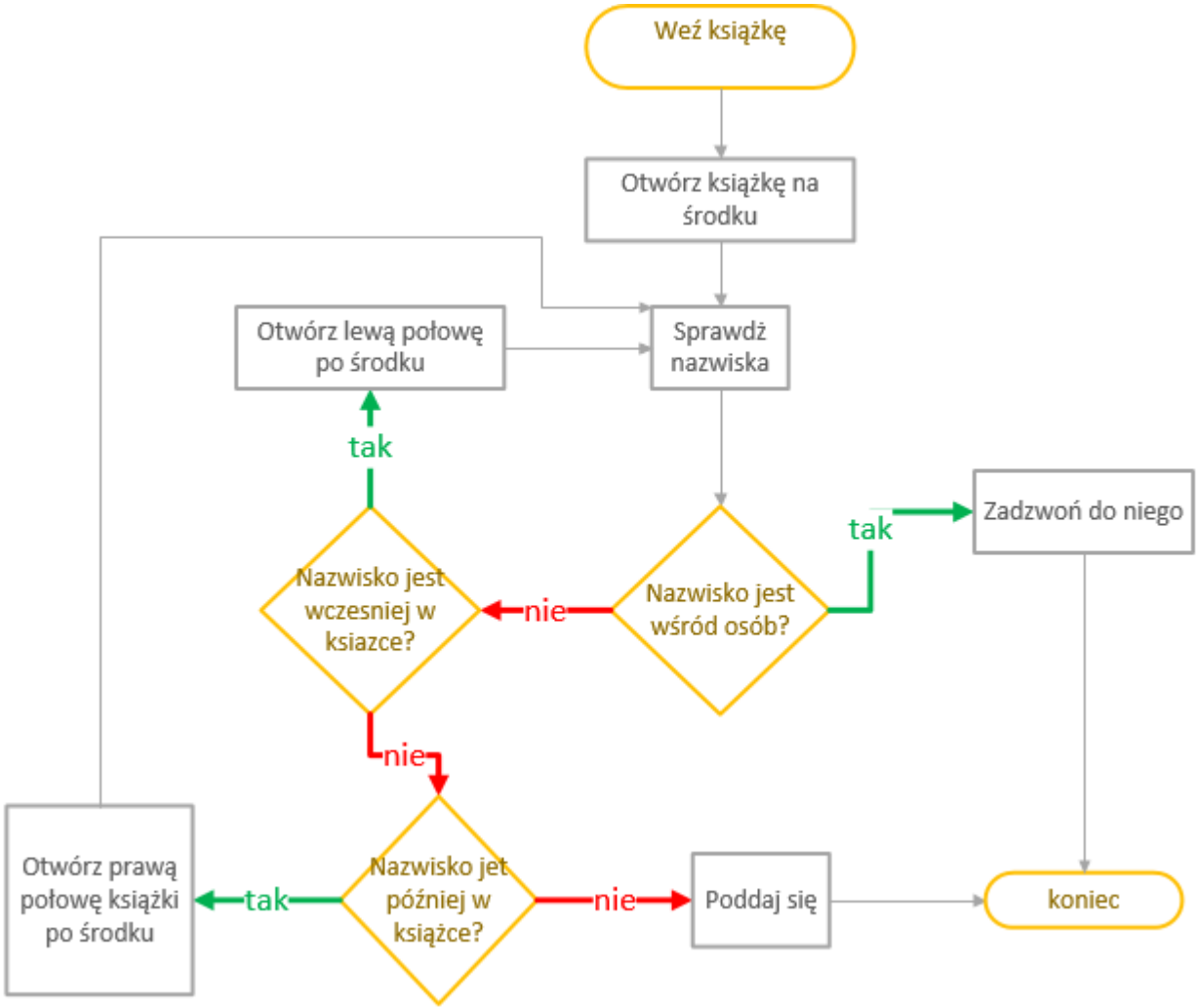
zdanie[1:7:2]

zdanie[::-1]

Instrukcje warunkowe

1. weź książkę telefoniczną
2. otwórz książkę na środku
3. zobacz nazwiska
4. jeśli ”Wojtkowiak” jest wśród osób
5. zadzwoń do niego
6. w przeciwnym razie jeśli ” Wojtkowiak” jest wcześniej w książce
7. otwórz lewą połowę po środku
8. Idź do kroku 3
9. w przeciwnym razie jeśli ” Wojtkowiak” jest później w książce
10. otwórz prawą połowę po środku
11. idź do kroku 3
12. w przeciwnym razie
13. poddaj się

