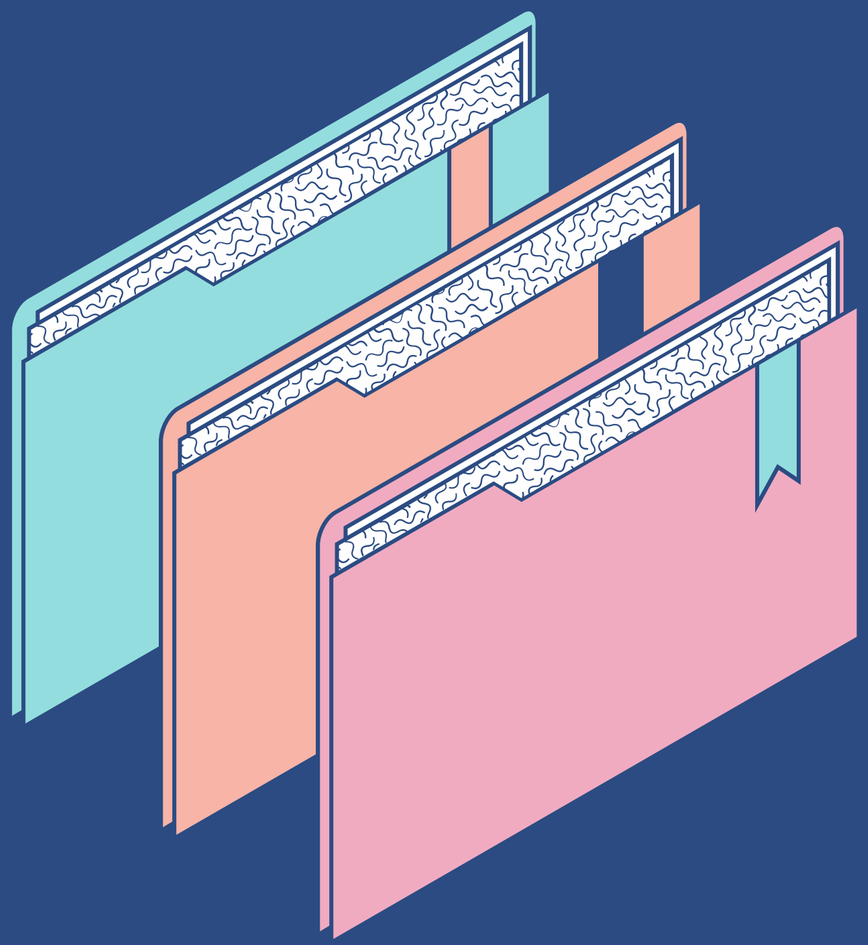
GRIK

27.01.2023

Warsztat z programowania

Małgorzata Trąbka



PLAN NA DZISIAJ

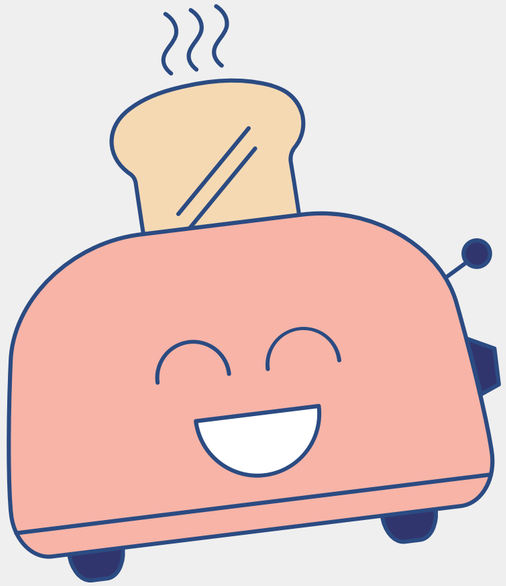
1. Czym jest programowanie?
2. Pierwszy program
3. Zmienne
4. Instrukcje warunkowe
5. Pętle
6. Listy

Czym jest programowanie?

To proces projektowania, tworzenia, testowania i utrzymywania kodu źródłowego programów komputerowych lub urządzeń mikroprocesorowych (mikrokontrolery). Kod źródłowy jest napisany w języku programowania, z użyciem określonych reguł, może on być modyfikacją istniejącego programu lub czymś zupełnie nowym.

A prościej?

Program komputerowy jest zapisem instrukcji, pozwalających na osiągnięcie określonego celu.



