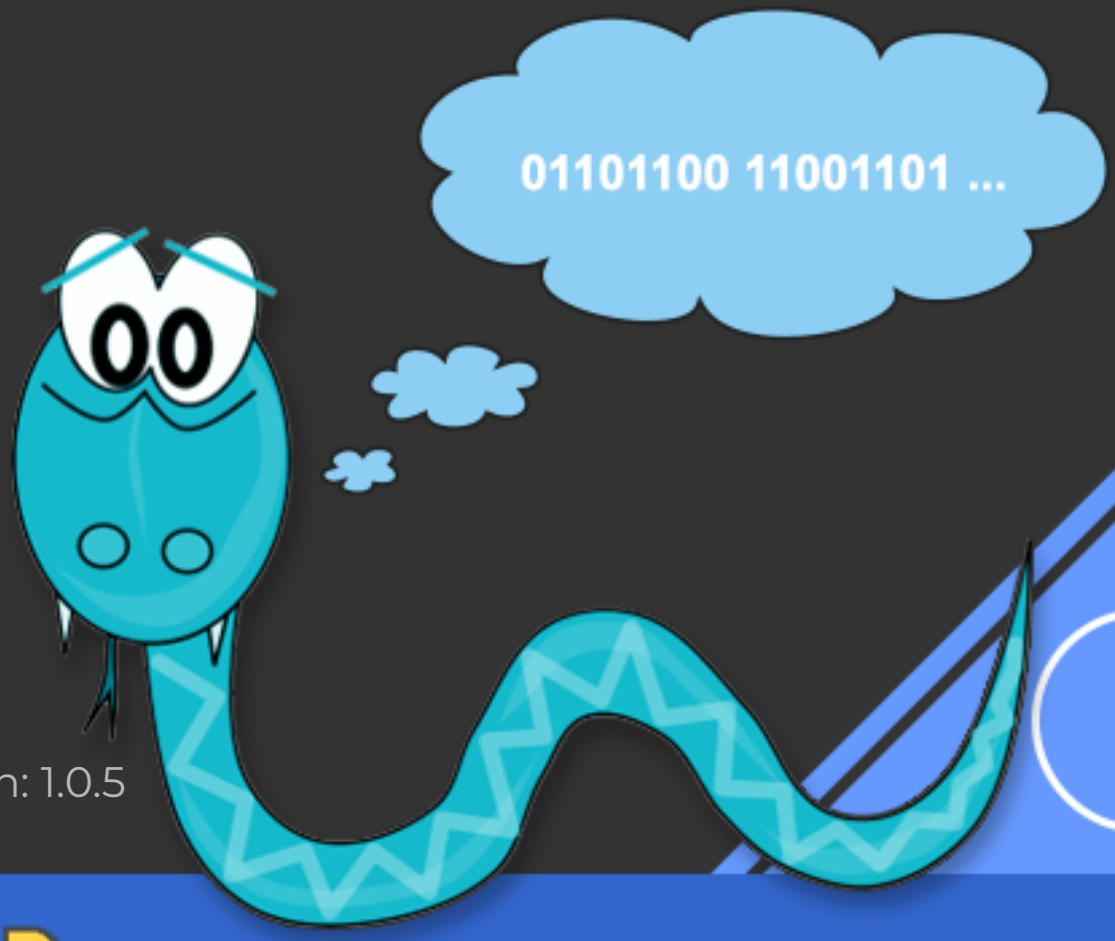
Mateusz Wiliński

CYFROWA GADZINA

Podstawy programowania w języku Python



1

--version: 1.0.5

print("Jak okiełznać cyfrową gadzinę?")

PIKADEMIA

CYFROWA GADZINA

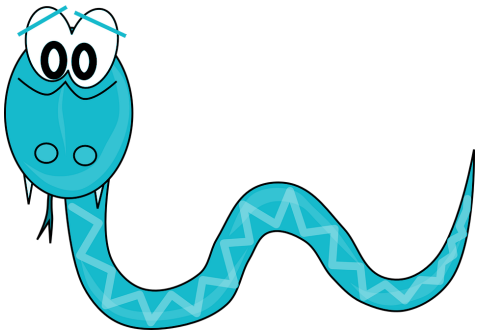
Podstawy programowania w języku Python

Ebook może być dowolnie rozpowszechniany
i używany w dowolnym celu.

Zapraszamy do współpracy przy jego tworzeniu i ulepszaniu.

Link do aktualnej wersji:

Cyfrowa gadzina



Autor: Mateusz Wiliński
Email: pikademia@gmail.com
Data ostatniej aktualizacji: 07.2022

https://www.pikademia.pl/
Kursy, ebooki, szkolenia, tutoriale

PIKADEMIA

Spis zagadnień

Spis zagadnień 2

Wprowadzenie 5

Typy danych 7

Wyświetlanie informacji - print() 8

Zmienne 9

Konkatenacja 12

Inkrementacja i dekrementacja 14

Operacje na wartościach tekstowych 15
- Indeksowanie / zakresy 15
- Wieloliniowe wartości tekstowe 17
- Escape characters 17
- Surowe łańcuchy 17

Wprowadzanie informacji do programu 18
- Przykład: Obliczanie potęgi podanej liczby 19
- Przykład: Obwód i pole powierzchni kwadratu 20

Losowa liczba 21
- Losowe liczby od 0 do 1 21
- Losowe liczby całkowite w zakresie 22
- Aliasy 23

Instrukcja warunkowa 24
- Instrukcja warunkowa 24
- Operator tenarny 26
- Indentacja 26
- Operatory porównania 27
- Operatory logiczne 28

Przykład: Suma kątów wewnętrznych w wieloboku 29
Sprawdzenie wystąpienia tekstu 30
Przykład: Działanie matematyczne 31

Funkcje 32
Funkcja 32
Funkcja z parametrem 34
Funkcja z parametrem domyślnym 35
Funkcja z return 36
Zmienne globalne i lokalne 37

Pętle 38
Pętla for 38
Pętla while 40
Przykład: Podanie hasła do skutku 41
Pętla while i for 42
Break, continue 43
Przykład: Losowe działanie matematyczne 44
Przykład: Zadanie matematyczne z odpowiedzią 45
Przykład: Wizualizacja gwiazdkowych kwadratów 46
Przykład: Gwiazdkowa choinka 46

Wyjątki try: except: 47
Przykład: Cena po obniżce 47
Przykład: Test matematyczny z punktami 48
Przykład: Gra - Zgadnij liczbę 49

Struktury danych 51
Listy 51
Listy z pętlą 52
Metody na listach 54
Przykład: Czy można zbudować trójkąt? 57
Przykład: Losowanie liczb totolotka 58

Zadanie: Różnica między min a max 59
Słownik (Dictionary) 60
Krotka (Tuple) 62
Set 63
Dodatkowe paczki z pip 65
Data i czas 67
Klasa 69
Tworzenie klasy i obiektu 69
Metody 70
Metody statyczne 71
Praca z plikami i danymi 72
Tworzenie / zapisywanie / usuwanie pliku 72
JSON 76
API requests 79
Funkcje matematyczne 81
Ciąg Fibbonacciego 82
Modulo 83
Przykład: Liczba podzielna przez 3 83
Metody wartości tekstowych 84
Tablica znaków ASCII / UNICODE 86
Przykład: Generowanie losowego hasła 88
Sekcja z zadaniami 89

Wprowadzenie

Python, lider wśród języków programowania

Python jest uznawany za jeden z najłatwiejszych do nauki języków programowania.
Jego syntax (struktura kodu) jest dużo prostszy i krótszy w porównaniu z innymi językami.
Dzięki dostępnym bibliotekom, zastosowaniu i niskiej barierze nauki, język python, wg. statystyk, jest najpopularniejszym językiem programowania w 2022 roku.

Python jest językiem wysokopoziomowym, oznacza to, że pisanie w nim programów jest po części zautomatyzowane i nie musimy się przejmować pewnymi aspektami (np. zarządzaniem pamięcią). Sprawia to, że nauka programowania jest znacznie ułatwiona ;)

Jak tworzyć kod?

Istnieje wiele narzędzi do tworzenia programów za pomocą języka python. Pliki z kodem (skrypty) muszą zostać zinterpretowane, tak, aby maszyna na której ten kod ma się wykonać, mogła go zrozumieć.
Jednym z najprostszych sposobów na rozpoczęcie nauki programowania w języku python jest skorzystanie ze środowisk programistycznych online, np:
https://www.jdoodle.com/python3-programming-online/
Na dalszym etapie nauki będziemy potrzebowali interpretera i edytora, który pozwoli tworzyć kod szybciej i wygodniej.
Wystarczy pobrać, zainstalować Python i środowisko programistyczne IDE, np:
PyCharm Community , Visual Studio Code lub inne...

Wersja wideo jak przygotować Visual Studio Code do pracy z python:
https://youtu.be/HJf22osGKu0

Typy danych

W pythonie posługujemy się danymi / informacjami różnego typu. Możemy wyróżnić 4 podstawowe typy danych:
liczby całkowite (int), np: -123, 12, 1234
liczby ułamkowe (float), np: -12.5, 0.05, 1.23, 44.95435543
wartości tekstowe (string), np: "Legnica", "Dawno temu..."
wartości logiczne (bool) określają prawdę lub fałsz: True, False

Pamiętaj!
Części dziesiętne w liczbach ułamkowych oddzielamy za pomocą kropki!
Wartości tekstowe umieszczane są wewnątrz apostrofu lub cudzysłowu.
Liczby całkowite w pythonie mogą być tak duże, na ile pozwala dostępna pamięć, oznacza to, niewyobrażalnie wielkie liczby.

