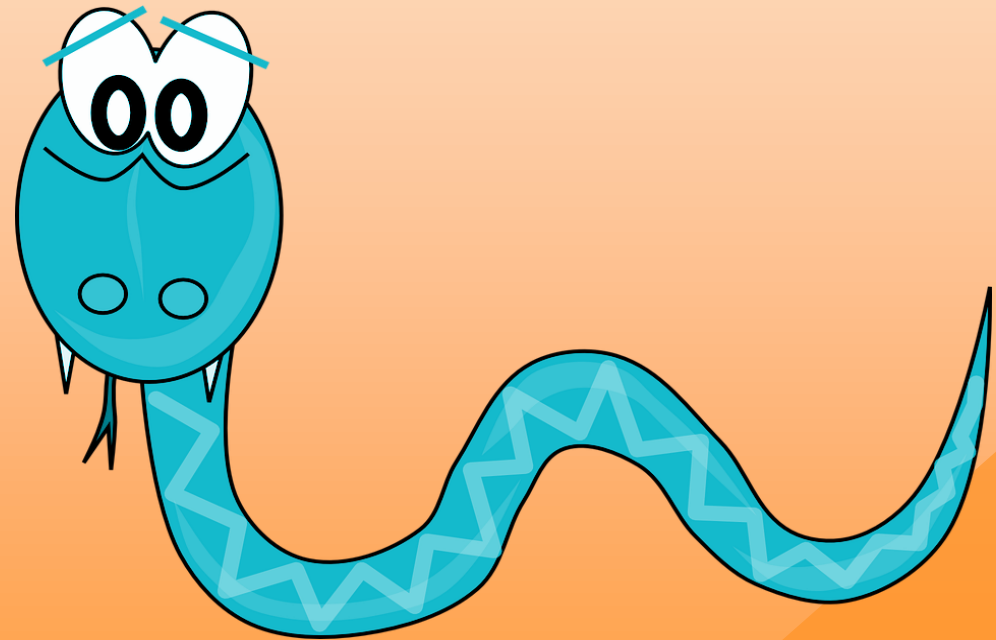


Jak ogarnąć cyfrową gadzinę?

Podstawy programowania w języku Python

01101100 11001101 ...

Dzięki tej publikacji, nauczysz się rozmawiać z komputerem (tylko nie rób tego na głos, bo domownicy zaczną na Ciebie dziwnie spoglądać :)



1

```
 print("Moja Pani i mój Panie, nadszedł czas na kodowanie")
```

Jak ogarnąć cyfrową gadzinę?

Podstawy programowania w języku Python

Zabierzemy Cię w świat pythona. W prosty i przejrzysty sposób wytłumaczymy Ci, o co, w tym wszystkim właściwie chodzi. Napišemy wiele programów konsolowych i rozwiążemy różnorodne zadania.

Przekaz merytoryczny, oprawa graficzna, przygotowanie
Aldona Wilińska, Mateusz Wiliński

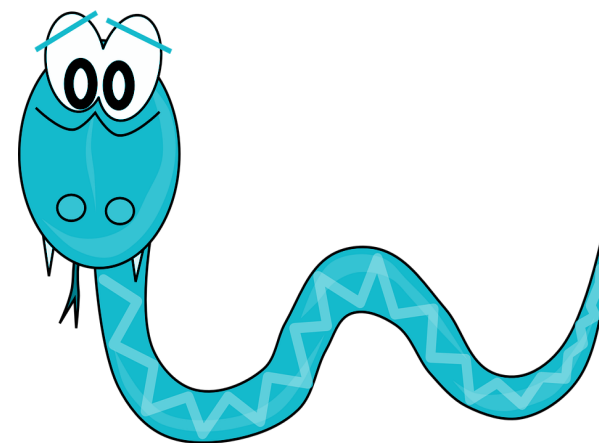
Wydawnictwo
PIKADEMIA

Wydanie 2



Legnica, 2022

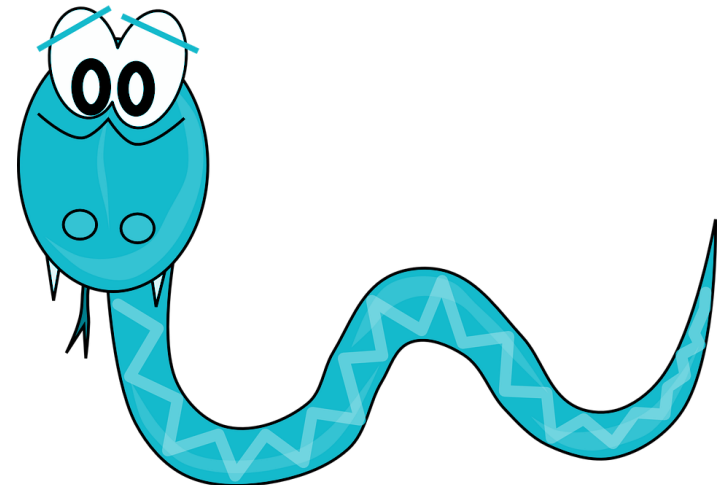
ISBN
978-83-943070-1-1



Spis zagadnień

<u>06</u>	<u>Typy informacji (int, float, string, bool)</u>
<u>08</u>	<u>Wyświetlanie informacji - print()</u>
<u>11</u>	<u>Konkatenacja</u>
<u>12</u>	<u>Konwersja typów</u>
<u>17</u>	<u>Zmienne</u>
<u>20</u>	<u>Inkrementacja i dekrementacja</u>
<u>28</u>	<u>Operacje na wartościach tekstowych</u>
<u>36</u>	<u>Wprowadzanie danych - input()</u>
<u>41</u>	<u>Instrukcja warunkowa</u>
<u>46</u>	<u>Operatory porównania (!=, ==, >, >=, <, <=)</u>
<u>47</u>	<u>Operatory logiczne (and, or, not)</u>
<u>56</u>	<u>Losowa liczba - random.randrange()</u>
<u>60</u>	<u>Pętla for</u>
<u>67</u>	<u>Pętla while</u>
<u>79</u>	<u>Listy</u>
<u>88</u>	<u>Funkcje</u>
<u>98</u>	<u>Zmienne lokalne i globalne</u>
<u>103</u>	<u>Modulo (%)</u>

+ wiele interesujących zadań.



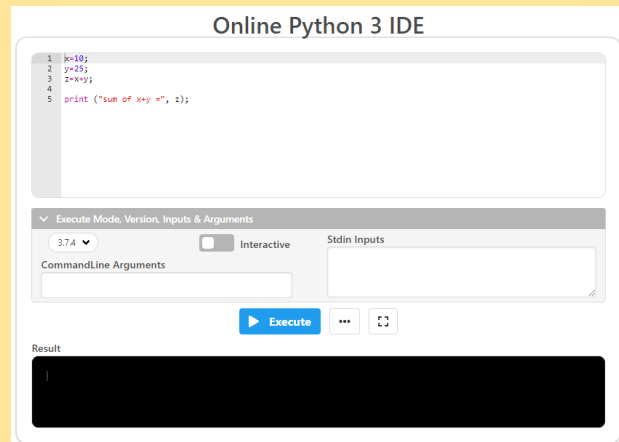
Edytor i kompilator

Python (czytaj pajton) jest tekstowym językiem programowania. Do pisania kodu potrzebujemy edytora tekstu, ale do tego, by nasz program zadziałał potrzebujemy specjalnego kompilatora.

Kompilator zamienia składnię języka python na język zrozumiały dla komputera. Zanim zaczniemy programowanie potrzebujemy zaopatrzyć się w taki edytor i kompilator.

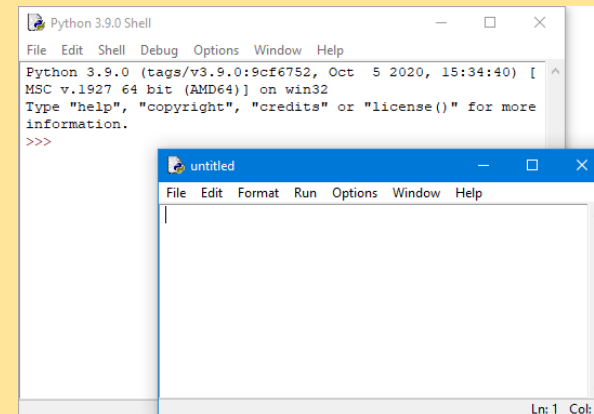
Najłatwiejszym rozwiązaniem jest skorzystanie z darmowych środowisk dostępnych online, np:

<https://www.jdoodle.com/python3-programming-online/>



Możemy też pobrać IDE, czyli środowisko zintegrowane (np. IDLE Python Shell), które można zainstalować na komputerze. Ta opcja jest o tyle lepsza, że możemy zapisywać swoje pliki, a także zainstalowane IDE działa szybciej od wersji przeglądarkowych.

<https://www.python.org/downloads/>

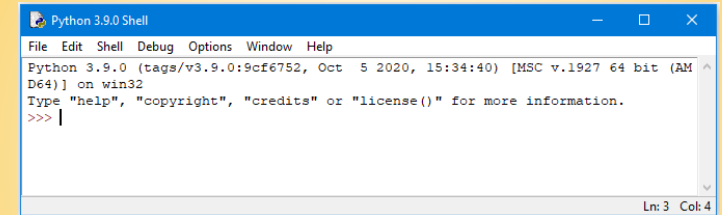


Obsługa środowiska IDLE Python Shell

1. Pobieramy program IDLE Python Shell z linku:
<https://www.python.org/downloadloads/>

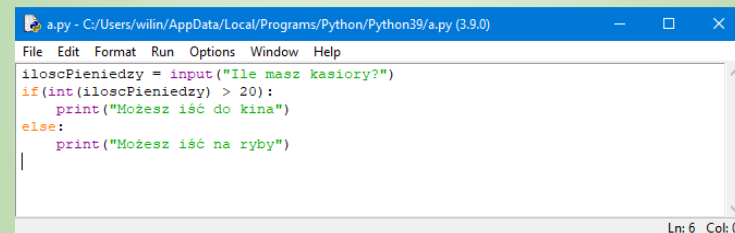
2. Instalujemy IDLE na komputerze.

3. Po uruchomieniu aplikacji zobaczymy taki ekran:



W tym oknie możemy wpisywać pojedyncze instrukcje oraz wchodzić w interakcję z tworzonym i uruchomionym przez nas programem

4. Kod, powyżej 1 linii będziemy pisać w nowym pliku.
W tym celu klikamy File >> New File.



Możemy też użyć skrótu Ctrl + N.

5. Po wpisaniu kodu, musimy go zapisać.
W tym celu wybieramy z menu File >> Save >> Podajemy nazwę i klikamy Save.
Możemy też użyć skrótu Ctrl + S.

6. Teraz możemy już uruchomić nasz program, zatem klikamy Run >> Run Module lub używamy skrótu F5.

Typy informacji

W życiu mamy do czynienia z wieloma typami informacji, np:

**Liczby całkowite
(integer)**

np. 12, -100, 256

**Liczby ułamkowe
(float)**

np. 0.25, 36.6, 1.0

**Tekst
(string)**

np. "Ania", "Cześć"

**Stan logiczny
(bool)**

True, False

Języki programowania też rozróżniają te typy i choć w wielu przypadkach nie musimy się tym przejmować, to czasami wiedza na ich temat okaże się niezbędna.

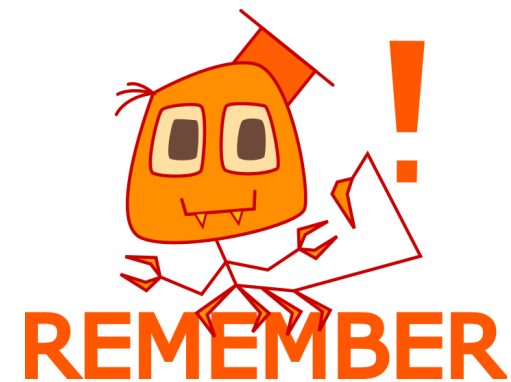
Na początek warto zapamiętać, że wartości tekstowe zapisujemy wewnątrz cudzysłowu, np: "Hello". Moglibyśmy też zapisać taki wyraz wewnątrz apostrofu, np: 'Hello', ale w pewnych sytuacjach użycie cudzysłowów jest lepszym rozwiązaniem.