Przykład instrukcji robienia tostów:

1. Podłącz wtyczkę tostera do gniazdka.
2. Włóż pieczywo przeznaczone do opiekania do kieszeni tostera.
3. Wybierz odpowiedni stopień opiekania.
4. Wciśnij dźwignię na prawym boku tostera.
5. Poczekaj aż pieczywo wysunie się do góry.
6. Wyjmij pieczywo z kieszeni tostera.

Języki programowania

Do porozumiewania się z komputerem niezbędna jest znajomość języka programowania. Na tych warsztatach będziemy wykorzystywać język Python w wersji 3.X

Python

Python jest językiem wysokiego poziomu, który w założeniu ma być dla ludzi stosunkowo prosty do czytania i pisania, a dla komputerów łatwy do odczytywania i przetwarzania.

Czas na pierwszy program!

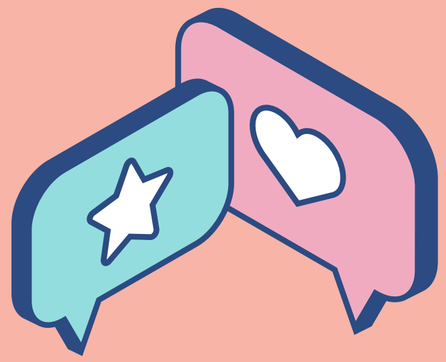
https://github.com/MTrabka/wzp

Do uruchomienia programu wystarczy zainstalowany Python i notatnik.
Pliki z programem zapisujemy stosując rozszerzenie .py

```
print('Hello world!')
```

Podczas warsztatu będziemy korzystać jednak z Jupyter Notebook

Daj głos! - funkcja print()



Pierwszym poznanym poleceniem jest „print"

- Służy do wyświetlania tekstu.
- Należy pamiętać, aby tekst zawierał się pomiędzy cudzysłowami lub apostrofami oraz nawiasami okrągłymi.

Niektóre znaki są szczególne!

\'	wyświetlenie apostrofu
\"	wyświetlenie cudzysłowu
\n	przejście do nowej linii
\t	tabulacja
\\	wyświetlenie ukośnika

Służą do przechowywania informacji oraz manipulowania nimi

Można przyjąć, że to pewne wydzielone miejsce w pamięci komputera, któremu nadajemy imię

ZMIENNE

SCHEMAT ZAPISU W PYTHONIE

nazwa_zmiennej = wartość zmiennej

```
a=4
```

Typy zmiennych

Integer i float

Służą do przechowywania liczb

- integer do przechowywania liczb całkowitych
- float do przechowywania liczb rzeczywistych

```
zmienna_int=4
zmienna_float=3.5
```

String

Służy do przechowywania łańcuchów znaków

```
tekst="Jestem stringiem"
```

Bool

Służy do przechowywania zmiennych binarnych

```
zmienna_bool1 = True
zmienna bool2 = False
```

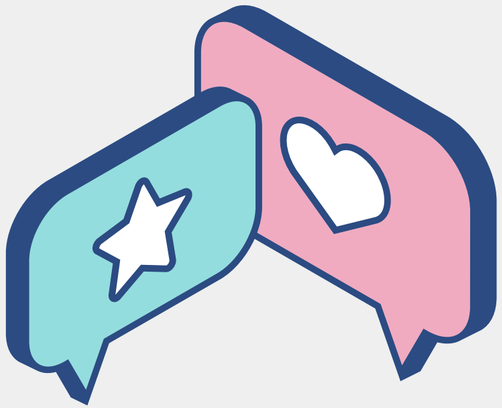
Wyświetlanie zmiennych

Do wyświetlania zmiennych służy funkcja print()

```
zmienna = 4
print(zmienna)
```

A co stanie się jeżeli zapiszemy to w ten sposób?

```
zmienna = 4
print("zmienna")
```



Operacje na zmiennych

- Na zmiennych int i float można wykonywać operacje matematyczne

```
zmienna1 = 4
zmienna2 = 6
zmienna3 = zmienna1 + zmienna2
```