Liczby ułamkowe można również zapisać w notacji wykładniczej, np: 3.52e8 co oznacza liczbę $3.52 * 10^8$, czyli 352000000

Liczby typu float mają precyzję 16 cyfr, oznacza to, że liczby zapisane z większą dokładnością zostaną zaokrąglone, np:

```
liczba = 1234567890.12345667855
print(liczba)
# wynik
# 1234567890.1234567
```

Wyświetlanie informacji - print()

W języku python, możemy wyświetlić informacje w konsoli, za pomocą metody **print()**
Wewnątrz nawiasu, jako argument, podajemy informację, która ma zostać wyświetlona. Może to być pojedyncza informacja, (liczba, tekst), wyrażenie matematyczne lub zbiór wielu elementów odpowiednio ze sobą połączonych.

```
# Przykłady użycia metody print do wypisania różnych
informacji w konsoli
print(12)
print(36.6)
print('@')
print("hello")
print(12*6)
```

Przetwarzanie kodu

Pamiętaj, że kod w skrypcie, interpretowany jest od góry w dół, od lewej do prawej strony.
Kolejność wykonywania poleceń zależy więc od miejsca ich wywołania. Polecenia wywołane na samej górze zostaną wykonane jako pierwsze.

Zmienne

Tworzenie zmiennych

Możemy sobie wyobrazić, że zmienna, to niewidzialne pudełko, wewnątrz którego możemy przechowywać informacje. Jak nazwa wskazuje, zawartość pudełka może się zmieniać.
Pudełko (czyli zmienna) musi mieć unikalną nazwę, tak, abyśmy mogli je w każdej chwili odnaleźć i zajrzeć do środka. Za każdym razem kiedy będziemy korzystać z nazwy zmiennej, interpreter odnajdzie to pudełko w pamięci komputera, zajrzy do środka i zwróci jego zawartość.
Zawartość pudełka, czyli wartość zmiennej przypisujemy za pomocą operatora przypisania, czyli znaku =.
Zmienną tworzymy podając jej nazwę, a następnie przypisujemy do niej pierwszą zawartość. W niektórych językach programowania musimy jeszcze podać jaki typ informacji będziemy przechowywać w danej zmiennej. Python sam określa jakiego typu jest wartość przypisana do zmiennej, dlatego tutaj nie jest to wymagane.

```
# Przykłady tworzenia zmiennych:
imie = "Janusz"
wiek = 23
kwota_do_zaplaty = 102.99
```

Język python jest case-sensitive, to znaczy, że rozróżnia małe i

wielkie litery alfabetu.
Zmienne: imie, Imie, IMIE to trzy zupełnie inne zmienne.

Ustalając nazwę zmiennej powinniśmy przestrzegać kilku zasad:
- zaczynamy od małej litery
- używamy liter podstawowych (bez polskich akcentów)
- można używać cyfr, ale nie można zaczynać nazwy od cyfry
- nazwy, składające się z wielu wyrazów, łączymy za pomocą notacji **snake case**, czyli łączymy wyrazy podkreśleniem '_'
- w nazwach nie stosujemy spacji

Nazwy muszą być deskryptywne, aby od razu było wiadomo, do czego służą!

Utworzone zmienne przechowują w sobie dane, do których będziemy chcieli się dostać w dalszej części kodu.
Dostęp do zmiennej uzyskujemy na podstawie jej nazwy.
Możemy pobrać zawarte w niej dane lub przypisać do niej nową wartość.

```
# Przykład utworzenia zmiennej, wyświetlenie zawartości,
nadpisanie zmiennej i ponowne wypisanie w konsoli:
liczba_punktow = 0
print(liczba_punktow)
liczba_punktow = 3
print(liczba_punktow)
```

Zmiana typu

Castowanie to zmiana typu danej wartości na inną.

```
x = str(3)    # wartość typu string ('3')
y = int('3')  # wartość typu integer (3)
z = float(3)  # wartość typu float (3.0)
print(x)
print(y)
print(z)
```

Sprawdzanie typu

W każdym momencie możemy sprawdzić jakiego typu wartość jest przechowywana w danej zmiennej:

```
liczba = 5
imie = "Ania"
print(type(liczba))
print(type(imie))
```

```
# Inny sposób utworzenia wielu zmiennych:
x, y, z = "Orange", "Banana", "Cherry"
```

Konkatenacja

Konkatenacja z operatorem + i ,

Często będziemy chcieli połączyć kilka informacji w jedno zdanie. Łączenie takie, nazywamy konkatenacją. Możemy zrobić to na wiele sposobów, np. za pomocą operatora + lub przecinka. Tworząc konkatenację z liczbami, używając operatora +, musimy zamieniać je na wartości tekstowe za pomocą metody str().

Przykłady konkatenacji z operatorem + i przecinkiem.

```
liczba_punktow = 10

# Konkatenacja z użyciem operatora +
print("Liczba puntow: " + str(liczba_punktow))

# Konkatenacja z użyciem przecinka
print("Liczba puntow:",liczba_punktow)
```

Konkatenacja, ostatecznie, tworzy treść typu tekstowego (string).

Konkatenacja przez interpolację f

Innym, eleganckim, sposobem konkatenacji, jest użycie interpolacji f.
Na początku tworzonego ciągu dodajemy literę f, a następnie kolejne zmienne podajemy wewnątrz nawiasu klamrowego.
Jest to preferowana opcja i będzie wykorzystywana w większości przykładów przedstawionych w tej publikacji.

```
# przykład 1
uczen = "Marcel"
punkty = 99
print(f"Uczeń {uczen} zdobył {punkty} punktów z egzaminu.")

# przykład 2
print(f"2 + 4 = { 2 + 4 }")
```

Inkrementacja i dekrementacja

Wartości zmiennych możemy modyfikować zwiększając jej wartość, bazując na jej poprzedniej wartości.
W takich przypadkach wykonuje się operację zwiększenia / zmniejszenia, a następnie przypisania nowej wartości do zmiennej. Zabieg taki nazywamy inkrementacją lub dekrementacją.
Najpierw podaje się operację arytmetyczną, następnie znak przypisania i wartość zwiększającą.

```
# Przykłady inkrementacji / dekrementacji
liczba_punktow = 0
liczba_punktow += 1   #zwiększenie o 1
liczba_punktow *= 3   #zwiększenie x3
liczba_punktow **= 2  #zwiększenie ^2
liczba_punktow -= 5   #zmniejszenie o 5
liczba_punktow /= 2   #zmniejszenie x2
print(liczba_punktow)
```

Inkrementację można przeprowadzać również na zmiennych typu string, zwiększając jej zawartość, np:

```
zdanie = ""
zdanie += "Ala "
zdanie += "ma "
zdanie += "kota ;)"
print(zdanie)
```

Operacje na wartościach tekstowych

Indeksowanie / zakresy

Często zachodzi potrzeba, by wyłonić z wyrazu lub zdania tylko część znaków. W wartościach tekstowych mamy do czynienia z łańcuchem znaków i każdy symbol ma swój numer porządkowy, tzw. numer indeksu.

W programowaniu, indeksowanie, czyli liczenie, zaczynamy od 0. Numer indeksu oznaczamy wewnątrz nawiasu kwadratowego [ ]

Przykład:
w wyrazie "python", znak 'h' ma numer indeksu równy 3.

```
# Wypisz pierwszy znak wyrazu "tekst"
print("tekst"[0])

# Wypisz trzeci znak wyrazu "domek"
print("domek"[2])
```

Za pomocą dwukropka(:) ustalamy zakres.
Dzięki wartościom ujemnym możemy wybrać liczby zaczynając od końca łańcucha znaków.

```
# Wypisz trzy pierwsze znaki wyrazu "tekst"
print("tekst"[0:3])


# Wypisz 2 i 3 znak podanej zmiennej
imie = "Ania"
print(imie[1:3])


# Wypisz trzy ostatnie znaki wyrazu "Programowanie" (nie)
print("Programowanie"[-3:])


# Wypisz 3 znaki wyrazu "Komputerek" licząc od tyłu, poza ostatnim znakiem (ere)
print("Komputerek"[-4:-1])
```

Wieloliniowe wartości tekstowe

Za pomocą potrójnego cudzysłowu (lub apostrofu) możemy dodać wieloliniowe wartości tekstowe, które zawierają przejścia do nowej linii.

```
tekst = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt """
print(tekst)
```

Escape characters

Znaki wyjścia pozwalają na formatowanie tekstu (przejście do nowej linii, dodanie cudzysłowu, itp).