Pamiętaj, że dla komputera, liczba 2 napisana bez cudzysłowu jest innego typu niż liczba 2 wewnątrz cudzysłowu "2".



Kolejność interpretacji kodu

**Pamiętaj, że kod jest czytany
z góry do dołu
i od lewej do prawej.
Taka jest też kolejność
wykonywanych instrukcji.**



Wyświetlanie informacji - print()

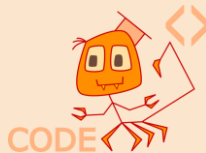
W języku python możemy wyświetlić informacje za pomocą metody **print()**.

print()

Wewnątrz nawiasu, jako argument, podajemy informację, która ma zostać wyświetlona. Może to być pojedyncza informacja, taka jak: liczba czy tekst lub zbiór wielu elementów, odpowiednio ze sobą połączonych. Może to być również wyrażenie matematyczne oraz wiele innych informacji.

```
# Wyświetlanie różnych  
treści w konsoli:
```

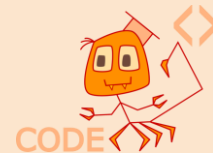
```
print(20)  
print(36.6)  
print("Cześć")
```



CODE

```
# Wyświetlanie wyniku działań  
matematycznych:
```

```
print(2+4)  
print(8-1)  
print(2*2)  
print(10/2)
```

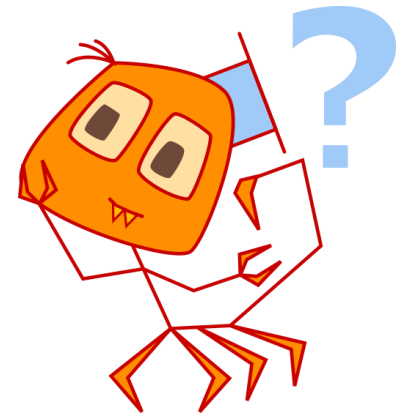


CODE

Zadania

1. Wyświetl w konsoli wynik działania: $2 + 2 * 2$

2. Wyświetl w konsoli swoje imię



Zadania z odpowiedziami

1. Wyświetl w konsoli wynik działania: $2 + 2 * 2$

```
print(2 + 2 * 2)
```

2. Wyświetl w konsoli swoje imię

```
print("Mateusz")
```



Łączenie informacji - konkatenacja

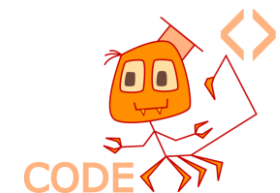
konkatenacja

Bardzo 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.

Konkatenacja tworzy treść typu tekstowego (string).

Łącząc wyrażenia typu string możemy używać znaku + bez żadnych komplikacji, aby jednak połączyć wartości tekstowe z liczbami, musimy liczby zamieniać na wartości tekstowe za pomocą metody str().

```
# Konkatenacja z użyciem przecinka  
print("Wynik działania wynosi ", 3 * 6)
```



```
# Konkatenacja z użyciem operatora +  
print("Wynik działania wynosi " + str(3 * 6))
```

Konwersja typów

Niejednokrotnie będziemy zmuszeni do konwersji danego typu na inny. Służą do tego odpowiednie metody, np:

int()

zamiana na liczby całkowite

np. `int("12")`

float()

zamiana na liczby ułamkowe

np. `float("12.5")`

str()

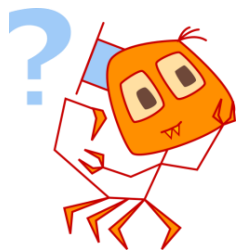
zamiana na tekst

np. `str(12)`

Przykłady konwersji

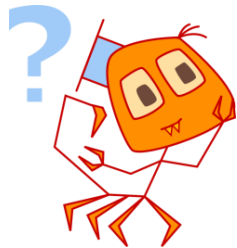
```
print("Maja ma " + str(5) + " lizaków")  
print("Maja ma " + str(int("5") + 2) + " lizaków")  
print(float(2))
```

Zadanie



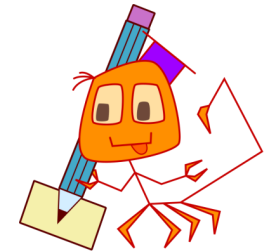
Wyświetl w konsoli informację o swoim adresie łącząc nazwę ulicy z numerem domu, np: ul.Wesoła 24

Zadanie z odpowiedzią



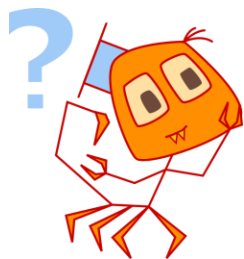
Wyświetl w konsoli informację o swoim adresie łącząc nazwę ulicy z numerem domu, np: ul.Wesoła 24

```
print("ul.Wesoła " +  
      str(24))
```



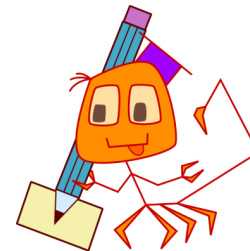
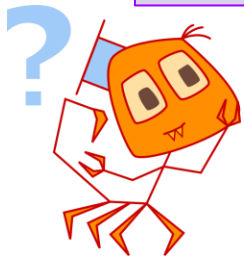
Zadanie

Wyświetl informację o swoim wieku łącząc 3 elementy,
np: Mam 12 lat.
Użyj tekstu, liczby i tekstu.



Zadanie z odpowiedzią

Wyświetl informację o swoim wieku łącząc 3 elementy,
np: Mam 12 lat.
Użyj tekstu, liczby i tekstu.



```
print("Mam " + str(12) + " lat.")
```


Zmienna

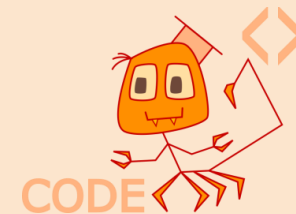
Czym jest zmienna?

Możemy sobie wyobrazić, że zmienna, to takie 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.



Zawartość pudełka, czyli wartość zmiennej przypisujemy za pomocą operatora przypisania, czyli znaku =.

```
# Tworzymy zmienną o nazwie liczbaPunktow i przypisujemy do niej
wartość 0, za pomocą znaku =.
Następnie wyświetlamy zawartość pudełka, czyli wartość zmiennej
liczbaPunktow.
liczbaPunktow = 0
print("Punkty: " + str(liczbaPunktow))
```



Nazwy zmiennych

Ustalając nazwę zmiennej powinniśmy przestrzegać kilku zasad:

- zaczynamy od małej litery
- używamy liter podstawowych (bez polskich akcentów)
- wielkość liter ma znaczenie (imie, Imie, IMie to trzy różne zmienne)
- możemy używać cyfr lub podkreślenia (_), ale nie jako pierwszy znak nazwy
- nazwy składające się z wielu wyrazów łączymy zaczynając kolejny wyraz wielką literą, np(mojAdres)
- w nazwach nie stosujemy spacji

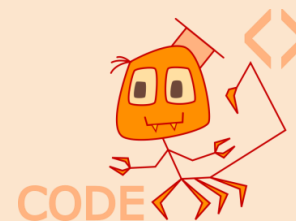


Nadpisywanie zmiennej

Wartości zmiennych możemy w łatwy sposób nadpisywać podając jej nazwę i nową wartość, np:

#1 Tworzymy zmienną o nazwie `liczbaPunktow` i przypisujemy do niej wartość `0`, za pomocą znaku `=`.
#2 Następnie używamy ponownie tej samej nazwy i podajemy nową wartość, czyli nadpisujemy ją.

```
liczbaPunktow = 0  
liczbaPunktow = 1  
print("Punkty: " + str(liczbaPunktow))
```



Zwiększenie wartości zmiennej - inkrementacja i dekrementacja

Wartości zmiennych możemy modyfikować zwiększając jej wartość, bazując na jej poprzedniej wartości.

W takich przypadkach nie możemy po prostu przypisać nowej wartości, musimy wykonać operację zwiększenia lub zmniejszenia poprzedniej wartości. Taki zabieg nazywamy inkrementacją lub dekrementacją.

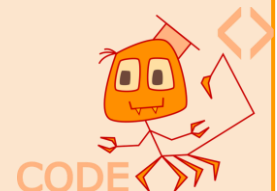
Czasami będziemy chcieli zwiększyć wartość za pomocą znaku + i -, a czasami możemy chcieć zwiększyć lub zmniejszyć daną wartość kilkukrotnie, wtedy posłużymy się znakiem * lub /.

Ponieważ jednocześnie będziemy zmieniać tą wartość i przypisywać ją do zmiennej tuż po znaku zmiany (+, -, *, /) dodamy znak przypisania, czyli =.

Po znaku =, podajemy liczbę o jaką, to zwiększenie lub zmniejszenie ma nastąpić.

```
# Zwiększamy zmienną o 1
# Tworzymy zmienną o nazwie liczbaPunktow i przypisujemy do niej wartość
0, za pomocą znaku =.
# Następnie dwukrotnie zwiększamy wartość zmiennej o 1.
```

```
liczbaPunktow = 0
liczbaPunktow += 1
liczbaPunktow += 1
print("Punkty: " + str(liczbaPunktow))
```



Zwiększenie wartości zmiennej - inkrementacja i dekrementacja

Wartość zmiennej możemy modyfikować poprzez różne działania matematyczne, tj. dodawanie, odejmowanie, mnożenie, dzielenie, itd.

```
# Przykłady inkrementacji i dekrementacji
```

```
liczbaPunktow = 0
```

```
liczbaPunktow += 3
```

```
liczbaPunktow -= 1
```

```
liczbaPunktow *= 4
```

```
liczbaPunktow /= 2
```

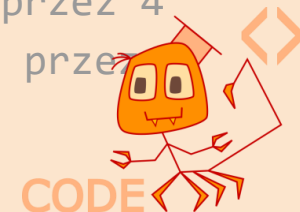
```
# dodajemy 3
```

```
# odejmujemy 1
```

```
# mnożymy wartość zmiennej przez 4
```

```
# dzielimy wartość zmiennej przez 2
```

```
print("Punkty: ", liczbaPunktow)
```

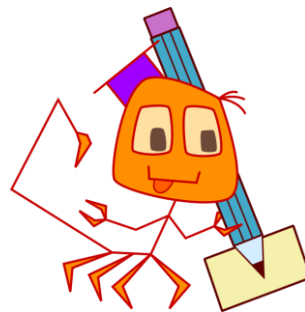
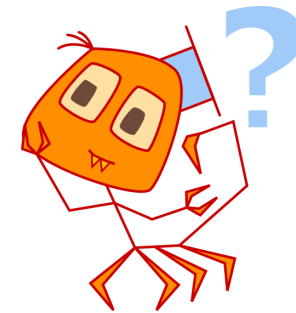


Zmienna

Przykład z rozwiązaniem

Stwórz kod na podstawie wytycznych poniżej:

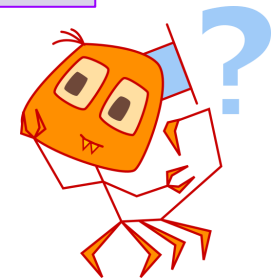
1. Stwórz zmienną o nazwie hp i przypisz do niej wartość 10.
2. Po otrzymaniu obrażeń, gracz traci 2 punkty hp. Zakoduj to.
3. Po wypiciu mikstury, gracz odzyskuje 1 punkt życia.
4. Po rzuceniu czaru ilość hp zwiększa się dwukrotnie.
5. Wróg rzuca na nas klątwę, która redukuje ilość hp 3-krotnie.
6. Wyświetl ilość hp jako zdanie: Hp wynosi X



```
hp = 10
hp -= 2
hp += 1
hp *= 2
hp /= 3
print("HP wynosi ", hp)
```

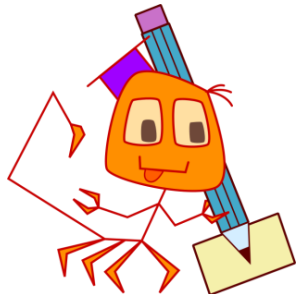
Zadanie

Stwórz zmienną o nazwie `imie` i przypisz do niej swoje imię.
Następnie za pomocą konkatencji wypisz zdanie:
Mam na imię + twoje imię.