- Zmienne typu string można ze sobą łączyć

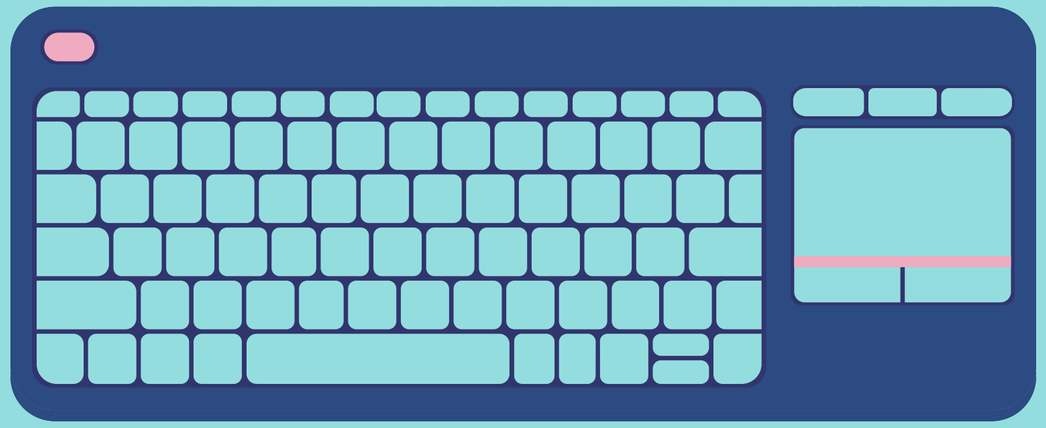
```
zmienna1 = "dzien"
zmienna2 = "dobry"
zmienna3 = zmienna1 + zmienna2
```

Pobieranie znaków z klawiatury

Do pobierania znaków z klawiatury służy funkcja input()

```
print("Ile masz lat?")
wiek=input()
print("Masz ", wiek, " lat")
```

Funkcja input()



- Domyślnie pobiera łańcuchy znaków.
- Kiedy zachodzi potrzeba, taki łańcuch należy zamienić na liczbę.
- Do konwersji zmiennych używamy funkcji int() lub float().

```
print("Ile masz lat?")
wiek=int(input())
print("Za dwa lata, będziesz miał ", wiek + 2, " lat")
```

Instrukcje warunkowe

Jeżeli masz ochotę zjeść ciastko, zjedz ciastko

```
if (warunek):
    instrukcja1
    instrukcja2
    instrukcja3
else:
    instrukcja4
    instrukcja5
    instrukcja6
```

Instrukcje warunkowe służą do sprawdzania prawdziwości wyrażenia warunkowego.

Schemat:

- Jeżeli warunek jest spełniony, wykonywane są instrukcje 1, 2 i 3.
- W przeciwnym wypadku, wykonywane są instrukcje 4, 5 i 6.

Wcięcia w kodzie

```
print("Ile masz lat?")
wiek = int(input())
if wiek >= 18:
    print("Jesteś pełnoletni")
```

W języku Python to wcięcia determinują, co "należy" do bloku instrukcji "if"

- Uważa się, że optymalne wcięcie powinno składać się z 4 spacji.
- Można użyć tabulacji, ale nie jest to polecane.

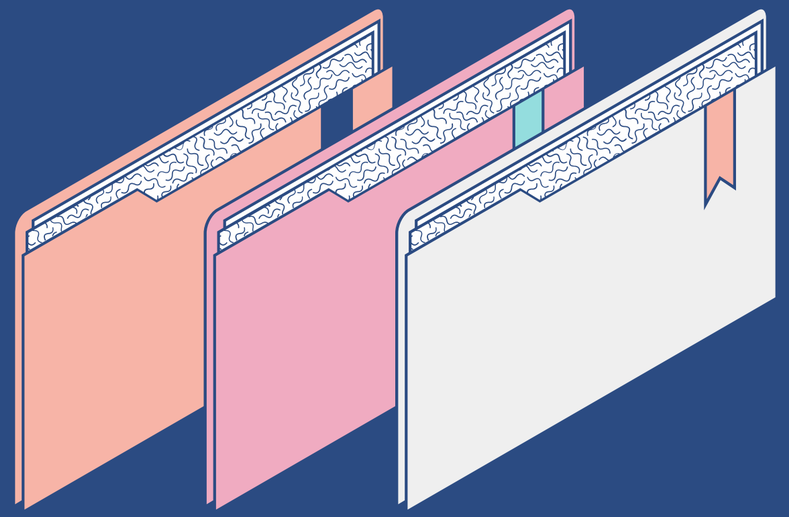
Operatory porównania

==	równa się
!=	nie równa się
<	mniejsze niż
>	większe niż
<=	mniejsze lub równe
>=	większe lub równ

Instrukcje warunkowe "if... else..."

```
print("Ile masz lat?")
wiek = int(input())
if wiek >= 18:
    print("Jesteś pełnoletni")
else:
    print("Nie jesteś pełnoletni")
```

Aby informacja zwrotna była wyświetlana także kiedy warunek nie jest spełniony, należy skorzystać z polecenia "else"



Instrukcje warunkowe "if... elif...else..."