\\	Backslash
\n	New Line
\r\n	Carriage Return with new line
\t	Tab

Surowe łańcuchy

Surowe łańcuchy tworzymy za pomocą litery r, dzięki czemu nie działają znaki wyjścia, co pomaga przy podawaniu ścieżek i tym podobnych wartości, np:

```
path = r"C:\Users\Microsoft\Windows\Start Menu\Programs"
print(path)
```

Wprowadzanie informacji do programu

Do wprowadzenia treści do programu używamy metody **input()** Wewnątrz nawiasu metody, możemy podać treść jaka zostanie wyświetlona użytkownikowi, która skłoni go do wpisania odpowiedniej treści. Najczęściej metodę input() przypisuje się do zmiennej, tak, aby móc wpisaną wartość zapisać i przetwarzać w programie.

```
# Wyświetlenie informacji na podstawie wprowadzonych danych
wprowadzone_imie = input("Podaj imie: ")
print("Witaj " + wprowadzone_imie)
```

Przykład: Obliczanie potęgi podanej liczby

```
#Obliczenie potęgi dowolnego stopnia podanej liczby
liczba = int(input("Podaj liczbę: "))
potega = int(input("Podaj potęgę: "))
wynik = liczba**potega
print(f"{liczba} ^ {potega} = {wynik}")
```

Przykład: Obwód i pole powierzchni kwadratu

```python
#Obliczanie pola i obwodu kwadratu o podanym boku
bok = float(input("Podaj bok kwadratu: "))
pole = bok * bok
obwod = 4 * bok
print(f"Pole wynosi: {pole}")
print(f"Obwód wynosi: {obwod}")
```

Losowa liczba

Losowe liczby od 0 do 1

W celu korzystania z generatora losowych liczb, należy dodać do programu bibliotekę random. Robi się to na samej górze skryptu za pomocą instrukcji import

```
import random
```

do zmiennej przypisujemy metodę generującą losową liczbę:

```
liczba = random.random()
```

Metoda random() zwraca wartość ułamkową od 0 do 1

```
#importujemy bibliotekę random
import random
#losujemy liczbę ułamkową od 0 do 1
liczba = random.random()
print(liczba)
```

Losowe liczby całkowite w zakresie

Chcąc losować liczby całkowite z podanego przedziału możemy posłużyć się metodą randrange(), która przyjmuje różne argumenty określające zakres losowanych liczb.
Standardowo wewnątrz nawiasu podajemy wartość minimalną, maksymalną i skok(zazwyczaj 1).
Wartość maksymalna nie jest wliczana do przedziału, zatem należy ją zwiększyć o 1.

```
liczba = random.randrange(min, max, 1)
```

```
#importujemy bibliotekę random
import random

#losujemy liczbę od 1 do 10
losowa_liczba = random.randrange(1, 11, 1)

#wypisujemy liczbę
print("Wylosowana liczba: ",losowa_liczba)
```

Aliasy

Importując moduły możemy skorzystać z aliasów, czyli zamienić nazwę modułu na krótszą (lub inną bardziej deskryptywną). Robimy to tak jak w przykładzie poniżej:

```
import random as ran
losowa = ran.random()
print(losowa)
```

Możemy też pobrać z danego modułu tylko tę część której potrzebujemy, np:

```
from random import randrange
losowa = randrange(0,100)
print(losowa)
```

Instrukcja warunkowa

Instrukcja warunkowa

Instrukcja warunkowa służy do podejmowania decyzji w oparciu o podane warunki. Sprawdzane warunki zawsze muszą być w takiej formie, aby ich wynik zwracał wartość typu bool, czyli prawdę lub fałsz.
Najprostsza instrukcja posiada jeden warunek po spełnieniu którego wykona się instrukcja. Używając pseudokodu możemy przytoczyć sytuację z życia wziętą:

```
jeżeli(dostanę co najmniej piątkę z matematyki):
    otrzymam świadectwo z czerwonym paskiem
```

A teraz w języku python mogłoby to wyglądać w ten sposób:

```
if(ocena >= 5):
    print("Otrzymujesz świadectwo z cz. paskiem")
```

Syntax pełnej instrukcji warunkowej:

```
import random
ocena = random.randrange(1,6)

if(ocena == 5):
    print("Super, dałeś czadu!")
elif(ocena == 4):
    print("Jest dobrze, brawo!")
else:
    print("Może kawa?")
```

Jeżeli warunek (warunki) będzie prawdziwy, wykona się instrukcja do niego przypisana, a reszta instrukcji warunkowej zostanie pominięta.
Pamiętaj, że w instrukcji warunkowej, może zostać wykonany tylko jeden blok. Będzie to pierwszy blok od góry, w którym warunek zostanie spełniony.
Jeżeli żaden z warunków nie zostanie spełniony, wykona się blok else lub nie wykona się nic, jeśli takiego bloku nie dodamy.
Więcej warunków podaje się za pomocą instrukcji **elif**

Operator tenarny

W prostych przypadkach można użyć tzw. operatora tenarnego, który można zapisać szybciej od pełnej instrukcji warunkowej.

```
warunek_prawdziwy = False
zmienna = "prawda" if warunek_prawdziwy else "nie prawda"
print(zmienna)
```

Indentacja

Język python działa na zasadzie indentacji, czyli wcięć kodu (tabulacji). Główne bloki kodu zaczynają się tuż przy krawędzi ekranu, a kolejne, wewnętrzne bloki kodu, tworzy się za pomocą tabulacji.
Standardowo, indentacja wynosi 4 spacje, ale głównie zależy to od ustawień edytora. Skrypt będzie działał prawidłowo również z indentacją na 3 spacje, ale ważne, by stosować identyczne zasady w całym skrypcie.

```
warunek = True
if(warunek):
    print("Instrukcje")     # wewnętrzny blok kodu
else:
    print("Instrukcje")     # wewnętrzny blok kodu
```

Operatory porównania

Operatory porównania służą do porównywania wartości w instrukcji warunkowej. Zauważ, że takie porównanie zawsze zwróci wartość typu bool, czyli prawdę lub fałsz.

==	Równe
!=	Nie jest równe
>	Większy niż
<	Mniej niż
>=	Większy niż lub równy
<=	Mniejszy niż lub równy

Zauważ, że sprawdzamy czy wyrażenia są równe za pomocą podwójnego znaku ==.
Pojedynczy znak = służy do przypisywania wartości, np. do zmiennej.

Operatory logiczne

Często zachodzi potrzeba, by w instrukcji warunkowej połączyć kilka warunków.

Przykład 1.
Autem można jechać, gdy kierowca ma prawo jazdy i kluczyki.
Stosujemy operator **and**, ponieważ musimy mieć zarówno prawo jazdy jak i kluczyki.

Przykład 2.
Możemy pojechać do sklepu autem lub rowerem.
Stosujemy operator **or**, ponieważ mamy alternatywę, możemy jechać autem lub rowerem.

and
Stosując operator and oba warunki muszą być prawdziwe, aby całe wyrażenie również było prawdziwe.
or
Stosując operator or wyrażenie będzie prawdziwy, gdy choć jeden z warunków będzie prawdziwy
not
Stosując operator not całe wyrażenie logiczne zmienia wartość (True zmienia się na False i odwrotnie)

Przykład: Suma kątów wewnętrznych w wieloboku

Przykład wyświetli sumę kątów wewnętrznych w wieloboku o podanej ilości boków.
Sprawdzamy, czy wprowadzona wartość jest liczbą oraz upewniamy się, że wielobok ma co najmniej 3 boki.
Sumę kątów określa się za pomocą wzoru: (n-2) *180

```
n = input("Podaj ilość boków: ")
if(n.isnumeric() and int(n) >= 3):
    suma_katow = (int(n)-2)*180
    print(f"Suma kątów w {n}-boku wynosi: {suma_katow}")
else:
    print("Wprowadzono złą wartość")
```

Interpreter stara się wykonać kod w najszybszy sposób.
Rozpatrując warunki w każdej instrukcji następują pewne procesy przyspieszające analizę kodu.
W przypadku użycia operatora logicznego and (w którym wszystkie warunki muszą być prawdziwe) przy napotkaniu pierwszego warunku fałszywego, całe wyrażenie od razu traktowane jest jako fałszywe i sprawdzanie reszty warunków też zostaje pominięte.
W przypadku operatora or przy napotkaniu pierwszego warunku prawdziwego, całe wyrażenie jest prawdziwe, więc sprawdzanie reszty warunków też zostaje pominięte.

Sprawdzenie wystąpienia tekstu

```
slowo = "anko"
zdanie = "Siemanko gadziny"

# wyszukaj ciąg znaków w zdaniu
if slowo in zdanie:
    print("ok")


# sprawdź brak ciągu znaków w zdaniu
if slowo not in zdanie:
    print("ok, nie ma takiego ciągu w zdaniu")


# wyszukaj ciąg znaków w zdaniu bez rozróżniania małych i
wielkich liter
if slowo.lower() in zdanie.lower():
    print("ok")


# wyszukaj pełne słowa w zdaniu
if slowo in zdanie.split():
    print("ok")
else:
    print("Nie znaleziono")
```

Przykład: Działanie matematyczne

Przykład programu, wyświetlającego działanie dodawania z losowymi liczbami. Jeśli użytkownik poda prawidłową odpowiedź, wyświetli się komunikat Brawo, w przeciwnym razie wyświetli się komunikat Błąd.

```
import random
l1 = random.randrange(1,11,1)
l2 = random.randrange(1,11,1)
suma = l1 + l2
tresc_zadania = f"{l1} + {l2} = "
odpowiedz = input(tresc_zadania)
if(int(odpowiedz) == suma):
    print("Brawo")
else:
    print("Błąd")
```

Pamiętaj, że należy porównywać wartości tego samego typu. Dlatego w przykładzie powyżej castujemy odpowiedź na typ int, ponieważ suma jest wartością liczbową. Moglibyśmy też zrobić odwrotnie, czyli zamienić sumę na wartość tekstową, co w tym przypadku, byłoby nawet lepszym rozwiązaniem, ponieważ uchroniłoby to program przed wprowadzeniem przez użytkownika odpowiedzi innej niż liczbowa.