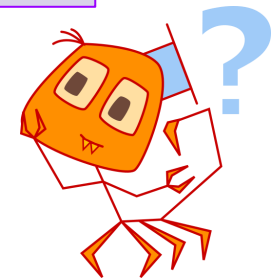


Zadanie z odpowiedzią

Stwórz zmienną o nazwie `imie` i przypisz do niej swoje imię.
Następnie za pomocą konkatencji wypisz zdanie:
Mam na imię + twoje imię.

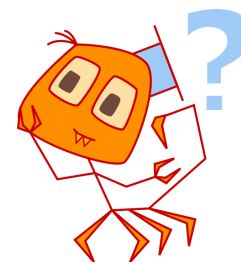


```
imie = "Mateusz"  
print("Mam na imię " + imie)
```



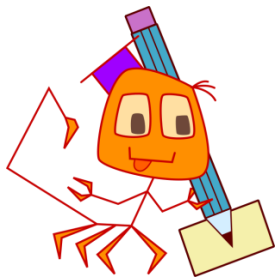
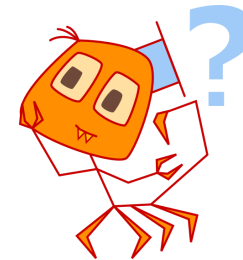
Zadanie

Stwórz 2 zmienne o nazwie liczba1 i liczba2, a następnie przypisz do nich dowolne liczby.
Za pomocą konkatencji wypisz treść działania dodawania tych liczb i ich wynik.



Zadanie z odpowiedzią

Stwórz 2 zmienne o nazwie liczba1 i liczba2, a następnie przypisz do nich dowolne liczby.
Za pomocą konkatencji wypisz treść działania dodawania tych liczb i ich wynik.



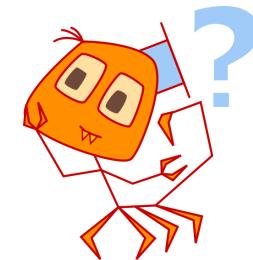
```
liczba1 = 8  
liczba2 = 4  
print(liczba1, " + ", liczba2, " = " ,liczba1 + liczba2)
```

Zadanie

Bez uruchamiania kodu, spróbuj przeanalizować kod i powiedzieć jaka liczba zostanie wypisana w konsoli.

Sprawdź wynik wklejając kod do kompilatora.

```
liczba = 2  
liczba *= 3  
liczba -= 1  
liczba *= 2  
liczba /= 5  
liczba -= 1  
print(liczba)
```

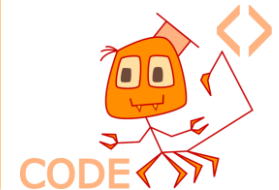


Operacje na wartościach tekstowych

Często zdarza się, że potrzebujemy 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. indeks. W programowaniu, indeksowanie, czyli liczenie, zaczynamy od 0. Prześledźmy kilka przykładów:

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

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



```
# Wypisz 2 i 3 znak wyrazu  
"Ania" przypisanego do  
zmiennnej o nazwie imie (ni)  
imie = "Ania"  
print(imie[1:3])
```

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

Operacje na wartościach tekstowych

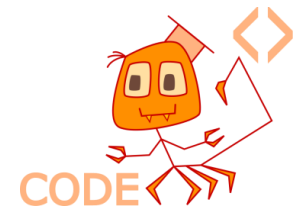
Za pomocą dwukropka(:) ustalamy zakres.

Dzięki wartościom ujemnym możemy wybrać liczby zaczynając od końca łańcucha znaków.

```
# Wypisz trzy ostatnie znaki  
wyrazu "Hallowina" (ina)  
print("Hallowina"[-3:])
```

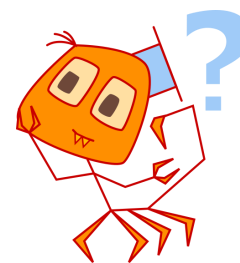
```
# Wypisz 3 znaki wyrazu  
"Komputer" licząc od tyłu,  
poza ostatnim znakiem (ere)  
print("Komputer"[-4:-1])
```

```
# Wypisz pięć ostatnich znaków  
wyrazu "telewizor" przypisanego  
do zmiennej o nazwie produkt  
(wizor)  
produkt = "telewizor"  
print(produkt[-5:])
```



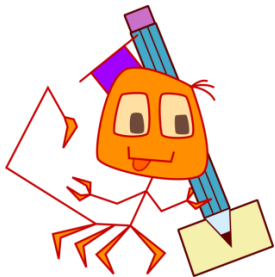
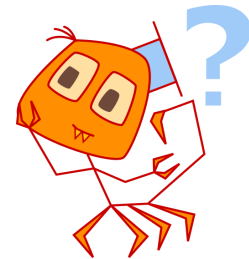
Zadanie

Napisz program, który używa 2 zmiennych do obliczenia i wyświetlenia działania odejmowania



Zadanie z odpowiedzią

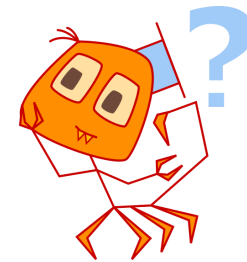
Napisz program, który używa 2 zmiennych do obliczenia i wyświetlenia działania odejmowania



```
licz1 = 30  
licz2 = 12  
print(licz1, " - " , licz2, " = " , licz1 - licz2)
```

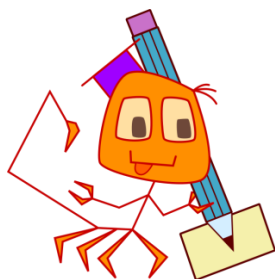
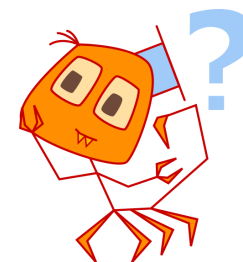
Zadanie

Z dowolnej zmiennej typu string wypisz w konsoli
tylko 3 i 4 literę



Zadanie z odpowiedzią

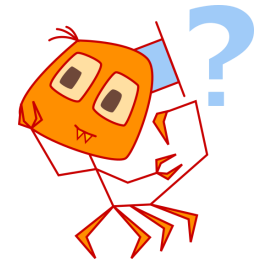
Z dowolnej zmiennej typu string wypisz w konsoli tylko 3 i 4 literę



```
imie = "Mateusz"  
print(imie[2:4])
```

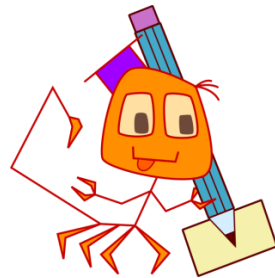
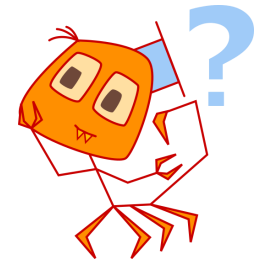
Zadanie

Wypisz wszystkie znaki z wyrazu "butterfly" poza trzema ostatnimi.
Spróbuj znaleźć rozwiązanie w internecie.



Zadanie z odpowiedzią

Wypisz wszystkie znaki z wyrazu "butterfly" poza trzema ostatnimi.
Spróbuj znaleźć rozwiązanie w internecie.



```
print("butterfly"[:-3])
```

Wprowadzanie treści do programu - input()

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 będziemy chcieli metodę input() przypisać do jakiejś zmiennej, tak, aby móc wpisaną wartość zapisać i przetwarzać w programie.

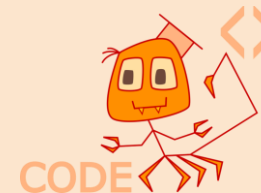
Korzystając z kompilatora jdoodle, powinniśmy włączyć opcję interactive:



input()

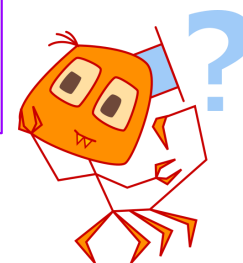
```
# Poproszenie użytkownika o podanie swojego imienia.  
Po naciśnięciu Enter, wyświetlimy mu komunikat  
powitalny wraz z podanym imieniem
```

```
wprowadzoneImie = input("Podaj imie: ")  
print("Witaj " + wprowadzoneImie)
```



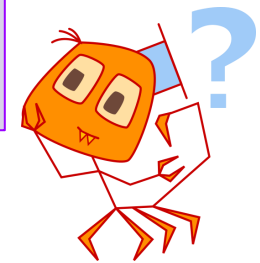
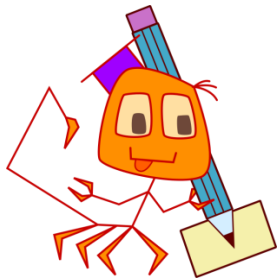
Zadanie

Napisz program, który poprosi użytkownika o podanie swojego imienia, a następnie wyświetli pierwszą i ostatnią literę jego imienia.



Zadanie z odpowiedzią

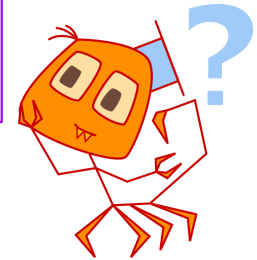
Napisz program, który poprosi użytkownika o podanie swojego imienia, a następnie wyświetli pierwszą i ostatnią literę jego imienia.



```
wprowadzoneImie = input("Podaj imie: ")  
print("Pierwsza litera Twojego imienia to " + wprowadzoneImie[0])  
print("Ostatnia litera Twojego imienia to " + wprowadzoneImie[-1])
```

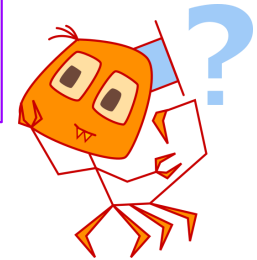
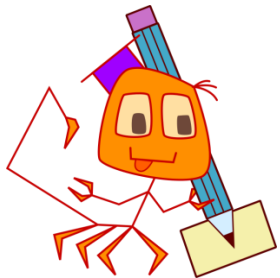
Zadanie

Napisz program, który poprosi użytkownika o podanie liczby, a następnie zwróci jej potęgę 2-stopnia.



Zadanie z odpowiedzią

Napisz program, który poprosi użytkownika o podanie liczby, a następnie zwróci jej potęgę 2-stopnia.



```
liczba = float(input("Podaj liczbę: "))  
print(liczba, " do kwadratu to ", liczba*liczba)
```


Instrukcja warunkowa if

Instrukcja warunkowa służy do podejmowania decyzji w oparciu o podane warunki.

Przykład z życia wzięty:

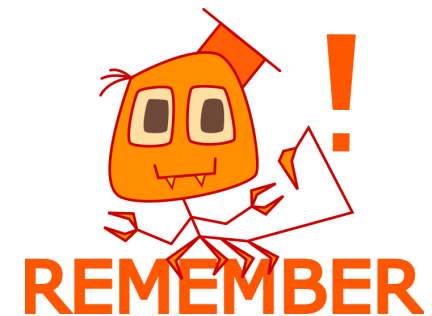
Jeżeli będę miał co najmniej 20 zł, to pójdę do kina, a jak nie, to pójdę na ryby.

Spróbujmy się zastanowić jakie kroki musimy zawrzeć w kodzie:

1. Musimy wiedzieć ile osoba ma pieniędzy. Możemy zapytać użytkownika za pomocą `input()` lub stworzyć zmienną i przypisywać do niej jakąś wartość.
2. Jeżeli `iloscPieniedzy` jest większa od 20 to wyświetl napis "Możesz iść do kina", a w przeciwnym razie wyświetl napis "Możesz iść na ryby". Ten punkt stworzymy za pomocą instrukcji warunkowej.

Szablon
podstawowej
instrukcji
warunkowej:

```
if(warunek):  
    wykonaj instrukcje  
else:  
    wykonaj instrukcje
```



Instrukcja warunkowa if

Jeżeli będę miał co najmniej 20 zł, to pójdę do kina, a jak nie, to pójdę na ryby.