Jeśli jest kilka warunków do sprawdzenia,
wykorzystujemy polecenie "elif".
"elif" jest skrótem od else if, polecenia znanego z innych języków programowania

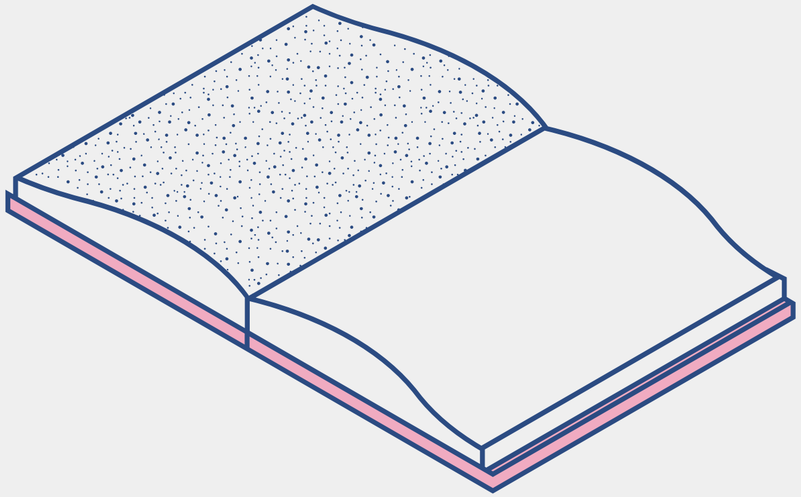
```
print("Ile masz lat?")
wiek = int(input())
if wiek <= 12:
    print("Jesteś dzieckiem")
elif wiek <= 19:
    print("Jesteś nastolatkiem")
else:
    print("Jesteś dorosły")
```

W pierwszym programie

- Najpierw sprawdzany jest pierwszy warunek
- Jeśli nie został spełniony, sprawdzany jest kolejny
- Jeśli drugi warunek nie jest spełniony, sprawdzany jest kolejny itd. aż do "else"
- Jeśli dany warunek jest spełniony, pozostałe nie są sprawdzane!

W drugim programie

Sprawdzane są wszystkie warunki, niezależnie od tego, czy któryś po drodze został spełniony



Operatory logiczne

Do budowania warunków można wykorzystać operatory znane z zajęć z Logiki: or, and i not. Wszystkie zasady mają zastosowanie w programowaniu

```
if p and q:
    print("Prawda")
else:
    print("Fałsz")
```

Warunki zagnieżdżone

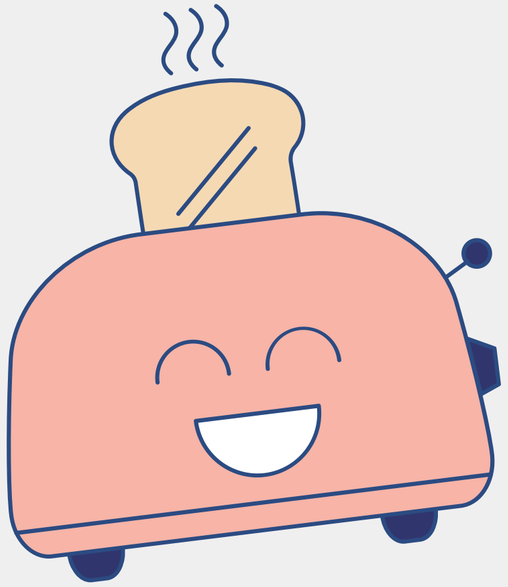
Warunek zagnieżdżony to warunek osadzony w innym warunku.