Funkcje

Funkcja

Funkcja to taki blok kodu, który odpowiada za wykonanie pewnej czynności. Uważa się, że funkcja powinna być bardzo krótka i rozwiązywać 1 konkretny problem.
Najpierw definiujemy funkcję, czyli piszemy słowo kluczowe def, a następnie podajemy nazwę funkcji.
Przechodzimy do wnętrza bloku funkcji i dodajemy kolejne instrukcje, które ta funkcja ma wykonywać.
Samo zdefiniowanie funkcji nie powoduje jej wywołania, tzn., że po jej stworzeniu ona sama się nie wykona. Interpreter ją rozpozna, ale nic z nią nie zrobi. W związku z powyższym musimy ją wywołać podając jej nazwę i nawias okrągły. Zobaczmy jak to działa na przykładzie funkcji, która wyświetla powitanie w konsoli.

```
# definicja funkcji
def wyswietl_powitanie():
    print("Hellowina")

# wywołanie funkcji
wyswietl_powitanie()
```

Patrząc na utworzoną funkcję, możecie powiedzieć,
że zwykłe wyświetlenie wyrazu "Hellowina" moglibyśmy uzyskać poprzez jedną linię kodu, a nie 3!!! Zatem, po co, to wszystko?
O ile nasz program jest na tyle krótki i prosty, że tą wiadomość wyświetlamy tylko raz lub 2 to rzeczywiście,
tworzenie osobnej funkcji nie ma sensu.
W bardziej rozbudowanych aplikacjach będziemy się ciągle spotykać z sytuacjami, gdzie zajdzie potrzeba wykorzystywania tych samych instrukcji z różnych miejsc w kodzie. Wtedy właśnie funkcje okazują się niezbędne. Wyobraź sobie, że to powitanie używasz w 10 różnych miejscach w kodzie, więc bez tworzenia funkcji musiałbyś 10-krotnie wstawić linię kodu:
print("Hellowina"). Po tygodniu jednak stwierdzasz, że ten wyraz nie pasuje, więc chcesz go zmienić na inny. Musisz to zrobić w 10 różnych miejscach. Myślisz, że łatwo zapamiętać te miejsca? I don't think so :) Dzięki funkcji, zmieniasz ten wyraz tylko raz i wszystkie miejsca w których ją wywołujesz dostosowują się automatycznie. I o to właśnie chodzi :)

```
# definicja funkcji
def wyswietl_powitanie():
    print("Hellowina")

# wywołanie funkcji
wyswietl_powitanie()
```

Funkcja z parametrem

Funkcje stają się jeszcze bardziej uniwersalne, kiedy dołączymy do nich parametry.
Jako przykład stworzymy funkcję, która będzie wyświetlać, podaną jako argument, wiadomość.
Każde wywołanie funkcji, z innym argumentem, daje inny wynik w konsoli. Niby funkcja ta sama, a jednak staje się bardziej uniwersalna.
Podczas definiowania funkcji podajemy parametry, a przy wywołaniu funkcji podajemy argumenty.

```
def wyswietl_wiadomosc(trescWiadomosci):
    print(trescWiadomosci)

wyswietl_wiadomosc("Hejka")
wyswietl_wiadomosc("Dzień dobry")
wyswietl_wiadomosc("Cześć")
```

Funkcja z parametrem domyślnym

Funkcja może również przyjmować domyślny parametr, który zostanie użyty jeśli nie podamy argumentu przy jej wywołaniu.

```
def wyswietl_wiadomosc(trescWiadomosci = "Hello"):
  print(trescWiadomosci)

wyswietl_wiadomosc()
```

W przypadku użycia parametrów domyślnych, muszą być one zadeklarowane po parametrach wymaganych.

```
def wyswietl_wiadomosc(imie, trescWiadomosci = "Hello"):
  print(trescWiadomosci, imie)

wyswietl_wiadomosc("Marian")
wyswietl_wiadomosc("Ania", "Cześć")
```

Funkcja z return

Funkcje mogą zwracać wartości, które można wykorzystać wewnątrz innych funkcji lub przypisać je do zmiennych.
Wewnątrz funkcji należy podać słowo kluczowe return, które określa co powinno być zwrócone.
Instrukcja return kończy działanie całej funkcji i powoduje wyjście do bloku zewnętrznego.

```
def zwroc_pole_trojkata(podstawa, wysokosc):
    return podstawa*wysokosc*0.5

print("pole trojkata wynosi:" ,zwroc_pole_trojkata(4,6))
```

Przykład zwracający losowo rzut monetą: orzeł-reszka

```
import random

def rzuc_moneta():
    losowa = random.randrange(0,2)
    if(losowa == 1):
        return "orzeł"
    else:
        return "reszka"

print(rzuc_moneta())
```

Zmienne globalne i lokalne

W programowaniu mamy do czynienia z zakresem dostępności, tzw. scope.
Zmienne tworzone wewnątrz danego bloku kodu (np. funkcji) mają zasięg lokalny i żyją (są dostępne) tylko w tym bloku. Nie można się do nich dostać z zewnątrz.
Zmienne tworzone poza funkcją mają zasięg globalny i są dostępne wewnątrz wszystkich funkcji w danym skrypcie.
Możemy tworzyć zmienne globalne i lokalne o takich samych nazwach, ale należy tego unikać.
Chcąc, wewnątrz funkcji, skorzystać ze zmiennej globalnej, należy to wcześniej określić za pomocą słowa kluczowego global.
Poniżej przykład nadpisania zmiennej globalnej z wnętrza funkcji:

```
ilosc_punktow = 0

def zmien_liczbe_punktow():
    global ilosc_punktow
    ilosc_punktow = 10

zmien_liczbe_punktow()

print(ilosc_punktow)
```

Pętle

Pętla for

Komputery zostały stworzone po to, by ułatwiać i usprawniać życie ludziom.
Pętla służy do tego, by wykonywać podobne operacje w szybki i łatwy sposób. Wyobraź sobie w jaki sposób napisałbyś program, który wyświetla 1000 kolejnych liczb od 1 do 1000?

```
for i in range(1, 1001):
    print(i)
```

2 linie kodu sprawiają, że wyświetlimy 1000 kolejnych liczb.
Zauważ, że podobnie jak z losową liczbą, tutaj też wartość maksymalną zwiększamy o 1.
i to zmienna lokalna, tzw. iteracyjna (dlatego nazywa się i :).
Z każdym przejściem pętli wartość tej zmiennej i zmienia się. W tym przypadku zwiększa się o 1, aż dojdzie do 1000, wtedy pętla zakończy działanie.
W pierwszym cyklu pętli została jej przypisana wartość 1 bo taka wartość jest podana w nawiasie. Kompilator sprawdza czy zachodzi tutaj warunek prawdziwy, czyli czy i (aktualnie 1) jest w przedziale 1 : 1000, jeśli tak, to przechodzi do wnętrza bloku i drukuje wartość i w konsoli.
Zaczyna się kolejny cykl (iteracja) pętli. Teraz i przyjmuje wartość

o 1 większą, czyli 2. Ponownie dzieje się to samo, sprawdzane jest czy dwójka jest w zakresie i jeśli tak jest, to kod przechodzi do bloku i drukuje dwójkę, itd..
Przy ostatnim cyklu i zwiększy się do 1001, ale nie będzie już w zakresie (bo wartość maksymalna jest zawsze zwiększana o 1) i wtedy warunek staje się fałszywy, w związku z tym, pętla przestaje działać, a interpretator wychodzi z tego bloku kodu i przechodzi do kolejnych instrukcji w programie.
W tym typie pętli wartość **i** zwiększa się automatycznie o 1, w innych pętlach sami możemy decydować jak ta wartość ma się zmieniać.

Przykład użycia pętli for do wypisania liczb od 1 do 10 w jednym wierszu, oddzielonych przecinkiem.

```
liczby = ""
for i in range(1, 11):
    liczby += str(i) + ", "
print(liczby[:-2])
```

Pętla while

Pętli while możemy użyć wtedy, gdy nie wiemy, ile razy ta pętla powinna się wykonać. Za jej pomocą, możemy wyświetlać działanie lub komunikat, dopóki użytkownik nie wpisze prawidłowej odpowiedzi.
Krótko mówiąc, używamy jej, by wykonywać ciągle jakąś instrukcję, dopóki podany warunek nie będzie prawdziwy.
To jest jej główne zadanie, ale możemy za jej pomocą, robić to, co robimy za pomocą pętli for :)
Pętla while wymaga, aby zmienna iteracyjna została zadeklarowana przed blokiem pętli, a w bloku pętli będziemy tą zmienną samodzielnie zwiększać lub zmniejszać.