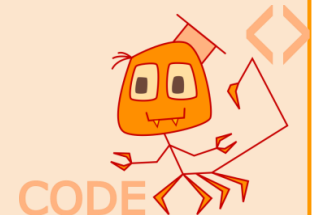
Używając pseudokodu, możemy to zadanie zapisać w ten sposób:

```
ilość pieniędzy = np. 25zł
if(ilość pieniędzy jest większa od 20 zł):
    Wypisz: Możesz iść do kina
else:
    Wypisz: Możesz iść na ryby.
```

Spróbuj zmienić wartość zmiennej
iloscPieniedzy na 15.
Jaka wiadomość zostanie wyświetlona?

Użycie instrukcji warunkowej if

```
iloscPieniedzy = 25
if(iloscPieniedzy > 20):
    print("Możesz iść do kina")
else:
    print("Możesz iść na ryby")
```



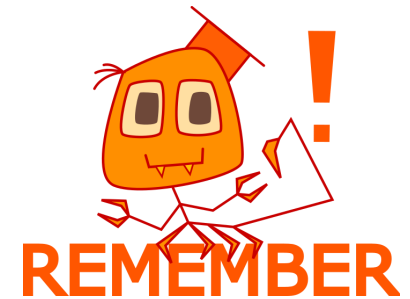
Wcięcia kodu

W przykładzie poniżej mamy do czynienia z dwoma blokami kodu, z czego jeden(pierwszy) posiada kolejny blok zagnieżdżony. To tak, jakbyśmy mieli ustawionych kilka pudełek obok siebie. Najpierw musimy otworzyć pierwsze pudełko, zajrzeć do niego i otworzyć wszystkie pudełka, które są w środku, a dopiero wtedy możemy przejść do kolejnego pudełka głównego.

```
if(warunek):
```

```
    print("Tekst z wnętrza bloku if")
```

```
print("Tekst z osobnego bloku")
```



Język Python działa na zasadzie indentacji, czyli wcięć kodu (tabulacji). Główne bloki kodu zaczynają się tuż przy krawędzi ekranu, a im bardziej zagnieżdżamy kolejne bloki, tym oddalamy je od ekranu za pomocą przycisku Tabulatora.

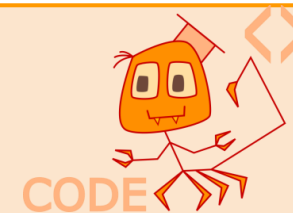
Instrukcja warunkowa if

Instrukcja warunkowa, wewnątrz nawiasu, sprawdza warunek, który, zawsze, w wyniku przyjmuje jedną z dwóch wartości: prawdę lub fałsz (True or False). Wewnątrz tego nawiasu zazwyczaj porównujemy jedną wartość z drugą (np. czy `iloscPieniedzy` jest większa od 20, czyli `iloscPieniedzy > 20`). Wynikiem takiego zapytanie może być tylko prawda lub fałsz.

Jeżeli jest to prawda, to wykonają się instrukcje wewnątrz tego bloku `if`, jeśli jednak warunek był fałszywy to kompilator pominie kod i przejdzie do kolejnego warunku, itd.

Przyjrzyjmy się kolejnemu przykładowi instrukcji warunkowej w którym będziemy sprawdzać poprawność hasła zapisanego w systemie i tego wprowadzanego przez użytkownika.

```
hasloWSystemie = "1234"
podaneHaslo = input("Podaj hasło: ")
if(podaneHaslo == hasloWSystemie):
    print("Hasło poprawne, witaj")
else:
    print("Hasło błędne, autodestrukcja...")
```

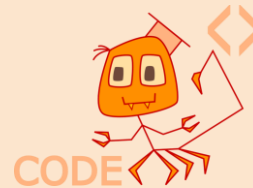


Zmienna typu bool

Zmienna typu bool (boolean) to zmienna, która może przechowywać tylko jedną z dwóch wartości: prawdę lub fałsz, czyli True or False, a także 1 lub 0. Jest to podstawowy typ wartości, na którym opiera się istota informatyki i programowania. W taki sposób tworzone są komputery, które opierają się na sygnałach wysokich (1, True) oraz niskich (0, False).

Poniżej 2 przykłady inicjalizacji zmiennych typu bool.

```
czyMenuJestWlaczzone = False  
czyGraczDotykaZiemi = True
```



Dzięki zmiennym typu bool, możemy przypisywać i sprawdzać stany elementów programu, np:

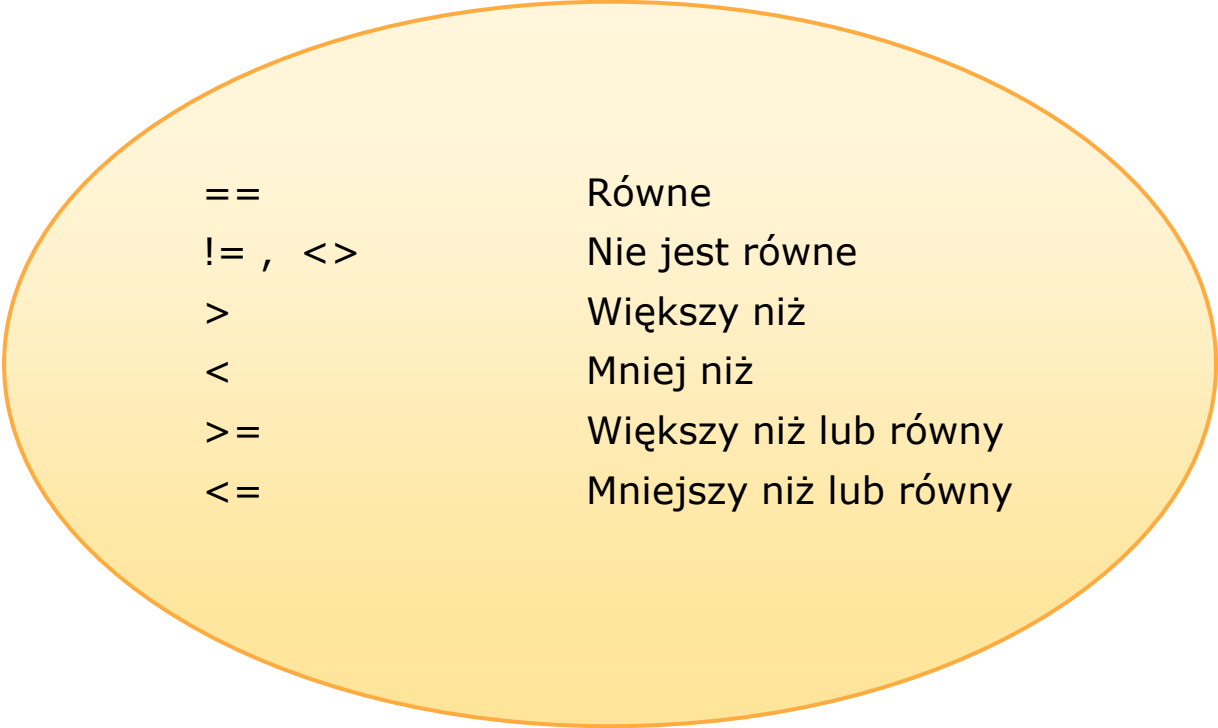
```
czyGraczDotykaZiemi = True
```

```
czyPrzeciwnikJestWZasieguStrzalu = False
```

Za pomocą instrukcji warunkowej, na podstawie zmiennych typu bool, możemy uruchamiać kolejne porcje kodu.

Operatory porównania

Aby odpowiednio porównywać wyrażenia w instrukcji warunkowej musimy znać operatory porównania. Możliwe, że już je znacie z lekcji matematyki.



==	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(i nie tylko) połączyć kilka warunków, np:

W życiu codziennym musimy mieć auto i paliwo, by gdzieś pojechać.

Stosujemy operator **and**, ponieważ musimy mieć i auto i paliwo.

lub

Możemy pojechać do sklepu autem lub rowerem.

Stosujemy operator **or**, ponieważ wystarczy nam tylko rower lub samochód, by pojechać do sklepu.

and

Stosując operator **and** oba warunki muszą być prawdziwe, by całe wyrażenie również było prawdziwe.

or

Stosując operator **or** wyrażenie będzie prawdziwy, gdy choć jeden z warunków będzie prawdziwy

not

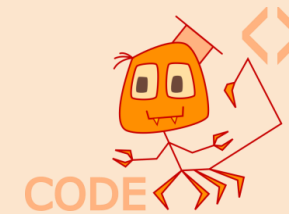
Stosując operator **not** całe wyrażenie logiczne (warunki) zmieniają wartość, czyli True zmienia się na False i odwrotnie

Instrukcja warunkowa if

Instrukcja warunkowa może określać więcej warunków, wtedy musimy posłużyć się kolejnymi blokami, które nazywamy elif (od else if).

Spójrzmy na przykład:

```
hasloWSystemie = "1234"
podaneHaslo = input("Podaj hasło: ")
if(podaneHaslo == hasloWSystemie):
    print("Hasło poprawne, witaj")
elif(podaneHaslo == "0000"):
    print("Uuu, znalazłeś lukę w systemie")
else:
    print("Hasło błędne, autodestrukcja...")
```



Instrukcja warunkowa if

```
if(warunek1):  
    wykonaj instrukcje 1  
elif(warunek2):  
    wykonaj instrukcje 2  
elif(warunek3):  
    wykonaj instrukcje 3  
else:  
    wykonaj instrukcje 4
```

Ważnym jest w jakiej kolejności projektujemy nasze warunki, ale tego nauczymy się z czasem :)

Pamiętaj, że kod jest czytany od góry w dół i w taki też sposób będą rozpatrywane kolejne bloki instrukcji warunkowej.

Najpierw kompilator sprawdza czy warunek1 jest prawdziwy.

Jeżeli warunek1 jest prawdziwy to kompilator przejdzie do wnętrza bloku i wykona instrukcje 1. Po wykonaniu instrukcji 1 kompilator będzie już szczęśliwy i nie będzie sprawdzał ani wykonywał kolejnych warunków i instrukcji. Jeżeli warunek1 będzie fałszywy to kompilator nie wejdzie do wnętrza, po prostu pominie ten blok i przejdzie do kolejnej instrukcji elif i sprawdzi warunek2.

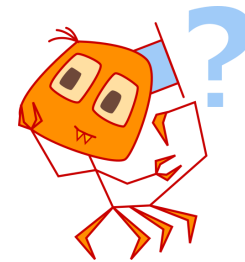
Jeżeli warunek2 będzie prawdziwy to kompilator wejdzie do wnętrza bloku i wykona instrukcje 2. W przeciwnym razie blok zostanie pominięty, a kompilator przejdzie do kolejnego warunku3.

Kompilator będzie sprawdzał warunki do momentu, aż jeden z nich będzie prawdziwy.

Ostatecznie wykona instrukcje zawarte w bloku else, jeśli żadne warunki powyżej nie były prawdziwe.

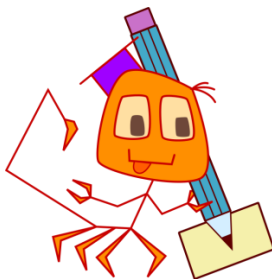
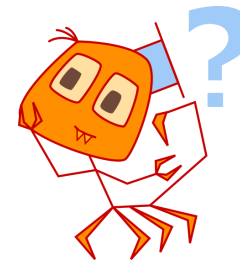
Zadanie

Napisz program, który pobierze bok kwadratu i zwróci jego pole powierzchni oraz obwód.



Zadanie z odpowiedzią

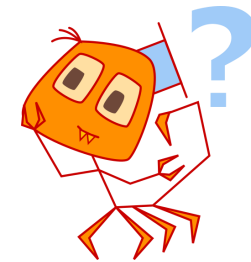
Napisz program, który pobierze bok kwadratu i zwróci jego pole powierzchni oraz obwód.



```
bok = float(input("Podaj bok kwadratu: "))
pole = bok * bok
obwod = 4 * bok
print("Pole wynosi: " + str(pole))
print("Obwód wynosi: " + str(obwod))
```

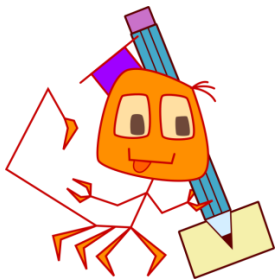
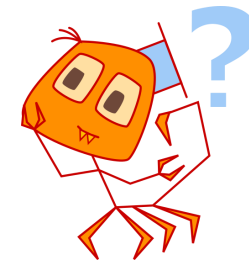
Zadanie

Napisz program, który przechowuje aktualny rok w zmiennej. Program pyta użytkownika o rok urodzenia, a następnie zwraca wiek użytkownika.