- Zagnieżdżony warunek wymaga podwójnego wcięcia!

```
if (warunek1):
    if (warunek2):
        instrukcja1
else:
    instrukcja2
```

Pętle

Wykonywanie czegoś w pętli, oznacza wielokrotne wykonywanie pewnego zestawu czynności

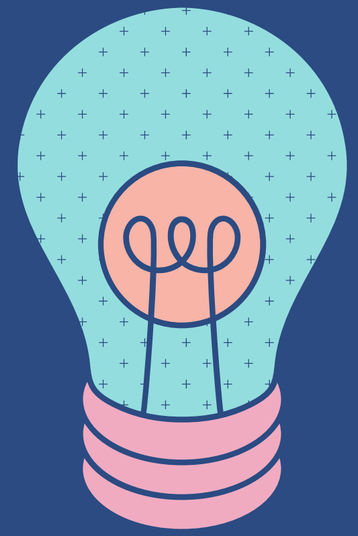


Przykład instrukcji robienia tostów:

1. Podłącz wtyczkę tostera do gniazdka.
2. Włóż pieczywo przeznaczone do opiekania do kieszeni tostera.
3. Wybierz odpowiedni stopień opiekania.
4. Wciśnij dźwignię na prawym boku tostera.
5. Poczekaj aż pieczywo wysunie się do góry.
6. Wyjmij pieczywo z kieszeni tostera.

Jeśli masz jeszcze chleb, powtórz czynności od punktu 1

Pętla "for"



- Pętlę "for" wykorzystujemy, kiedy wiemy dokładnie ile razy ma zostać wykonana

```
for i in range(10):
    print(i)
```

- Litera "i" to tak zwany licznik. Jest to zmienna, pod którą zapisywany jest numer aktualnie wykonywanej iteracji
- Funkcja range() służy do określenia od jakiej liczby ma rozpocząć się iteracja i na jakiej skończyć

Więcej o funkcji range()

- Funkcja przyjmuje do 3 argumentów
  - argument pierwszy służy do ustalenia początku zakresu (domyślnie 0)
  - argument drugi służy do ustalenia końca zakresu (-1 od podanego)
  - argument trzeci jest krokiem, który ustala co ile ma zwiększać się licznik w kolejnych iteracjach

```
for i in range(0,30,3):
    print(i)
```

Pętle zagnieżdżone

- Pętle zagnieżdżone, czyli pętle w pętli. Służą do tego, aby daną pętlę wykonać wielokrotnie.

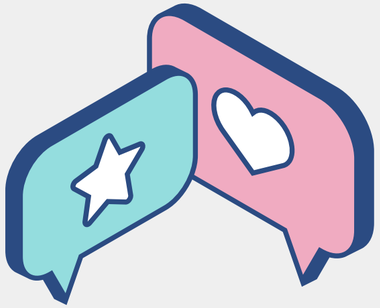
```
for i in range(zakres1):
    for j in range(zakres2):
        instrukcja
```

Pętla "while"

**Pętla "while" wykonywana jest aż do momentu, kiedy określony warunek jest prawdziwy.
Nie ma odgórnego założenia, ile razy ta pętla się wykona w przeciwieństwie do pętli "for"**

```
cyfra = 0
while cyfra != 10:
    print(cyfra)
    cyfra += 1
```

Wciśnij przycisk aby zakończyć program



- Dotychczasowe programy wykonywały się tylko raz. Aby skorzystać z nich ponownie, należało uruchomić ponownie program.
- Znane nam z życia programy się tak nie zachowują. Decyzję o zakończeniu wykonywania danego programu podejmuje użytkownik.

```
znak = ""
print("aby zakończyć program, wciśnij \"q\"")
while znak != "q":
print("nie wcisnąłeś \"q\"")
znak = input()
```

Pętla nieskończona i polecenie break

Polecenie break służy do natychmiastowego przerwania wykonywanej pętli

```
print("aby zakończyć program, wciśnij \"q\"")
while True:
    znak = input()
    if znak == "q"
        break
    print("nie wcisnąłeś \"q\"")
```

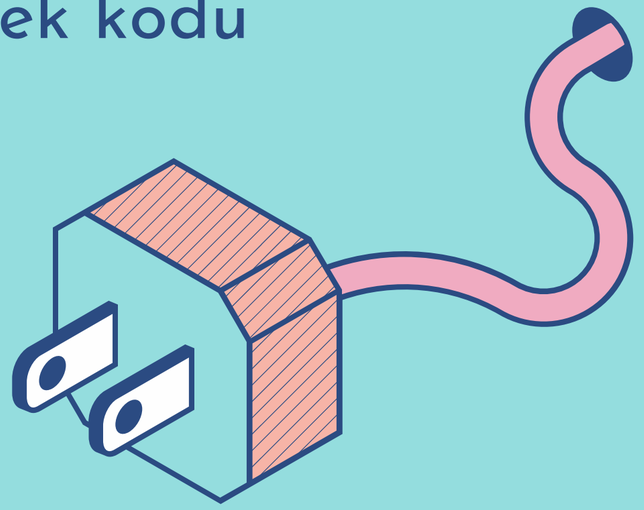
Typy tablicowe

Tworzenie zbiorów danych tylko za pomocą zmiennych jest bardzo kłopotliwe np.:

- żeby zapisać informację o wieku 50 osób, trzeba zadeklarować 50 zmiennych, a to 50 linijek kodu

```
wiek_osoby1 = 15
wiek_osoby2 = 23
.
.
.
wiek_osoby49 = 43
wiek_osoby50 = 12
```

- w przyszłym roku należy zwiększyć wartość zmiennej "wiek_osobyX" o 1, a to kolejne 50 linijek kodu



Typy tablicowe - listy

Z pomocą przychodzą typy tablicowe. W Pythonie wyróżniamy: krotki, listy, zbiory oraz słowniki. My zajmiemy się listami.
Tablica jest zbiorem komórek w pamięci.

- Każda komórka może posiadać różną zawartość.
- Każda komórka posiada swój indeks (numer), co znacznie ułatwia przeszukiwanie jej zawartości.

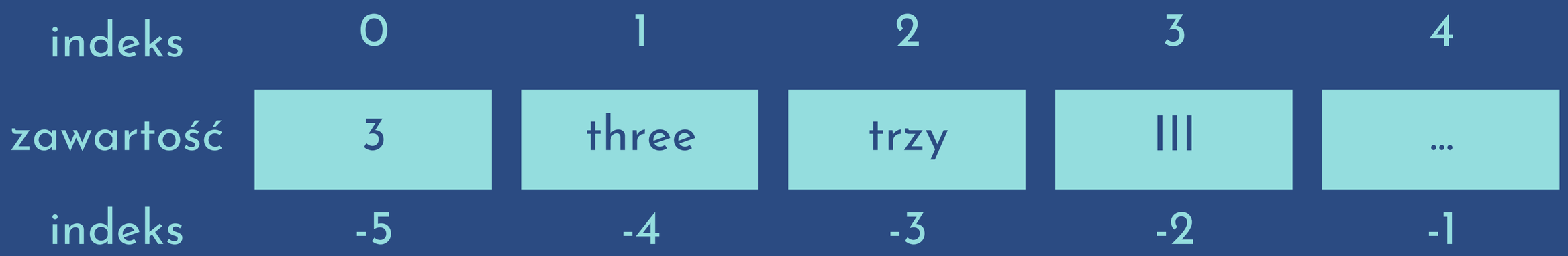
```
lista_cyfry_trzy = [3, "three", "trzy", "III", "..."]
```

- Lista, tak jak zmienna, musi mieć nazwę
- Zawartość umieszczana jest między kwadratowymi nawiasami
- Jej zawartością mogą być wszystkie typy zmiennych (int, float, str, bool)

Indeksowanie

Każde pole w liście ma swój indeks, pozwalający dostać się do zawartości danego pola.

- Indeksowanie może zaczynać się od lewej strony, wtedy pierwszym indeksem jest "0", a kolejne numery indeksów zwiększają się
- Indeksowanie może zaczynać się od prawej strony, wtedy pierwszym indeksem jest "-1", a kolejne numery indeksów zmniejszają się



Długość listy

Do wyznaczenia liczby elementów listy służy funkcja len()

```
lista_cyfry_trzy = [3, "three", "trzy", "III", "..."]
print("Liczba elementów listy to: ", len(lista_cyfry_trzy))
```

Funkcję len() najczęściej będziemy używać do wskazania zakresu pętli for - bo najczęściej operacje wykonuje się na wszystkich elementach listy

Wyświetlanie zawartości listy

- Do wyświetlania zawartości typów tablicowych warto posłużyć się pętlą "for"

```
lista_cyfry_trzy = [3, "three", "trzy", "III", "..."]
for i in range(len(lista_cyfry_trzy)):
    print(lista_cyfry_trzy[i])
```

- Gdyby zaistniała potrzeba wyświetlenia tylko części tablicy, należy modyfikować funkcję range()