Przykład użycia pętli while do wypisania liczb od 1 do 100

```
i = 1
while (i < 101):
    print(i)
    i += 1
```

Przykład: Podanie hasła do skutku

Przykład programu, który wyświetla komunikat o podanie hasła, dopóki prawidłowe hasło nie zostanie wpisane.

```
# powtarzaj komunikat, aż zostanie podane prawidłowe hasło
haslo = "1234"
podane_haslo = ""
while(haslo != podane_haslo):
    podane_haslo = input("Podaj hasło: ")
print("Hasło prawidłowe")
```

Przykład programu, który wyświetla liczby parzyste od 0 do 50, za pomocą obu pętli

```
# wyświetla liczby parzyste od 0 do 50
liczba = 0
while(liczba < 51):
    print(liczba)
    liczba += 2


# wyświetla liczby parzyste od 0 do 50
for i in range(0,51,2):
    print(i)
```

Przykład programu, który wyświetla liczby malejące od 10 do 1

```
i = 10
while(i>=1):
    print(i)
    i-=1


for i in range(10,0,-1):
    print(i)
```

Break, continue

Instrukcja break powoduje przerwanie pętli, a instrukcja continue pozwala pominąć wynik pętli na podstawie podanych warunków.

```
# Przykład programu, który będzie losował 10 liczb od 1 do 10 i jeżeli zostanie wylosowana 1, przerwie działanie programu.
import random
for i in range(1,11):
    los = random.randrange(1,11,1)
    if los == 1:
        break
    print(los)


# Przykład programu, który wyświetla liczby od 1 do 10 poza liczbą 5
for i in range(1,11):
    if i == 5:
        continue
    print(i)
```

Przykład: Losowe działanie matematyczne

Przykład funkcji, która generuje losowe działanie matematyczne. Wielokrotne wywołanie funkcji za pomocą pętli.

```
import random

def stworz_dzialanie():
    liczba1 = random.randrange(1,20)
    liczba2 = random.randrange(6,30)
    return f"{liczba1} + {liczba2} = "

for i in range(1,11):
    print(stworz_dzialanie())
```

Przykład: Zadanie matematyczne z odpowiedzią

Przykład, który wyświetla losowe działanie matematyczne do czasu, aż zostanie udzielona prawidłowa odpowiedź.

```
import random
liczba1 = random.randrange(1,20,1)
liczba2 = random.randrange(1,20,1)
suma = liczba1 + liczba2
zadanie = f"{liczba1} + {liczba2} = "
odp = input(zadanie)
while(odp != str(suma)):
    print("Odpowiedź niepoprawna, spróbuj ponownie")
    odp = input(zadanie)
print("Brawo Ty")
```

Przykład: Wizualizacja gwiazdkowych kwadratów

```
q = 6
for i in range(0,q):
    line = '*  ' * q
    print(line)
```

```
*  *  *  *  *  *
*  *  *  *  *  *
*  *  *  *  *  *
*  *  *  *  *  *
*  *  *  *  *  *
*  *  *  *  *  *
```

Przykład: Gwiazdkowa choinka

```
q = 10
for i in range(1,q+1):
    digits = q - i
    space = " " * digits
    line = space + '* '*i
    print(line)
```

```
          *
         * *
        * * *
       * * * *
      * * * * *
     * * * * * *
    * * * * * * *
   * * * * * * * *
  * * * * * * * * *
 * * * * * * * * * *
```

Wyjątki try: except:

Wyjątki pozwalają nam zabezpieczyć się przed nieprawidłowym działaniem programu (np. wtedy, gdy zerwie się połączenie z internetem lub ktoś będzie chciał wprowadzić tekst zamiast liczby, itp).
Najłatwiej zrobić to za pomocą bloków try i except. W bloku try: próbujemy wykonać kod, a jeśli się to nie uda, uruchomi się blok except:

```
try:
    print("try this code")
except:
    print("execute if try block failed")
```

Przykład: Cena po obniżce

Przykład programu, który pobierze cenę towaru, wartość procentową obniżki i zwróci cenę po obniżce.

```
cena = input("Podaj cenę przed obniżką: ")
obnizka = input("Wartość procentowa obniżki: ")
try :
    cena = float(cena)
    obnizka = float(obnizka)
    cena_konc = cena - (cena * obnizka / 100)
    print("Cena po obniżce = ", cena_konc)
except:
    print("Nie można obliczyć ceny")
```

Przykład: Test matematyczny z punktami

Przykład programu, który wyświetli pewną ilość zadań Po podaniu odpowiedzi, przechodzimy do kolejnego zadania. Przy udzieleniu poprawnej odpowiedzi dostajemy punkt. Po udzieleniu wszystkich odpowiedzi, na koniec testu pokazuje nam się wynik poprawnie udzielonych odpowiedzi.

```
import random
liczba_punktow = 0
ilosc_zadan = 2

def tworz_zadanie():
    liczba1 = random.randrange(1,21)
    liczba2 = random.randrange(1,21)
    tresc_zadania = f"{liczba1} + {liczba2} = "
    odpowiedz = input(tresc_zadania)
    try:
        odpowiedz = int(odpowiedz)
        if(odpowiedz == liczba1 + liczba2):
            global liczba_punktow
            liczba_punktow += 1
    except:
        print("Wynik powinien być podany w postaci liczby")

for i in range(0, ilosc_zadan):
    tworz_zadanie()

print(f"Poprawnych odpowiedzi: {liczba_punktow} /
{ilosc_zadan}")
```

Przykład: Gra - Zgadnij liczbę

Przykład gry , w której będziemy starali się odgadnąć losowo wybraną liczbę z przedziału od 1 do 100.
Program losuje liczbę, a następnie prosi użytkownika o wpisanie liczby. Po udzieleniu niepoprawnej odpowiedzi program podpowie czy zgadywana liczba jest większa od podanej czy mniejsza i ponownie poprosi o wpisanie liczby.
Po udzieleniu poprawnej odpowiedzi, gra się kończy, a użytkownik dostaje komunikat ile razy udzielił złej odpowiedzi.

Przykład: Gra - Zgadnij liczbę

```
import random
szukana_liczba = random.randrange(1,101)
odpowiedz = 0
ilosc_prob = 0

def podaj_odpowiedz():
    global odpowiedz
    if(odpowiedz == 0):
        trescPytania = "Podaj liczbę od 1 do 100: "
    elif(odpowiedz > szukana_liczba):
        trescPytania = "Podaj mniejszą liczbę: "
    else:
        trescPytania = "Podaj większą liczbę: "
    try:
        odpowiedz = int(input(trescPytania))
    except:
        print("Podaj wartość numeryczną!")
    global ilosc_prob
    ilosc_prob += 1

while(odpowiedz != szukana_liczba):
    podaj_odpowiedz()

print(f"Brawo! Zgadłeś! Ilość prób: {ilosc_prob}")
```

Struktury danych

Listy

Listy służą do tworzenia kolekcji, uporządkowanego zbioru danych.
Listy tworzymy przy użyciu nawiasów kwadratowych [ ].
Elementy listy posiadają numery porządkowe (indeksy), a ich zliczanie zaczyna się od 0. Chcąc wyświetlić dany element listy używa się jej nazwy z numerem indeksu podanym wewnątrz nawiasu kwadratowego.

```
# tworzenie listy z imionami uczniów i wyświetlenie
drugiego elementu (Ania)
uczniowie = ["Sara", "Ania", "Maciek", "Zenek"]
print(uczniowie[1])


#Pobranie losowego elementu z listy
import random
uczniowie = ["Sara", "Ania", "Maciek", "Zenek"]
print(random.choice(uczniowie))
```

Listy z pętlą

Zawartość listy możemy z łatwością wypisać za pomocą pętli for. W tym przypadku, zmienna **uczen** nie będzie liczbą porządkową, a danym elementem tablicy. W takiej wersji pętli, pojedynczy element nazywa się liczbą pojedynczą nazwy listy.

```
# wypisanie elementów listy uczniowie
uczniowie = ["Sara", "Ania", "Maciek", "Zenek"]
for uczen in uczniowie:
    print(uczen)

# lub w ten sposób z dodatkiem liczby porządkowej
uczniowie = ["Sara", "Ania", "Maciek", "Zenek"]
for i,uczen in enumerate(uczniowie):
    print(i+1, uczen)


# wypisanie elementów listy uczniowie za pomocą indeksu
uczniowie = ["Sara", "Ania", "Maciek", "Zenek"]
for i in range(0,len(uczniowie)):
    print(uczniowie[i])


# wypisanie elementów listy liczbyTotka
liczby_totka = [12, 7, 29, 2, 41, 36]
for liczba in liczby_totka:
    print(liczba)
```

Iterowanie po wielu listach jednocześnie

Iterowanie po wielu listach tej samej długości za pomocą metody zip()

```
liczby = [1, 2, 3, 4]
potegi = [1, 4, 9, 16]

for l, p in zip(liczby, potegi):
    print(l, p)
```

Metody na listach

```
import random


kolekcja = 24,45,67
lista = ["python", "C++", "JavaScript"]


lista.append("C#")  #dodaje element na końcu listy
lista.pop()         #usuwa element o podanym w nawiasie
                    indeksie (lub ostatni, jeżeli brak arg.)
len(lista)          #zwraca liczbę elementów w liście
max(lista)          #zwraca największą wartość na liście
min(lista)          #zwraca najmniejszą wartość na liście
sum(lista)          #zwraca sumę elementów na liście
lista.sort()        #sortujemy rosnąco listę
lista.reverse()     #odwraca kolejność elementów na liście
list(kolekcja)      #zamiana kolekcji na listę
random.shuffle(lista)   #zmienia losowo kolejność elementów
listy
```

Sortowanie listy

Zawartość listy możemy sortować, np: liczby rosnąco, wartości tekstowo alfabetycznie, itp.
Sortujemy odnosząc się do nazwy listy, a następnie dopisując po kropce nazwę metody **sort()**

```
# sortowanie i wypisanie elementów listy uczniowie
oddzielonych przecinkiem
uczniowie = ["Sara", "Ania", "Maciek", "Zenek"]
uczniowie.sort()
lista_uczniow = ""
for uczen in uczniowie:
   lista_uczniow += uczen + ", "
print(lista_uczniow[:-2])
```

Jeżeli chcemy posortować tablicę w odwrotnym kierunku to wewnątrz nawiasu, jako argument metody sort(), możemy dodać opcję reverse = True, np:
uczniowie.sort(reverse = True)
lub po metodzie sort() możemy użyć metody reverse()

Przypisanie elementów listy do zmiennych

Python pozwala przypisać elementy listy do osobnych zmiennych:

```
uczniowie = ["Ania", "Maciek", "Antoś"]
x, y, z = uczniowie
print(x)
print(y)
print(z)
```

Mapowanie

Czasami zajdzie potrzeba zmienić tablicę danego typu na inny. W takim przypadku należy posłużyć się metodą mapowania - map().
W przykładzie poniżej zamienimy tablicę typu string na int.

```
oceny = ['4','5','2','1','5','3']
oceny = list(map(int,oceny))
print(oceny)
```

Przykład: Czy można zbudować trójkąt?