Zadanie z odpowiedzią

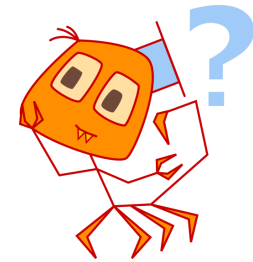
Napisz program, który przechowuje aktualny rok w zmiennej. Program pyta użytkownika o rok urodzenia, a następnie zwraca wiek użytkownika.



```
aktualnyRok = 2020
rokUrodzenia = int(input("Podaj rok urodzenia: "))
wiek = aktualnyRok - rokUrodzenia
print("Masz " + str(wiek) + " lat.")
```

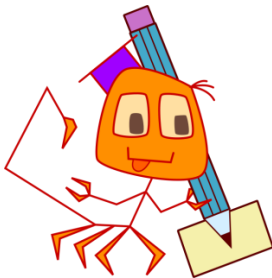
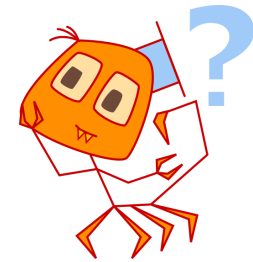
Zadanie

Napisz program, który tworzy proste działanie matematyczne, np: dodawanie na podstawie dwóch zmiennych. Działanie ma się wyświetlić użytkownikowi, tak, aby mógł wpisać odpowiedź. Jeśli odpowiedź będzie prawidłowa, ma się wyświetlić komunikat "Brawo", a w przeciwnym razie ma się wyświetlić komunikat o treści "Błąd".



Zadanie z odpowiedzią

Napisz program, który tworzy proste działanie matematyczne, np: dodawanie na podstawie dwóch zmiennych. Działanie ma się wyświetlić użytkownikowi, tak, aby mógł wpisać odpowiedź. Jeśli odpowiedź będzie prawidłowa, ma się wyświetlić komunikat "Brawo", a w przeciwnym razie ma się wyświetlić komunikat o treści "Błąd".



```
licz1 = 6
licz2 = 3
suma = licz1 + licz2
trescZadania = licz1, " + ",licz2, " = "
odpowiedz = input(trescZadania)
if(int(odpowiedz) == suma):
    print("Brawo")
else:
    print("Błąd")
```

Losowa liczba z przedziału

Czas wygenerować losową liczbę z pewnego przedziału. Zazwyczaj będziemy chcieli taką wartość przypisać do zmiennej. Aby jednak mieć taką możliwość musimy powiedzieć kompilatorowi, że chcemy dołączyć taką opcję do programu. Musimy zaimportować bibliotekę random na samym początku programu.

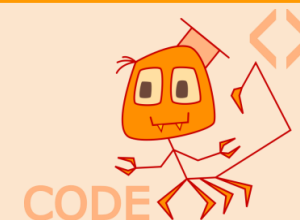
import random

Mamy do dyspozycji kilka metod do generowania losowych liczb, my skorzystamy z `random.randrange()`. Jako argument funkcji podajemy wartość minimalną, maksymalną (która jest zawsze o 1 większa od tej, którą chcemy wylosować) oraz skok, czyli jeśli ma to być każda liczba, to wpisujemy 1.

```
#importujemy bibliotekę random
import random

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

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



Losowa liczba z przedziału

```
import random
```

```
# losuje liczby całkowite od 0 do 10
```

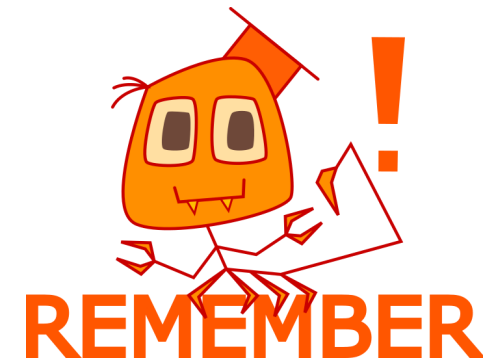
```
random.randrange(11)
```

```
# losuje liczby od 10 do 20, co 1
```

```
random.randrange(10, 21, 1)
```

```
# losuje liczby ułamkowe od 0 do 1
```

```
random.random()
```



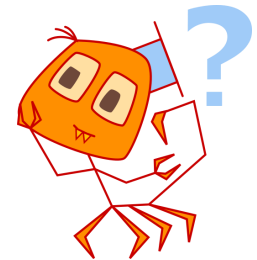
Zadanie

Napisz program, który tworzy proste działanie matematyczne, np: dodawanie na podstawie trzech zmiennych losowych w przedziale od 1 do 10.

Działanie ma się wyświetlić użytkownikowi, tak, aby mógł wpisać odpowiedź.

Jeśli odpowiedź będzie prawidłowa, ma się wyświetlić komunikat "Brawo", a w przeciwnym razie ma się wyświetlić komunikat o treści "Błąd".

Spróbuj napisać program od nowa, nie patrząc na poprzednie zadanie.



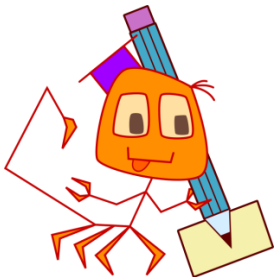
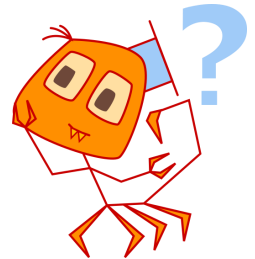
Zadanie z odpowiedzią

Napisz program, który tworzy proste działanie matematyczne, np: dodawanie na podstawie trzech zmiennych losowych w przedziale od 1 do 10.

Działanie ma się wyświetlić użytkownikowi, tak, aby mógł wpisać odpowiedź.

Jeśli odpowiedź będzie prawidłowa, ma się wyświetlić komunikat "Brawo", a w przeciwnym razie ma się wyświetlić komunikat o treści "Błąd".

Spróbuj napisać program od nowa, nie patrząc na poprzednie zadanie.



```
import random
l1 = random.randrange(1,11,1)
l2 = random.randrange(1,11,1)
l3 = random.randrange(1,11,1)
suma = l1 + l2 + l3
trescZadania = str(l1) + " + " + str(l2) + " + " + str(l3) + " = "
odpowiedz = input(trescZadania)
if(int(odpowiedz) == suma):
    print("Brawo")
else:
    print("Błąd")
```

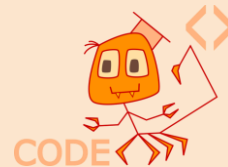
Pętla for

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?

Bez znajomości pętli zadanie to byłoby mordęgą. Zobaczmy więc jak to zrobić za pomocą pętli for.

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



Czy to nie wspaniałe? 2 linie kodu sprawia, że wyświetlimy 1000 kolejnych liczb. Zauważ, że podobnie jak z losową liczbą, tutaj też wartość maksymalną zwiększamy o 1.

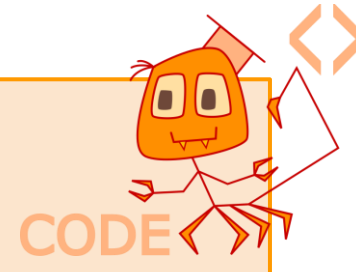
`i` to zmienna lokalna, tzw. iteracyjna (dlatego nazywa się `i` :).

Z każdym przejściem pętli wartość tej zmiennej zmienia się, w tym przypadku zwiększa się o 1, aż dojdzie do 1000.

Pętla for

Jak działa pętla?

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



Dla zmiennej **i** sprawdzamy jaka jest jej wartość. W pierwszym cyklu pętli zostaje jej przypisana wartość 1 bo taka wartość jest podana w nawiasie. Kompilator sprawdza czy zachodzi tutaj warunek prawdziwy, czyli czy **i** jest w przedziale 1 : 1000, jeśli tak jest to przechodzi do wnętrza bloku i drukuje wartość **i**.

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 kod przechodzi do kolejnego bloku, jeżeli taki istnieje w naszym kodzie.

W tym typie pętli wartość **i** zwiększa się automatycznie o 1, w innych pętlach sami możemy zdecydować jak ta wartość ma się zmieniać.

Pętla for

```
# wypisanie liczb od 1 do 10  
for i in range(1, 11):  
    print(i)
```



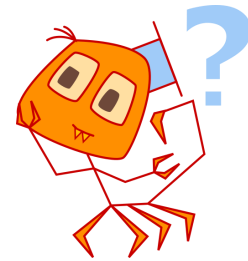
W pętli for zmienna `i` zwiększa się o 1 automatycznie.

Po wyjściu poza zakres podany w nawiasie, działanie pętli kończy się.

Liczba maksymalna zakresu jest zwiększana o 1.

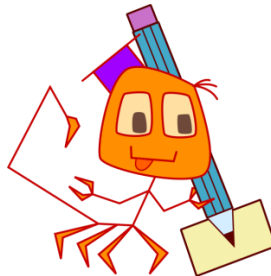
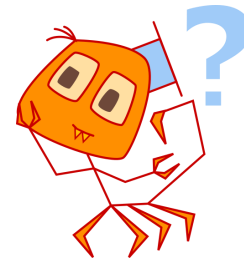
Zadanie

Za pomocą pętli for wypisz liczby od 50 do 74.



Zadanie z odpowiedzią

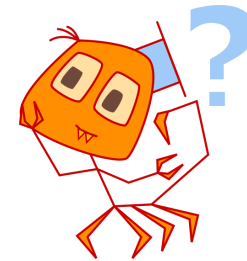
Za pomocą pętli for wypisz liczby od 50 do 74.



```
for i in range(50, 75):  
    print(i)
```


Zadanie

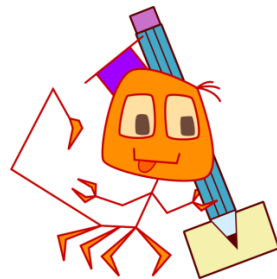
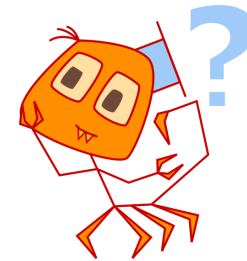
Za pomocą pętli for wypisz liczby od 1 do 10, tak, aby były w jednym wierszu.
Posłuż się zmienną, która będzie dopisywać kolejne liczby jako wartości tekstowe oraz operatora inkrementacji, który poznaliśmy wcześniej, tj. +=.
Każdą liczbę oddziel przecinkiem.



Zadanie z odpowiedzią

Za pomocą pętli for wypisz liczby od 1 do 10, tak, aby były w jednym wierszu.

Posłuż się zmienną, która będzie dopisywać kolejne liczby jako wartości tekstowe oraz operatora inkrementacji, który poznaliśmy wcześniej, tj. +=. Każdą liczbę oddziel przecinkiem.



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

Pętla while

Pętli while możemy użyć wtedy, gdy nie wiemy, ile razy ta pętla powinna się wykonać.

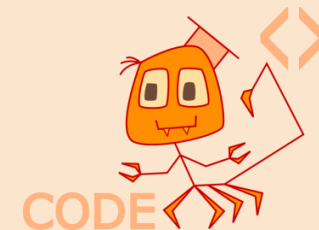
Np. za jej pomocą, możemy wyświetlać jakieś działanie lub komunikat dopóki użytkownik nie wpisze prawidłowej odpowiedzi.

Krótko mówiąc, używamy jej, by wykonywać ciągle jakąś instrukcję, dopóki podany warunek będzie prawdziwy.

To jest jej główne zadanie, ale też możemy, za jej pomocą, robić to, co robimy za pomocą pętli for :)

Pętla while potrzebuje, aby zmienna iteracyjna została zadeklarowana poza blokiem funkcji, a w bloku funkcji będziemy tą zmienną samodzielnie zwiększać lub zmniejszać.

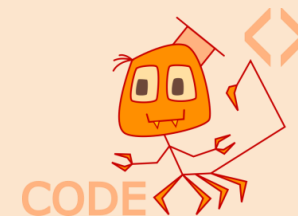
```
# wypisz liczby od 1 do 100 za pomocą pętli while
i = 1
while (i < 101):
    print(i)
    i += 1
```



Pętla while

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

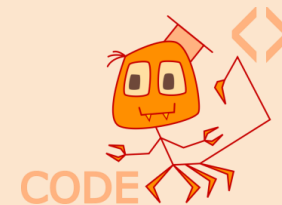
```
# powtarzaj komunikat, aż zostanie podane prawidłowe hasło
haslo = "1234"
podaneHaslo = input("Podaj hasło: ")
while(haslo != podaneHaslo):
    podaneHaslo = input("Podaj hasło: ")
print("Hasło prawidłowe")
```



Pętla while

Przykład programu, który wyświetla liczby parzyste od 0 do 50

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



Pętla while

zasada działania pętli while

while (wynik podanego warunku jest prawdziwy):

wykonuj instrukcje

możemy tutaj zmieniać wartości warunku,

ale nie musimy



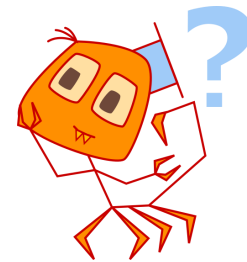
W pętli while możemy sami decydować jak zmienia się zmienna i.

Pętla while używamy, gdy nie wiemy ile razy ma się ona wykonać (np. gdy czekamy, aż ktoś poda prawidłową odpowiedź)

Pętla while wykonuje się tak długo dopóki warunek ewoluuje do prawdy (True)

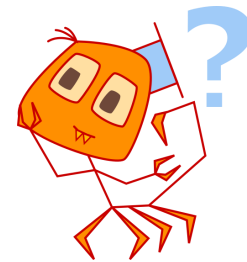
Zadanie

Wyświetl liczby od 100 do 200 za pomocą pętli while

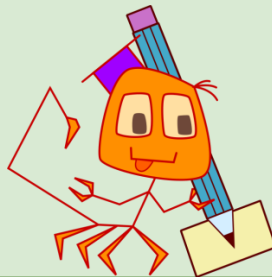


Zadanie z odpowiedzią

Wyświetl liczby od 100 do 200 za pomocą pętli while