Wyświetlanie zawartości listy

Łatwiejszy sposób na wyświetlenie całej listy

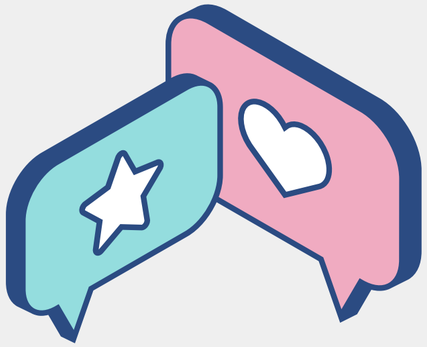
```
lista_cyfry_trzy = [3, "three", "trzy", "III", "..."]
for cyfra in lista_cyfry_trzy:
    print(cyfra)
```

- Taką pętlę możemy odczytać następująco: "Dla każdego elementu (nazwanego "cyfra") w liście "lista_cyfry_trzy" wykonaj:"

Wyświetlanie zakresu

- Fragment listy można wyświetlić podając w nawiasach indeksy elementów

```
lista_cyfry_trzy = [3, "three", "trzy", "III", "..."]
print(lista_cyfry_trzy[1:2])
print(lista_cyfry_trzy[1:4])
print(lista_cyfry_trzy[2:])
print(lista_cyfry_trzy[:3])
```



Zmiana wartości pola

```
lista_liczb = [1,2,3]
print("Podaj nową wartość zerowego elementu")
lista_liczb[0] = int(input())
```

Dodawanie elementów

Do dodawania elementów listy służy funkcja append()

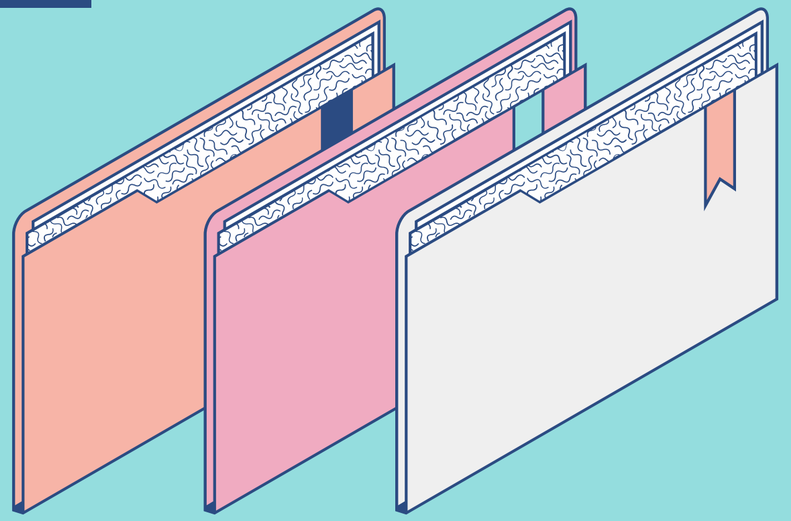
```
lista_liczb = [1,2,3]
dodaj = input("Jaki element dodać?")
lista_liczb.append(dodaj)
```

Istnieją różne metody dodawania elementów do listy. Funkcja append() dodaje nowe elementy na końcu listy. Warto zaznajomić się z funkcjami insert(), extend() itp. we własnym zakresie

Usuwanie elementów

Podobnie jak dodawanie wygląda usuwanie elementów. Do usuwania wykorzystywana jest funkcja remove()

```
lista_liczb = [1,2,3]
usun= int(input("Jaki element usunąć?"))
lista_liczb.remove(usun)
```



Uwaga!

Jeżeli z listy chcemy usunąć elementy równe 2 w następujący sposób"

```
lista_liczb = [1,2,3,2,5,2,4]
for i in range(len(lista_liczb)):
    if lista_liczb[i] == 2:
        lista_liczb.remove(lista_liczb[i])
```

Uzyskamy błąd

Usuwanie elementów

Nie wolno usuwać elementów listy, którą iterujemy! Usuwając elementy listy, zmniejszamy jej wielkość. Dlatego zostanie zgłoszony błąd. Program będzie próbował odczytać zawartość elementu tablicy, który nie istnieje.

Jak to naprawić?

```
lista_liczb = [1,2,3,2,5,2,4]
for liczba in lista_liczb[:]:
    if liczba == 2:
        lista_liczb.remove(liczba)
```

[:] oznacza, że iterujemy po klonie listy, a nie liście, na której operujemy!

Macie jakieś pytania?

Dziękuję za uwagę