Instrukcje warunkowe



Instrukcje warunkowe

```
if (warunek):
        # jakiś kod wykonany gdy warunek prawdziwy
elif (inny warunek):
        # kod wykonany gdy warunek w if był fałszywy
        # warunek w tym elif musi być prawdziwy aby ten kod wykonać
elif (inny warunek):
        # elif-ów może być wielu. lub żadnego, kod wew. elif wykona się tylko gdy
                wszystkie wyższe warunki były fałszywe
else:
        # przypadek domyślny, tu nie sprawdzamy warunku, kod w else
        # będzie wykonany gdy wszystkie w if- elif były fałszywe
        # else może być tylko jeden lub wcale
```

Tablica prawdy

A	B	A and B	A or B	not A
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

range()

class ***range(****stop****)***

class ***range(****start, stop[, step]****)***

to nie jest funkcja

sekwencyjny niemutowalny typ danych

bardzo wydajny

range(3)

<0, 1, 2>

range(4, 8)

<4, 5, 6, 7>

range(0, 10, 3)

<0, 3, 6, 9>

list()

class list([iterable])

[]

```
# to nie jest funkcja

# mutowalny typ danych

# możemy indeksować, slice'ować...
```

```
lista1 = [1, 2, 3]

lista2 = ["kwiatek", "doniczka", "ziemia", "woda"]

lista3 = []

lista4 = [1, "dwa", 3, 4]

lista5 = list("dwa")

lista6 = list(1)

lista7 = list(range(2,5))
```

Listy zagnieżdżone

```
lista = [ [1,2,3],[4,5,6],[7,8,9] ]

lista = [
                                    [1,2,3],
                                    [4,5,6],
                                    [7,8,9]
                        ]

print(lista[1][2])
```

tuple() - krotka

class tuple(*[iterable]****)***

()

```
# to nie jest funkcja

# niemutowalny typ danych

# możemy indeksować, slice'ować...
```

```
krotka1 = (1, 2, 3)

krotka2 = ("kwiatek", "doniczka", "ziemia", "woda")

krotka3 = ()

krotka4 = (1, "dwa", 3, 4)

krotka5 = tuple("dwa")

krotka6 = tuple(1)

krotka7 = list(range(2,5))
```

Listy zagnieżdżone

```
wyrazy = ("raz", "dwa", "trzy")

wyrazy[0] = "jeden"

print(wyrazy)
```