```
liczba = 100  
  
while(liczba < 201):  
    print(liczba)  
    liczba += 1
```

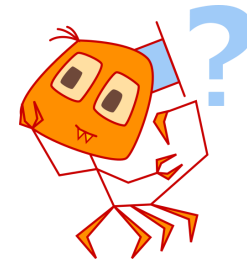


wersja z operatorem \leq

```
liczba = 100  
  
while(liczba <= 200):  
    print(liczba)  
    liczba += 1
```

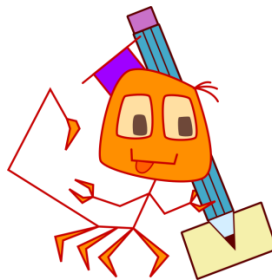
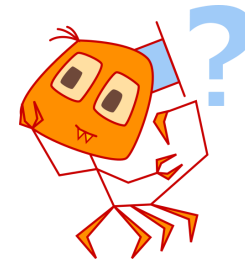

Zadanie

Wyświetl liczby od 300 do 200
za pomocą pętli while (np. 300, 299, ... , 200)



Zadanie z odpowiedzią

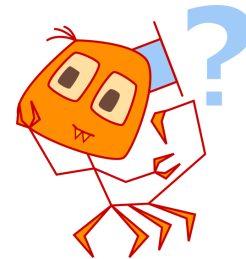
Wyświetl liczby od 300 do 200
za pomocą pętli while (np. 300, 299, ... , 200)



```
liczba = 300
while(liczba >= 200):
    print(liczba)
    liczba -= 1
```

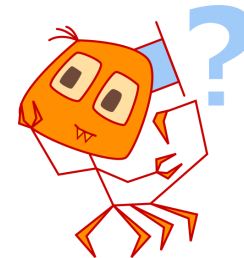
Zadanie

Wyświetl liczby od 50 do 500,
co 5 (np. 50, 55, 60, ..., 500)



Zadanie z odpowiedzią

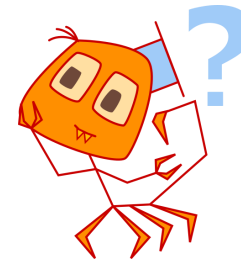
Wyświetl liczby od 50 do 500,
co 5 (np. 50, 55, 60, ..., 500)



```
liczba = 50  
while(liczba <= 500):  
    print(liczba)  
    liczba += 5
```

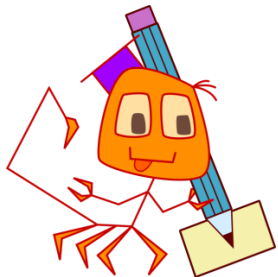
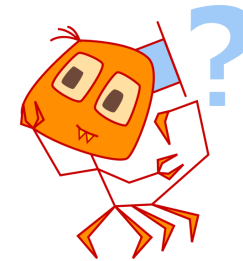
Zadanie

Napisz program, który wyświetla działanie matematyczne do czasu, do momentu, aż zostanie udzielona prawidłowa odpowiedź.



Zadanie z odpowiedzią

Napisz program, który wyświetla działanie matematyczne do czasu, do momentu, aż zostanie udzielona prawidłowa odpowiedź.



```
liczba1 = 12
liczba2 = 5
suma = liczba1 + liczba2
zadanie = str(liczba1) + " + " + str(liczba2) + " = "
odp = int(input(zadanie))
while(odp != suma):
    print("Odpowiedź niepoprawna, spróbuj ponownie")
    odp = int(input(zadanie))
print("Brawo Ty")
```

Listy

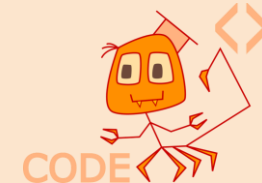
Listy służą do tworzenia kolekcji, czyli danych, które służą pewnemu celowi.

Np. jeśli chcemy przechować wszystkie imiona dzieci z klasy najłatwiej byłoby je zapisać w liście bo w przeciwnym razie musielibyśmy stworzyć osobne zmienne, np: uczen1, uczen2, uczen3, itd., a to byłoby bardzo złe. Dzięki temu, że zawrzemy imiona w jednej tablicy, będziemy mogli robić z nimi wiele operacji, np: łatwo je wypisać, posortować alfabetycznie czy sprawdzić ich ilość.

Listy tworzymy przy użyciu nawiasów kwadratowych. W podobny sposób, w innych językach programowania, tworzy się kolekcję, którą nazywamy Tablicą.

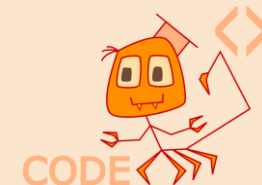
```
# tworzenie listy z imionami uczniów
```

```
uczniowie = ["Sara", "Ania", "Maciek", "Zenek"]
```



```
# tworzenie listy z liczbami totolotka
```

```
liczbyTotka = [12, 7, 29, 2, 41, 36]
```

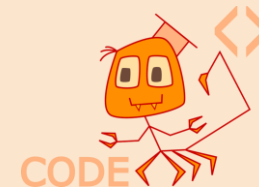


Listy - wypisanie elementów za pomocą pętli for

Zawartość listy możemy z łatwością wypisać za pomocą pętli for. W tym przypadku, zmienna `i` nie będzie liczbą porządkową, a danym elementem tablicy.

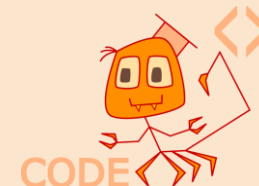
```
# wypisanie elementów listy uczniowie
```

```
uczniowie = ["Sara", "Ania", "Maciek", "Zenek"]  
for i in uczniowie:  
    print(i)
```



```
# wypisanie elementów listy liczbyTotka
```

```
liczbyTotka = [12, 7, 29, 2, 41, 36]  
for i in liczbyTotka:  
    print(i)
```



Listy - sortowanie

Zawartość listy możemy posortować na wiele sposobów, np:
liczby od najmniejszej do największej czy teksty alfabetycznie

Sortujemy odnosząc się do nazwy listy, a następnie dopisując po kropce nazwę metody - sort()

sort()

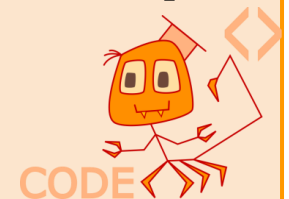
```
# sortowanie i wypisanie elementów listy uczniowie
```

```
uczniowie = ["Sara", "Ania", "Maciek", "Zenek"]  
uczniowie.sort()  
for i in uczniowie:  
    print(i)
```

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()

```
# sortowanie i wypisanie elementów  
listy liczbyTotka
```

```
liczbyTotka = [12, 7, 29, 2, 41, 36]  
liczbyTotka.sort()  
for i in liczbyTotka:  
    print(i)
```



Listy - podstawowe metody

<code>append()</code>	dodaje element na końcu listy
<code>pop()</code>	usuwa element przy podanym w nawiasie indeksie
<code>len()</code>	zwraca liczbę elementów dostępnych na liście
<code>max()</code>	zwraca największą wartość na liście
<code>min()</code>	zwraca najmniejszą wartość na liście
<code>sum()</code>	zwraca sumę elementów na liście
<code>sort()</code>	sortujemy rosnąco listę
<code>reverse()</code>	zmieniamy kolejność elementów na liście (od tyłu)



Istnieje więcej metod, które możemy wykorzystać z listami, ale na tym etapie nie musimy się nimi zajmować.

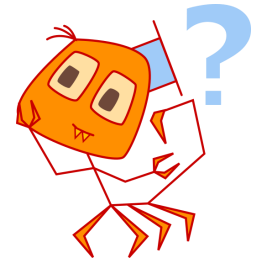
Zdecydowanie warto zapamiętać jak sortować listy i jak wyłonić jej liczbę elementów.

Pamiętaj, że wartości tekstowe, też są traktowane jako lista pojedynczych znaków.

Zadanie

Mamy listę imion: ["Basia", "Karol", "Zosia", "Damian", "Wiktoria", "Janek", "Ania"]

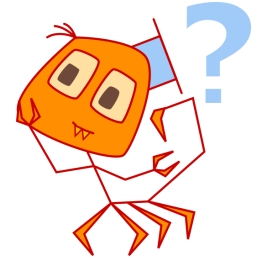
1. Wypisz wszystkie elementy listy w jednej linii oddzielone przecinkiem
2. Wypisz ilość wszystkich elementów listy
3. Posortuj imiona od Z do A i wypisz je w kolumnie



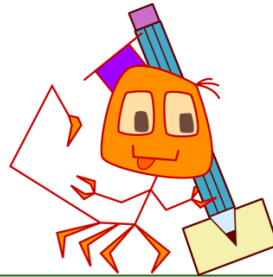
Zadanie z odpowiedzią

Mamy listę imion: ["Basia", "Karol", "Zosia", "Damian", "Wiktoria", "Janek", "Ania"]

1. Wypisz wszystkie elementy listy w jednej linii oddzielone przecinkiem
2. Wypisz ilość wszystkich elementów listy
3. Posortuj imiona od Z do A i wypisz je w kolumnie



```
# 1
imiona = ["Basia", "Karol",
"Zosia", "Damian", "Wiktoria",
"Janek", "Ania"]
liniaImion = ""
for i in imiona:
    liniaImion += i + ", "
print(liniaImion)
```



```
# 2
imiona = ["Basia", "Karol",
"Zosia", "Damian", "Wiktoria",
"Janek", "Ania"]

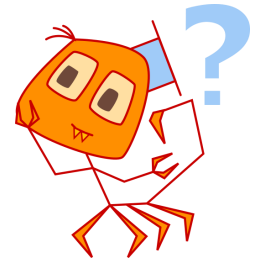
print(len(imiona))
```

```
# 3
imiona = ["Basia", "Karol",
"Zosia", "Damian", "Wiktoria",
"Janek", "Ania"]
imiona.sort()
for i in imiona:
    print(i)
```

Zadanie

Stwórz pustą listę, a następnie za pomocą pętli for i metody `append()` przypisz do niej 6 losowych liczb z przedziału od 1 do 49. To będą Twoje szczęśliwe numery do wygrania w Lotka :) Na koniec posortuj je i wypisz w jednej linii, oddzielone przecinkami.

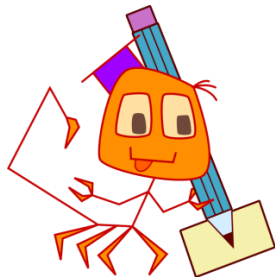
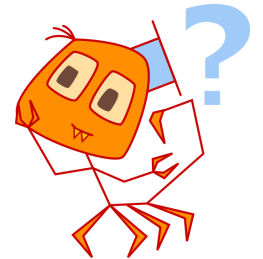
Oczywiście program nie jest idealny, ponieważ w jego prostej wersji będą zdarzać się sytuacje kiedy jedna liczba zostanie wylosowana kilkukrotnie.



Zadanie z odpowiedzią

Stwórz pustą listę, a następnie za pomocą pętli for i metody append() przypisz do niej 6 losowych liczb z przedziału od 1 do 49. To będą Twoje szczęśliwe numery do wygrania w Lotka :) Na koniec posortuj je i wypisz w jednej linii, oddzielone przecinkami.

Oczywiście program nie jest idealny, ponieważ w jego prostej wersji będą zdarzać się sytuacje kiedy jedna liczba zostanie wylosowana kilkukrotnie.