Przykład programu, który sprawdza czy z podanych długości można zbudować trójkąt.
Aby zbudować trójkąt, suma długości dwóch krótszych boków musi być większa od najdłuższego boku.

```
def pobierz_boki_trojkata():
    boki_trojkata = []
    try:
        for i in range(1,4):
            bok = float(input(f"Bok nr {i}: "))
            boki_trojkata.append(bok)

        boki_trojkata.sort()
        if(boki_trojkata[0] > 0):
            sprawdz_czy_mozna_zbudowac_trojkat(boki_trojkata)
        else:
            print("Boki muszą być większe od 0")
    except:
        print("Podaj długości boków w postaci liczby")

def sprawdz_czy_mozna_zbudowac_trojkat(boki):
    if(boki[2] < (boki[0] + boki[1])):
        print("Można zbudować trójkąt o podanych bokach.")
        print(f"Obwód trójkąta wynosi: {sum(boki)}")
    else:
        print("Nie można zbudować trójkąta")

pobierz_boki_trojkata()
```

Przykład: Losowanie liczb totolotka

Program, który losuje 6 liczb z 49 i wypisuje je w kolejności rosnącej. Losowane liczby nie mogą się powtarzać.

```
import random

#Tworzymy zestaw pomieszanych liczb
def generuj_zestaw_liczb_mix(max):
    wszystkie_liczby = []
    for i in range(0, max):
        wszystkie_liczby.append(i+1)
    random.shuffle(wszystkie_liczby)
    return wszystkie_liczby

# Zwracamy określoną ilość losowych liczb,
# wylosowaną liczbę dodajemy do nowej tablicy wygranych i
usuwamy ją z tablicy wszystkich liczb by uniknąć powtórek
def losuj_wygrane_liczby(zestaw_liczb, ilosc_liczb):
    wygrane_liczby = []
    for i in range(0, ilosc_liczb):
        wybrana_liczba = random.randrange(0, len(zestaw_liczb))
        wygrane_liczby.append(zestaw_liczb[wybrana_liczba])
        zestaw_liczb.pop(wybrana_liczba)
    wygrane_liczby.sort()
    return wygrane_liczby

zestaw_wszystkich_liczb = generuj_zestaw_liczb_mix(49)
wygrane_liczby = losuj_wygrane_liczby(zestaw_wszystkich_liczb,
6)
```

```
print(wygrane_liczby)
```

Zadanie: Różnica między min a max

Wyświetl liczby rosnąco oraz wyświetl jaka jest różnica między najmniejszą i największą wartością w tablicy:

```
liczby = [8,2,-23,9,56,-29,44,3,-76,12,89,-65,57]
```

Słownik (Dictionary)

Słownik (dictionary) pozwala tworzyć tablice asocjacyjne za pomocą pary - key : value.
Dla każdego klucza, można przypisać konkretną wartość.
Słowniki są uporządkowane, zmienialne i nie przyjmują zduplikowanych kluczy.

```
dane_osobowe = {
   "imie" : "Ania",
   "wiek" : 36
}

#wypisanie pojedynczych wartości
print(dane_osobowe["imie"])
print(dane_osobowe["wiek"])


#zmiana wartości w słowniku lub dodanie nowej pozycji
dane_osobowe["wiek"] = 30
print(dane_osobowe["wiek"])


Wypisanie kluczy / wartości danego słownika:
print(list(dane_osobowe.keys()))
print(list(dane_osobowe.values()))


#Pobranie losowego klucza dictionary
```

```python
import random
dane_osobowe = {
   "imie" : "Ania",
   "wiek" : 36
}

print(random.choice(
    list(dane_osobowe.keys()))
)
```

Krotka (Tuple)

Tuple pozwala zapisać wiele różnych wartości w jednej zmiennej. Tuple są niemutowalne, czyli niezmienne. Raz ustawione elementy nie mogą być później nadpisane.

```
tupelek = 3,22,21
print(tupelek[0])
print(tupelek[1])
```

```
tuple1 = ("Mateusz", "Legnica", 120)
print(tuple1)
print(tuple1[0]) #element z indeksem 0
print(tuple1[1:3]) #element o indeksie 1 i 2
```

```
# zwracanie tupli z funkcji
def zwroc_tuple():
    return "Mateusz", "Legnica", 120

print(zwroc_tuple()[2])
```

Set

Set to zbiór w którym każdy element jest unikatowy. Elementy zbioru są niemutowalne, nie indeksowane i nie uporządkowane, przez co nigdy nie ma pewności w jakiej kolejności zostaną pobrane.

Set tworzy się za pomocą nawiasów klamrowych.

```
set_owocowy = {"apple", "banana", "cherry"}
print(set_owocowy)
```

Tworzenie pustego seta

```
pusty_set = set()
```

Nie można zmieniać wartości elementów seta, ale można je usuwać i dodawać nowe.

```
set_owocowy.add("pear")
set_owocowy.discard("pear")
```

Set może zawierać różne typy danych, np:

```
set1 = {"abc", 34, True, 40, "male"}
```

Ponieważ set może zawierać tylko unikalne elementy to można go użyć do pozbycia się duplikatów z listy:

```
lista = [1,2,3,1,2,1]
my_set2 = set(lista)
print(my_set2)
```

Operacje na wielu zbiorach:

```
uczniowie = {"Ania", "Kasia", "Marek", "Zenek"}
kursanci = {"Zosia", "Kasia", "Marek"}
print(uczniowie - kursanci) #różnica zbiorów
print(uczniowie & kursanci) #część wspólna zbiorów
print(uczniowie | kursanci) #suma zbiorów
```

Set / lista / tuple

```
# przykłady utworzenia 3 różnych struktur
set_owocowy = {"apple", "banana", "cherry", 12}
lista_owocowa = ["apple", "banana", "cherry", 12]
tuple_owocowy = ("apple", "banana", "cherry", 12)
```

Dodatkowe paczki z pip

Możemy w łatwy sposób zwiększyć możliwości programu za pomocą pobranych z internetu bibliotek w postaci paczek (packages). Obszerna baza dostępnych paczek znajduje się na stronie https://pypi.org/ ale wiele ciekawych projektów można również znaleźć na githubie.
Paczki takie można pobrać i zainstalować manualnie lub skorzystać z tzw. pip, czyli python managera. W najnowszych wersjach pythona, pip jest instalowany automatycznie wraz z interpreterem. Możemy sprawdzić czy znajduje się on na naszym komputerze i jaką ma wersję uruchamiając w konsoli instrukcję:

```
pip --version
```

powinien pokazać się numer wersji lub poniższy komunikat, który oznacza, że manager pip nie jest zainstalowany.
'pip' is not recognized as an internal or external command, operable program or batch file.

Pythona można zainstalować pobierając plik ze strony https://www.python.org/downloads/

Wersja wideo, w jaki sposób zainstalować python: https://youtu.be/eeM-bVEWA_A

Wersja wideo, jak zainstalować przez pip i użyć paczki pyinstaller do konwersji skryptów .py do pliku wykonawczego .exe https://youtu.be/rcuxn6Xizbc

Aby zainstalować paczkę za pomocą managera pip należy wpisać pip install nazwaPaczki, w tym przypadku pokażemy jak zainstalować paczkę pyinstaller do zamiany skryptu .py na .exe

```
pip install pyinstaller
```

Zamiana skryptu .py na .exe w jednym pliku następuje po wpisaniu instrukcji:

```
pyinstaller --onefile main.py
```

gdzie main.py to nazwa pliku, która może być inna

Tworzenie programu jednoplikowego - - onefile jest źle postrzegane i traktowane jako wirus. Dlatego chcąc udostępniać tworzone pliki lepiej zrobić to w ten sposób:
dla wersji konsolowych:

```
pyinstaller -c --console --nowindowed main.py
```

lub dla wersji z gui

```
pyinstaller -w --windowed --noconsole main.py
```

Data i czas

Zarządzanie datami w python wymaga pobrania modułu datetime.

```
import datetime
```

Tworzymy zmienną z aktualną datą i godziną:

```
aktualna_data_godzina = datetime.datetime.now()
print(aktualna_data_godzina)
```

Tworzenie daty

W przypadku, gdy chcemy stworzyć datę, wywołujemy konstruktor datetime() i podajemy parametry opisujące rok, miesiąc i dzień, godzinę, minutę, itd.. .