rozpakowywanie

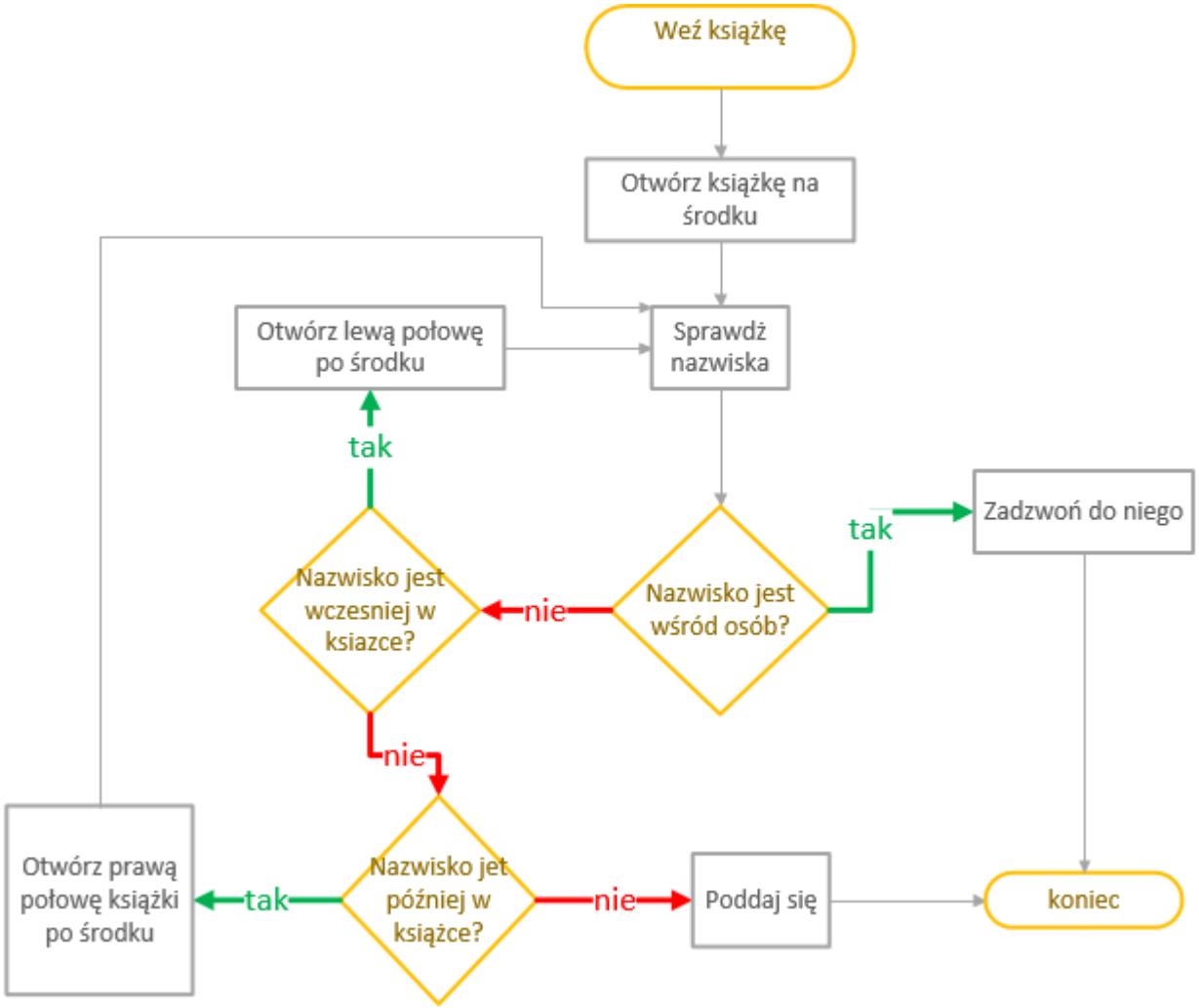
```
cyfry = range(3)
i, j, k = cyfry

liczby = [23, 78, 12]
x, y, z = liczby

wyrazy = ("raz", "dwa", "trzy")
a, b, c = wyrazy
```

Pseudokod

Pętle

1. weź książkę telefoniczną
2. otwórz książkę na środku
3. zobacz nazwiska
4. jeśli ”Wojtkowiak” jest wśród osób
5. zadzwoń do niego
6. w przeciwnym razie jeśli ” Wojtkowiak” jest wcześniej w książce
7. otwórz lewą połowę po środku
8. Idź do kroku 3
9. w przeciwnym razie jeśli ” Wojtkowiak” jest później w książce
10. otwórz prawą połowę po środku
11. idź do kroku 3
12. w przeciwnym razie
13. poddaj się

Schemat blokowy



Pętla while

while* (wartość logiczna True) *:

```
    # kod który będzie powtarzany tak
    # długo dopóki spełniony będzie warunek
    ”update wartości logicznej”
```

pass, continue, break

pass

```
# nic nie robi :)
```

continue

```
# program pomija pozostałe instrukcje w bloku i
# wraca do sprawdzenia warunku (while) lub do kolejnego
# elementu (for)
```

break

```
# działanie pętli jest przerywane, program przechodzi do
# kolejnej instrukcji po całym bloku pętli
```

Pętla for

for element in zbiór/zakres :

```
    # kod który wykona się tyle razy ile jest elementów zbioru/zakresu*
    # w tym czasie np. możemy zrobić coś z elementem bo jest on dostępny
    # w ramach pętli for
```

* - są instrukcje którymi możemy to zmienić

enumerate, zip

```
for indeks, element in enumerate(kolekcja):
        # enumerate daje nam dwie wartości: indeks
        # bieżącego elementu oraz ten element

for element_a, element_b in zip(kolekcja_a, kolecja_b):
        # daje nam elementy z tej samej pozycji w kilku kolekcjach;
        # gdy kolekcje są różnej długości, wielkość najkrótszej
        # kolekcji będzie brana przy ilości powtórzeń pętli
```

DZIĘKUJĘ