```
import random
liczby = []
for i in range(1, 7):
    losowaLiczba = random.randrange(1,50)
    liczby.append(losowaLiczba)

tekstLiczbowy = ""
liczby.sort()
for i in liczby:
    tekstLiczbowy += str(i) + ", "

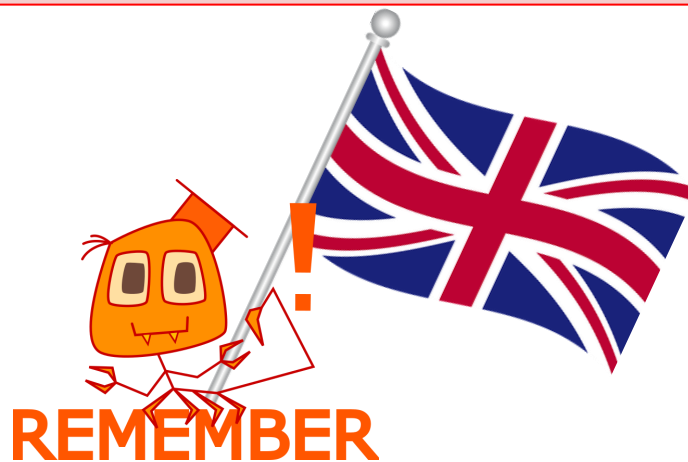
print(tekstLiczbowy)
```

Ucz się języka angielskiego

Pamiętaj!

W programowaniu, jak i w innych dziedzinach życia, jedną z najważniejszych umiejętności jest znajomość języka angielskiego.

W wolnej chwili sięgaj po książkę od anglika, a z pewnością nie pożałujesz!



Funkcje

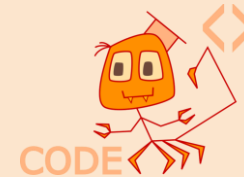
Funkcja to taki blok kodu, który odpowiada za wykonanie pewnej czynności. Uważa się, że funkcja powinna być bardzo krótka i rozwiązywać 1 konkretny problem.

Najpierw definiujemy funkcję, czyli piszemy słowo kluczowe `def`, a następnie podajemy nazwę funkcji. Po tym, 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. Kompilator 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 WyszwietlPowitanie():
    print("Hellowina")

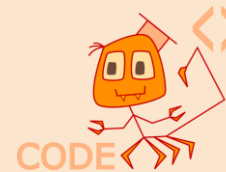
# wywołanie funkcji
WyszwietlPowitanie()
```



Funkcje

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

# wywołanie funkcji
WswietlPowitanie()
```



Oczywiście, patrząc na powyższą 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 i z jednego miejsca w kodzie, to rzeczywiście, tworzenie osobne 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 :)

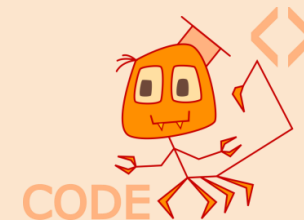
Funkcje

Jako przykład stwórzmy funkcję, która tworzy losowe działanie matematyczne, a następnie wyświetlimy 10 różnych działań za pomocą pętli. Do dzieła!

```
import random

# definicja funkcji
def StworzDzialanie():
    l1 = random.randrange(1,21)
    l2 = random.randrange(5,21)
    print(str(l1) + " + " + str(l2) + " = ")

# wywołanie funkcji w pętli
for i in range(1,11):
    StworzDzialanie()
```

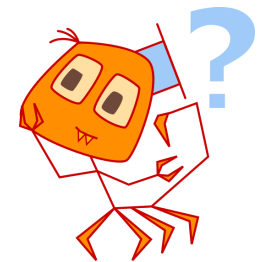


Zadanie

Stwórz funkcję, która pyta użytkownika o imię, a następnie wyświetla powitanie z podaniem wpisanego imienia.

Na koniec oczywiście wywołaj funkcję.

Czy masz pomysł na nazwę funkcji?

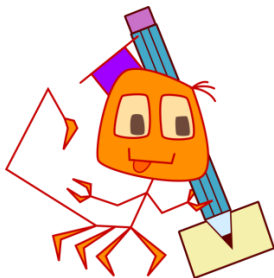
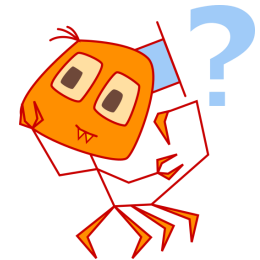


Zadanie z odpowiedzią

Stwórz funkcję, która pyta użytkownika o imię, a następnie wyświetla powitanie z podaniem wpisanego imienia.

Na koniec oczywiście wywołaj funkcję.

Czy masz pomysł na nazwę funkcji?



```
def PowitajUzytkownika():  
  
    imie = input("Podaj imie: ")  
  
    print("Witaj, " + imie)
```

```
PowitajUzytkownika()
```

Funkcje

W poprzednim zadaniu stworzyliśmy jedną funkcję, która tak naprawdę odpowiada za 2 rzeczy.

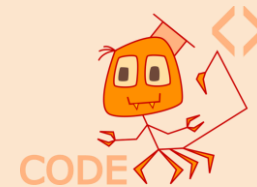
1 Pyta użytkownika o imię

2 Wyświetla powitanie

Moglibyśmy zostawić to w ten sposób, ale chciałbym abyśmy ją podzielili na 2 mniejsze funkcje. Dzięki temu nauczymy się jak zwracać wartość z funkcji, a robimy to za pomocą słowa kluczowego **return**.

Poniższy przykład zrobimy w ten sposób, że funkcję pytającą o imię, wywołamy z drugiej funkcji.

```
def ZapytajOImie():  
    imie = input("Podaj imie: ")  
    return imie  
  
def WyświetlPowitanie():  
    print("Witaj, " + ZapytajOImie())  
  
WyświetlPowitanie()
```



Funkcje z parametrem

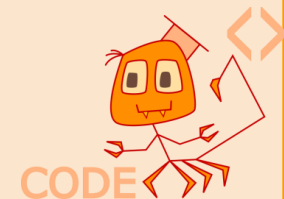
Funkcje stają się jeszcze bardziej uniwersalne, kiedy dołączymy do nich parametry.

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.

```
def WyświetlWiadomosc(trescWiadomosci):  
    print(trescWiadomosci)
```

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



Funkcje z 2 parametrami

W tym przykładzie stworzymy funkcję z dwoma parametrami.

Jej zadaniem będzie zwracanie losowej liczby z różnych przedziałów. Przedziały te będziemy sami zmieniać podczas wywołania funkcji, za pomocą argumentów.

Pamiętaj, że jeżeli funkcja została stworzona z dwoma parametrami, to podczas jej wywołania musimy podać 2 argumenty.

```
import random
```

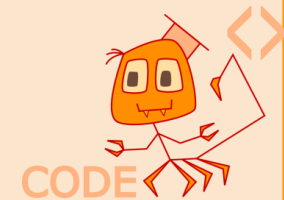
```
def GenerujLiczbe(min,max):
```

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

```
    return losowaLiczba
```

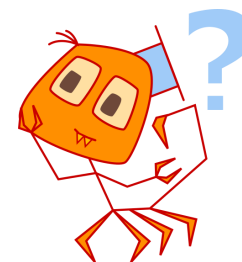
```
print(GenerujLiczbe(10,21))      # wyświetla liczbę od 10 do 20
```

```
print(GenerujLiczbe(100,1001))  # wyświetla liczbę od 100 do 1001
```



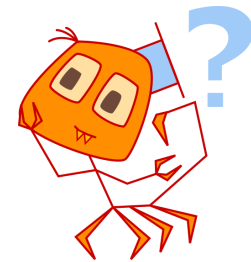
Zadanie - Znajdowanie największej liczby

Napisz program, który pobiera 3 liczby, a następnie zwraca tą największą.



Zadanie z odpowiedzią - Znajdowanie największej liczby

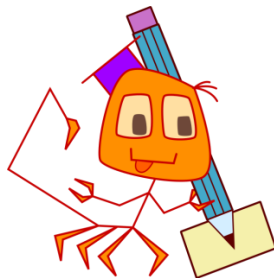
Napisz program, który pobiera 3 liczby, a następnie zwraca tą największą.



```
liczba1 = float(input("Podaj 1 liczbę: "))
liczba2 = float(input("Podaj 2 liczbę: "))
liczba3 = float(input("Podaj 3 liczbę: "))

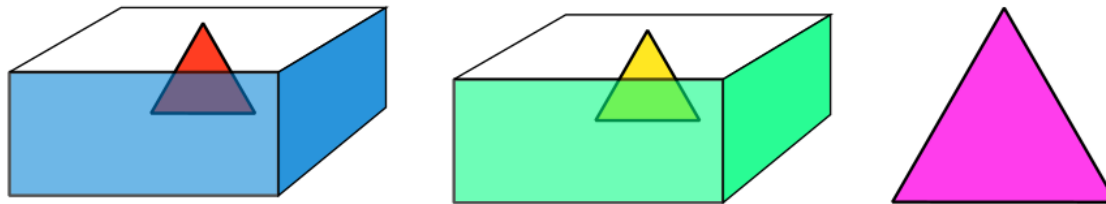
if(liczba1 > liczba2 and liczba1 > liczba3):
    max = liczba1
elif(liczba2 > liczba1 and liczba2 > liczba3):
    max = liczba2
else:
    max = liczba3