Poniżej przykład stworzenia daty: 21 lipiec 2020, 18:55

```
import datetime
data_urodzenia = datetime.datetime(2020, 7, 21, 18, 55)
```

Formatowanie daty

Utworzony obiekt daty zawiera wiele informacji, które możemy przefiltrować i sformatować za pomocą odpowiednich metod i argumentów, których część znajduje się w tabeli poniżej, np:

```
print(aktualna_data_godzina.strftime("%x"))
print(data_urodzenia.strftime("%d %B %Y, %H:%M"))
```

Tabela formatów daty i czasu

Argument	Opis	Przykład
%a	Dzień tygodnia, krótka wersja	Wed
%A	Dzień tygodnia, pełna wersja	Wednesday
%w	Dzień tygodnia jako numer 0-6, 0-Niedziela	3
%d	Dzień miesiąca 01-31	31
%b	Nazwa miesiąca, krótka wersja	Dec
%B	Nazwa miesiąca, długa wersja	December
%m	Miesiąc jako numer 01-12	12
%y	Rok, krótka wersja	18
%Y	Rok, pełna wersja	2018
%H	Godzina 00-23	17
%I	Godzina 00-12	5
%p	AM/PM	PM
%M	Minuta 00-59	41
%S	Sekunda 00-59	8
%j	Numer dnia w roku 001-366	365
%U	Numer tygodnia w roku, Niedziela pierwszym dniem, 00-53	52
%W	Numer tygodnia w roku, Poniedziałek pierwszym dniem, 00-53	52
%C	Wiek	20
%x	Lokalna wersja daty	12/31/18
%X	Lokalna wersja czasu	17:41:00

Klasa

Tworzenie klasy i obiektu

Klasa jest instrukcją tworzenia obiektów. Nazwy klas powinny być w postaci rzeczownika i rozpoczynać się z wielkiej litery.
Wewnątrz klasy znajduje się konstruktor (init), który wywołuje się automatycznie w chwili tworzenia nowego obiektu klasy.
W przykładzie obok stworzymy klasę osoby, która może zawierać takie dane jak imię i wiek, a następnie stworzymy obiekt tej klasy.
Słowo self, oznacza aktualną instancję klasy. Należy je dodawać jako pierwszy argument konstruktora.
Zmienna (name, age) jest również określana jako pole.

```
class Osoba:
    def __init__(self, imie, wiek):
        self.imie = imie
        self.wiek = wiek

# tworzenie obiektu klasy Osoba z danymi
p1 = Osoba("Marian", 36)

# wyświetlanie danych osoby p1
print(p1.imie)
print(p1.wiek)
```

Metody

Klasy, poza polami, mogą posiadać jeszcze metody, które można wywoływać na tworzonych obiektach.
W przykładzie, stworzymy metodę, wyświetlającą dane osobowe.

```
class Osoba:
    def __init__(self, imie, wiek):
        self.imie = imie
        self.wiek = wiek

    def wyswietl_dane_osobowe(self):
        print(f"Imię:{self.imie}, wiek:{self.wiek}")


p1 = Osoba("Marian", 36)
p1.wyswietl_dane_osobowe()
```

Metody statyczne

Możemy również tworzyć metody statyczne, które przynależą do całej klasy, a nie poszczególnych obiektów.
W przykładzie poniżej metoda statyczna wyświetla ilość wszystkich utworzonych osób.

```
class Osoba:
    liczba_osob = 0

    def __init__(self, imie, wiek):
        self.imie = imie
        self.wiek = wiek
        Osoba.liczba_osob += 1

    def wyswietl_dane_osobowe(self):
        print(f"Imię:{self.imie}, wiek:{self.wiek}")

    @staticmethod
    def wyswietl_ilosc_osob():
        print("Liczba osob: ", Osoba.liczba_osob)

p1 = Osoba("Marian", 36)
p2 = Osoba("Anna", 32)
p3 = Osoba("Kasia", 22)
p1.wyswietl_dane_osobowe()
p2.wyswietl_dane_osobowe()
p3.wyswietl_dane_osobowe()

Osoba.wyswietl_ilosc_osob()
```

Praca z plikami i danymi

Tworzenie / zapisywanie / usuwanie pliku

Tworzone programy często pracują z plikami, np. tekstowymi. Otwieranie plików, zarówno do odczytu jak i zapisu, przebiega za pomocą metody open(), która przyjmuje pewne argumenty: ścieżkę dostępu do pliku i sposób pracy na pliku.

Możemy otworzyć plik:
r - tylko do odczytu (read), w tym przypadku plik musi istnieć i znajdować się w podanej lokalizacji
a - dodanie zawartości (append), w tym trybie można dodać coś do istniejącej zawartości pliku. W przypadku, jeśli plik nie istnieje w podanej lokalizacji, zostanie utworzony od nowa.
w - zapisać coś od nowa(write). Plik zostanie wyczyszczony i zapisany nową wartością. W przypadku, jeśli plik nie istnieje w podanej lokalizacji, zostanie utworzony od nowa.

Możemy też sprecyzować czy plik jest zwykłym plikiem tekstowym (t) czy binarnym, np. grafiką (b).
Pliki, na których pracujemy, powinny być zamknięte, kiedy skończymy operacje odczytywania / zapisywania za pomocą metody close()

Przykład dodania treści na koniec pliku o nazwie demo.txt

```
f = open("demo.txt", "a")
f.write("Maasdian")
f.write("\n")
f.close()
```

Przykład wyczyszczenia i zapisu aktualnej daty i godziny do pliku o nazwie demo.txt w znacznie lepszy sposób za pomocą instrukcji with dzięki czemu nie trzeba zamykać pliku

```
import datetime
with open("demo.txt",'w',encoding = 'utf-8') as f:
    d = datetime.datetime.now()
    f.write(d.strftime("%d %m %Y, %H:%M:%S"))
```

Za pomocą modułu os możemy zarządzać plikami.

```
# usuń plik jeśli istnieje
import os
if os.path.exists("demo.txt"):
  os.remove("demo.txt")
else:
  print("Plik nie istnieje")
```

Odczytywanie plików

Otwierając plik do odczytu(r) warto sprawdzić czy w ogólne znajduje się on w podanej lokalizacji za pomocą modułu os i metody exists()

```
# Odczytywanie całej zawartości pliku tekstowego:
import os
if os.path.exists("demo.txt"):
    with open ("demo.txt", "r") as f:
        print(f.read())
```

```
# Odczytywanie jednej linii
import os
if os.path.exists("demo.txt"):
    with open ("demo.txt", "r") as f:
        print(f.readline())
```

```
# Odczytywanie wszystkich linii w pętli
import os
if os.path.exists("demo.txt"):
    with open ("demo.txt", "r") as f:
        for line in f:
            print(line)
```

```python
# Wyświetlanie losowej linii z pliku tekstowego
import os
import random

if os.path.exists("demo.txt"):
    with open ("demo.txt", "r") as f:
        lista = []
        for line in f:
            lista.append(line)
        losowa = random.randrange(0, len(lista))
        print(lista[losowa])
```

JSON

JSON to format zapisu i wymiany danych. Zazwyczaj w tym formacie będziemy pobierać dane ze źródeł zewnętrznych, co umożliwia interakcję programów pisanych w różnych językach lub używanych na różnych platformach.

Poniżej przykład konwersji słownika (json.dumps()) na format json i zapisu do pliku txt oraz konwersji json z powrotem na obiekt (json.loads()).

```
import json

with open("plik.txt", "w") as plik:
    aDict = {"a":54, "b":87}
    jsonString = json.dumps(aDict, indent = 4)
    plik.write(jsonString)



with open("plik.txt", "r") as plik:
    aDict = json.loads(plik.read())
    print(aDict)
```

Formatowanie obiektów na JSON

Przykład pokazuje jak zamienić prosty obiekt na format json i załadowanie go z powrotem na obiekt o danym typie.

```
import json

class Osoba:
    def __init__(self, imie, wiek):
        self.imie = imie
        self.wiek = wiek

p1 = Osoba("Marian", 36)

with open("plik.json", "w") as plik:
    jsonString = json.dumps(p1.__dict__, indent = 4)
    plik.write(jsonString)

with open("plik.json", "r") as plik:
    osoba = Osoba(**json.loads(plik.read()))
    print(osoba.imie, osoba.wiek)
```

Przykład programu zapisującego high score do pliku json

```
import json
import random

def play_game():
    hscore = load_scores()
    print(f'Highest score: {hscore[1]} by {hscore[0]}')

    player = []
    player.append(input("Your name: "))
    player.append(random.randrange(100000))

    if(player[1] > hscore[1]):
        print(f"Great {player[0]}! You achieved the highest
score ever: {player[1]}")
        save_scores(player)
    else:
        print("Good try, but not this time..")

def save_scores(player):
    json_string = json.dumps(player)
    with open ("players.json", "w") as json_file:
        json_file.write(json_string)

def load_scores():
    with open ("players.json", "r") as json_file:
        file_text = json_file.read()
        try:
            data = json.loads(file_text)
            return data
        except:
            return ["...", 0]

play_game()
```

API requests

API Application Programming Interface to sposób komunikacji i wymiany informacji między aplikacjami. Dzięki interfejsom aplikacje mogą ze sobą współpracować i integrować z innymi systemami.
https://jsonformatter.org/

Pobieranie danych z REST API za pomocą request

Zainstaluj moduł requests: **pip install requests**

```
import requests
import json

def get_json_data():
    parameters = "per_page=3"
    response =
requests.get("https://pikademia.pl/wp-json/wp/v2/posts",par
ameters)
    articles = response.json()
    # content = response.content
    # articles = json.loads(content)
    for article in articles:
        print(article["title"]["rendered"])

get_json_data()
```

Zapytania (requesty) zawsze zwracają informacje odnośnie połączenia.
Możemy je pobrać za pomocą klucza status_code:

```
print(response.status_code)
```

200 — Wszystko przebiegło ok
301 — Serwer przekierowuje do innego punktu (endpointu).
401 — Serwer uważa, że nie posiadasz uprawnień, dzieje się tak, kiedy nie prześlesz odpowiednich danych autentykacyjnych
400 — Złe zapytanie
403 — Zasoby są zakazane lub nie masz uprawnień
404 — Serwer nie odnalazł zasobów

response zwraca również dane o tym, jak informacje zostały wygenerowane i w jakim formacie, za pomocą właściwości **headers**.

```
print(response.headers)
```

Funkcje matematyczne

Podstawowe metody matematyczne

Python posiada wiele metod matematycznych, niektóre z nich są dostępne po dodaniu modułu math

```
import math

# Wybór najmniejszej / największej liczby z podanych
x = min(5, 10, 25, 34, 1, 101)
y = max(5, 10, 25, 34, 1, 101)

# Wartość bezwzględna
x = abs(-7.25)

import math
#sinus wartości kątowej podanej w radianach
sin = round(math.sin(math.pi / 6), 4)

#sinus z x podane w stopniach
sin = round(math.sin(math.radians(30)), 4)

# Wartość liczby pi
import math
x = math.pi
```

```
# Potęga / pierwiastek
import math

x = pow(4, 3)        # 4 do potęgi 3
y = math.sqrt(64)
z = pow(8, 2/3)      # pierwiastek 3 stopnia z 8 do potęgi 2
w = pow (16, 1/4)    # pierwiastek 4 stopnia z 16
```

Ciąg Fibbonacciego

Ciąg Fibbonacciego zaczyna się od 1,1, a następnie każdy kolejny element, jest sumą dwóch poprzednich, tj:
1,1,2,3,5,8,13,21,34,55,...

```
def get_fibbonacci_list(n):
    result = [1, 1]
    for i in range(n-2):
        result.append(result[i] + result[(i + 1)])

    return result

print(get_fibbonacci_list(10))
```

Modulo

Modulo to operator, który zwraca resztę z dzielenia.
Np. w działaniu 12/5, otrzymujemy resztę 2. Zatem modulo 12/5 wynosi 2.
Symbolem modulo jest znak %

```
print(12 % 5)
```

Modulo przydaje się, by sprawdzić czy dana liczba jest parzysta (bo wtedy liczba % 2 == 0) lub czy dana liczba jest podzielna przez inną liczbę.

Przykład: Liczba podzielna przez 3

Przykład programu, który sprawdza czy podana liczba jest podzielna przez 3

```
liczba = int(input("Podaj liczbę całkowitą: "))
if(liczba % 3 == 0):
    print("Liczba jest podzielna przez 3")
else:
   print("Liczba nie jest podzielna przez 3")
```

Metody wartości tekstowych

Zamiana tekstu - replace()

```
# Przykład zamienia kropki na przecinki
txt = "Bułki kosztują 0.99zł, a oranżada 1.29zł."
txt = txt.replace(".", ",")
print(txt)
```

Licznik wystąpień w tekście

```
# Przykład zlicza ilość kropek w tekście
txt = "Bułki kosztują 0.99zł, a oranżada 1.29zł."
dot_counter = txt.count(".")
print(dot_counter)
```

Metoda split()

Służy do dzielenia tekstu wg. określonego wzorca, np. możemy dzielić tekst przy każdym wystąpieniu średnika lub przecinka. Jest to bardzo istotne przy przetwarzaniu informacji zaczerpywanych z plików lub innych źródeł (np. internetu). W wyniku tej funkcji powstaje nam lista zawierająca podzielone elementy, którą następnie można odpowiednio przetworzyć. Np. możemy pobrać jakieś dane z excela i wyłuskać z nich tylko to co potrzebujemy.

W przykładzie mamy kilka dat, oddzielonych średnikiem. Za pomocą metody split oddzielimy daty, a następnie za pomocą pętli wyciągniemy z każdej daty sam rok.

```
tekst = "12-05-2022;22-05-2022;18-06-2022"
rok = tekst.split(';')
for i in range(0,len(rok),1):
    rok[i] = rok[i][-4:]
print(rok)
```

Istnieje wiele innych, użytecznych metod na wartościach typu string, które z łatwością można wyszukać w internecie w razie potrzeby. Wystarczy wpisać np. python string methods, aby znaleźć ich listę z przykładami zastosowania.

Tablica znaków ASCII / UNICODE

Tablica ASCII / Unicode to system kodowania znaków za pomocą liczb. W tabeli przedstawione zostały podstawowe znaki, które można zapisać za pomocą liczby w systemie dziesiętnym, choć znacznie popularniejszy w tym zakresie jest system szesnastkowy (0x)

Przykładowo, znak $ można przedstawić za pomocą liczby 36, a znak wielkiej litery W za pomocą liczby 87.
W postaci liczby szesnastkowej, możemy przedstawić znak © za pomocą zapisu 0x00A9

Chcąc zamienić liczbę na znak użyjemy metody chr(),np:

```
print(chr(0x00A9))
```

lub

```
print(chr(169))
```

Operacja odwrotna, czyli zamiana znaku na liczbę, odbywa się za pomocą metody ord()

```
znak = '$'
print(ord(znak))
```

Dec	Char	Dec	Char	Dec	Char	Dec	Char
32	[space]	58	:	84	T	110	n
33	!	59	;	85	U	111	o
34	"	60	<	86	V	112	p
35	#	61	[=]	87	W	113	q
36	$	62	>	88	X	114	r
37	%	63	?	89	Y	115	s
38	&	64	@	90	Z	116	t
39	'	65	A	91	[	117	u
40	(	66	B	92	\	118	v
41	)	67	C	93	]	119	w
42	*	68	D	94	^	120	x
43	=+	69	E	95	_	121	y
44	,	70	F	96	`	122	z
45	-	71	G	97	a	123	{
46	.	72	H	98	b	124	|
47	/	73	I	99	c	125	}
48	0	74	J	100	d	126	~
49	1	75	K	101	e	127	
50	2	76	L	102	f		
51	3	77	M	103	g		
52	4	78	N	104	h		
53	5	79	O	105	i		
54	6	80	P	106	j		
55	7	81	Q	107	k		
56	8	82	R	108	l		
57	9	83	S	109	m		

https://unicode-table.com/en/

Przykład: Generowanie losowego hasła

Przykład funkcji, która generuje losowe hasło stworzone z 10 znaków. Wynikiem jej wywołania jest ciąg znaków, który możemy wyświetlić lub przekazać do kolejnych funkcji.

```
import random
def generuj_haslo():
    haslo = ""
    for i in range(0,10):
        losowy_znak = chr(random.randrange(48,123,1))
        haslo += losowy_znak
    return haslo

print(generuj_haslo())
```

Sekcja z zadaniami

Zad.1 Napisz program, który prosi o podanie podstawy i wysokości trójkąta, a następnie wyświetla jego pole.

Zad.2 Napisz program, który przelicza cale na centymetry.
1 cal to 2.54 cm.

Zad.3 Napisz program, który będzie generował losową literę od A do Z.

Zad.4 Napisz program, który wyświetla losową liczbę w zakresie <1,10> oraz małą literę <a,j>, a następnie wyświetla ich połączenie, np: a2, b6, d7.
Mógłby to być pierwszy krok do stworzenia komputerowej gry w statki :)

Zad.5 Napisz program, który wyświetli pierwszą kolumnę tabliczki mnożenia.

Zad.6 Wyświetl liczby od 14 do 33, poza liczbą 20 i 30.

Zad.7 Napisz program, który przypisuje do listy 10 losowych liczb z przedziału od 0 - 100, sortuje je i wyświetla w jednej linii.

Zad.8 Napisz program, który pobiera długości boków akwarium w centymetrach, a następnie wyświetla jego kubaturę w litrach.

Zad.9 Sprawdzić, czy wprowadzona liczba całkowita X jest jednocześnie podzielna przez 7 oraz 3.

Zad 10. Oblicz pierwiastki równania kwadratowego: ax2 + bx + c = 0

Zad 11. Obliczyć i wyświetlić jaką drogę pokona ciało spadające swobodnie po 1, 2, ..,10 sek.
S = a*pow(t,2)/2
a = 9.81

Zad 12. Znajdź sumę cyfr podanej liczby, np: 143 = 8

Zad 13. Program obliczający rezystancję zastępczą dwóch rezystorów połączonych równolegle.

Zad 14. Napisz program, książkę. Po wyświetleniu danej strony, wyświetli zapytanie o gotowość przejścia do kolejnej lub poprzedniej strony. Po wpisaniu 2, przejdzie do kolejnej strony, a po wciśnięciu 1 wróci do poprzedniej. Po wciśnięciu 0, program się zakończy.