print("max = ", max)
```



Zmienne globalne i lokalne

W programowaniu mamy do czynienia z zakresem dostępności, tzw. scope. Chodzi o to, że niektóre elementy składowe (zmienne lub funkcje) są ukryte dla innych bloków kodu. Spróbujmy to zrozumieć na zasadzie pudełek.



Wyobraźmy sobie, że pudełka i trójkąty mają oczy :)

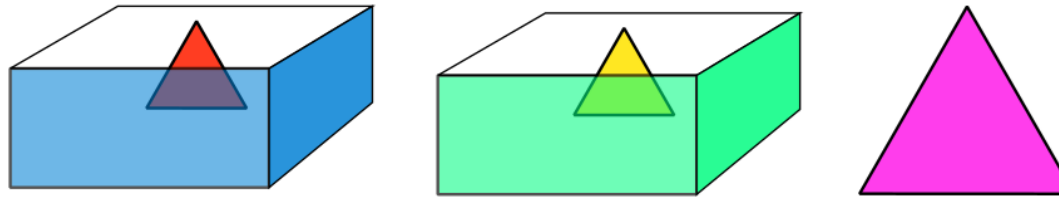
Niebieskie pudełko ma dostęp do czerwonego trójkąta bo jest wewnątrz niego. Niebieskie pudełko również ma dostęp do zielonego pudełka i różowego trójkąta bo "widzi" je, tymi, swoimi oczami :) Niebieskie pudełko jednak nie ma dostępu do żółtego trójkąta, ponieważ jest on, wewnątrz zielonego pudełka i jest poza jego zasięgiem widzenia.

Żółty trójkąt ma zasięg lokalny i nie ma szans, by go ktokolwiek dostrzegł, poza zielonym pudełkiem.

Czerwony trójkąt nigdy nie będzie miał dostępu do żółtego trójkąta, ponieważ oba obiekty mają tzw. scope lokalny, wewnętrzny.

Problem pojawia się, gdy zmienne lokalne i globalne mają takie same nazwy. Oczywiście są do tego odpowiednie narzędzia, ale na tym etapie, nie będziemy tego omawiać. W związku z tym, najlepiej starać się, tworzyć różne nazwy zmiennych lokalnych i globalnych.

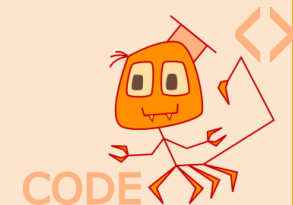
Zmienne globalne i lokalne



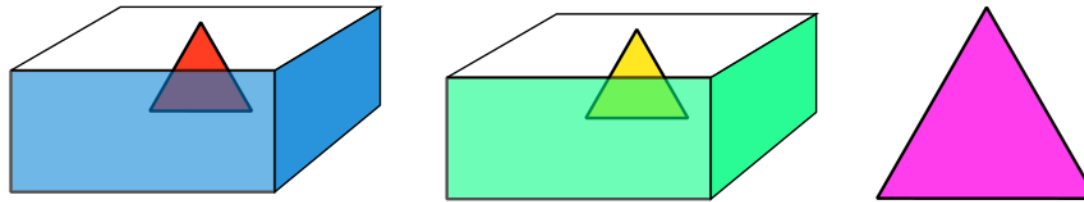
```
pudelko = "Zielone pudełko" # zmienna globalna

def SpojrzZPudelkaNiebieskiego():
    trojkat = "Czerwony trójkąt" # zmienna lokalna (wewnątrz funkcji)
    print(trojkat)                # wyświetla zmienną lokalną
    print(pudelko)                # wyświetla zmienną globalną
```

```
SpojrzZPudelkaNiebieskiego()
```



Zmienne globalne i lokalne



Aby zmieniać wartość w zmiennej globalnej, musimy wewnątrz funkcji powiedzieć kompilatorowi, że ta zmienna ma być tą zmienną globalną. Robimy to za pomocą słowa kluczowego **global**.

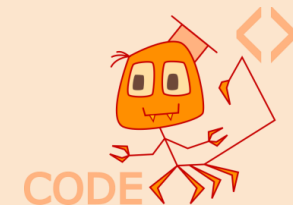
Przykład poniżej:

```
pudelko = "Zielone pudełko" # zmienna globalna

def SpojrzZPudelkaNiebieskiego():
    global pudelko
    pudelko = "Inne pudełko"

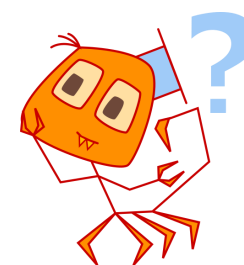
SpojrzZPudelkaNiebieskiego()

print(pudelko)
```



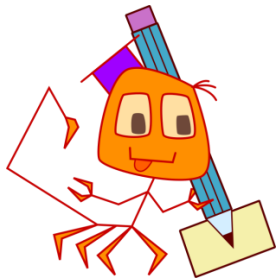
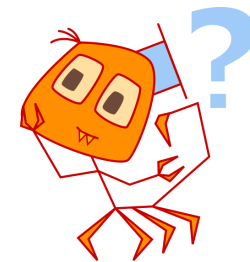
Zadanie

Napisz program, który pobierze podstawę i wysokość trójkąta, a następnie wyświetli jego pole. Zrób to bez tworzenia własnych funkcji.



Zadanie z odpowiedzią

Napisz program, który pobierze podstawę i wysokość trójkąta, a następnie wyświetli jego pole. Zrób to bez tworzenia własnych funkcji.



```
podstawa = float(input("Podaj podstawę trójkąta: "))  
wysokosc = float(input("Podaj wysokość trójkąta: "))  
pole = podstawa * wysokosc * 0.5  
print("Pole wynosi: ", pole)
```

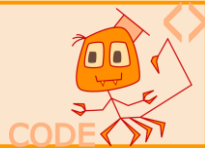
Modulo %

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 %

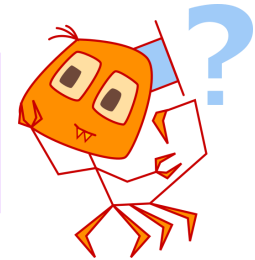
```
print(12 % 5)
```



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

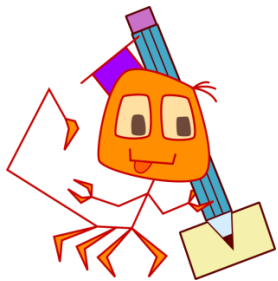
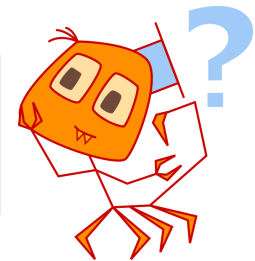
Zadanie z odpowiedzią z modulo

Napisz program, który sprawdzi czy dana liczba jest podzielna przez 3.



Zadanie z modulo

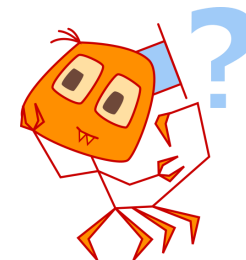
Napisz program, który sprawdzi czy dana liczba jest podzielna przez 3.



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

Zadanie

Napisz program, który pobierze podstawę i wysokość trójkąta, a następnie wyświetli jego pole. Zrób to za pomocą 4 osobnych funkcji :
`pobierzPodstawe()`, `pobierzWysokosc()`, `obliczPole()`, `wyswietlPole()`

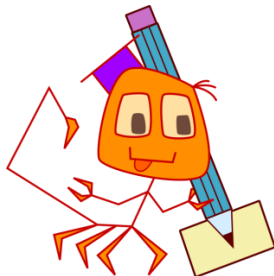


Zadanie z odpowiedzią

Napisz program, który pobierze podstawę i wysokość trójkąta, a następnie wyświetli jego pole. Zrób to za pomocą 4 osobnych funkcji :
pobierzPodstawe(), pobierzWysokosc(), obliczPole(), wyswietlPole()



```
def pobierzPodstawe():  
    return float(input("Podaj podstawę trójkąta: "))  
  
def pobierzWysokosc():  
    return float(input("Podaj wysokość trójkąta: "))  
  
def obliczPole():  
    podstawa = pobierzPodstawe()  
    wysokosc = pobierzWysokosc()  
    pole = podstawa * wysokosc * 0.5  
    return pole  
  
def wyswietlPole():  
    print("Pole trójkąta wynosi: ", obliczPole())  
  
wyswietlPole()
```

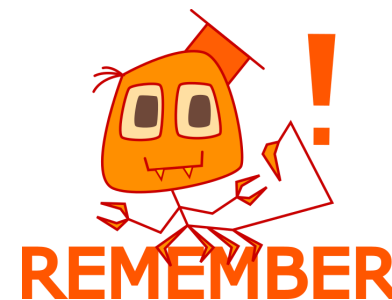


Prostota programowania

W programowaniu opieramy się na podstawowych operacjach i składowych, których znaczną część poznaliśmy w poprzednich lekcjach. Oczywiście mamy jeszcze wiele do nauki w przyszłości, ale na ten moment, konstrukcje, które poznaliśmy pozwolą nam stworzyć wiele interesujących projektów.

Po opanowaniu podstaw języka, należy poświęcić chwilę, by nauczyć się zasad zarządzania projektem, ponieważ wraz ze wzrostem ilości linii kodu, można łatwo się pogubić :)

W międzyczasie, oczywiście, też musimy się nauczyć myśleć jak komputer, ponieważ większość stanów czy decyzji, jest opartych na podstawie liczb i zmiennych typu bool. Po rozwiązaniu wszystkich zadań i przykładów z pewnością wszystko stanie się bardziej przejrzyste.



Prostota programowania

Pamiętaj!

Internet, to Twój przyjaciel.

Zagadnienia, które są nadal niezrozumiałe warto prześledzić na różnorodnych stronach internetowych i forach.

Postawienie sobie problemu do rozwiązania, a następnie wyszukiwanie rozwiązań w internecie jest bardzo dobrym sposobem nauki. Szczerze Was do tego zachęcam.



Sekcja z zadaniami

Rozpoczynamy sekcję z trudniejszymi zadaniami.

Spróbuj wykonać je samodzielnie wspomagając się internetem. Oczywiście, odpowiedzi następują po zadaniu więc możesz sobie je prześledzić, ale bardzo cię proszę byś spróbował/a zrobić je samodzielnie. Pamiętaj też, że istnieje wiele różnych dróg na rozwiązanie jednego zadania.

Czytanie kodu jest tak samo ważne jak jego pisanie. Staraj się śledzić programy w tym ebooku i w internecie i zrozumieć zasadę działania danego programu.

Baw się kodem, zmieniaj różne rzeczy, eksperymentuj, bo dzięki temu rozwijasz swoje umiejętności.

I na koniec, nie przejmuj się, jeśli część z tych zadań okażą się zbyt trudne. Prześledź odpowiedzi i próbuj pisać przedstawione w ebooku zadania kilkakrotnie, aż ostatecznie będziecie w stanie je wyrecytować z pamięci :) :)

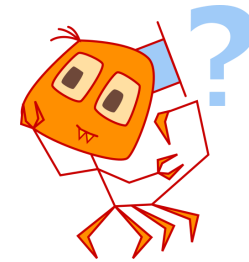
Do dzieła!

Zadanie

Za pomocą pętli for wypisz liczby od 1 do 10, tak, aby były w jednym wierszu.

Każdą liczbę oddziel przecinkiem, ale w taki sposób, aby po ostatniej liczbie zamiast przecinka została wstawiona kropka.

Wskazówka: Użyj instrukcji warunkowej.

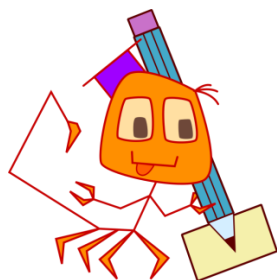
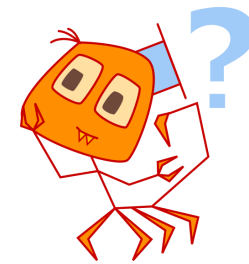


Zadanie z odpowiedzią

Za pomocą pętli for wypisz liczby od 1 do 10, tak, aby były w jednym wierszu.

Każdą liczbę oddziel przecinkiem, ale w taki sposób, aby po ostatniej liczbie zamiast przecinka została wstawiona kropka.

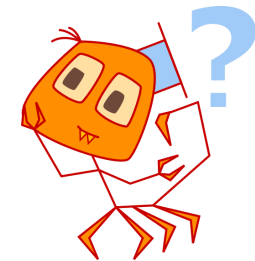
Wskazówka: Użyj instrukcji warunkowej.



```
tekstLiczby = ""  
for i in range(1,11):  
    if(i != 10):  
        tekstLiczby += str(i) + ", "  
    else:  
        tekstLiczby += str(i) + ". "  
print(tekstLiczby)
```

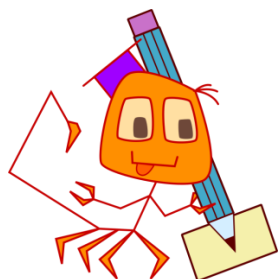
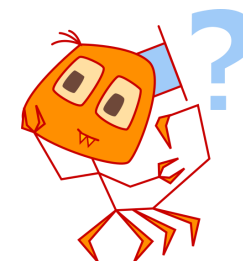

Zadanie - Obliczanie ceny po obniżce

Napisz program, który pobierze cenę towaru, a następnie wartość procentową obniżki. Następnie wyświetli kwotę po obniżce.



Zadanie z odpowiedzią - Obliczanie ceny po obniżce

Napisz program, który pobierze cenę towaru, a następnie wartość procentową obniżki. Następnie wyświetli kwotę po obniżce.

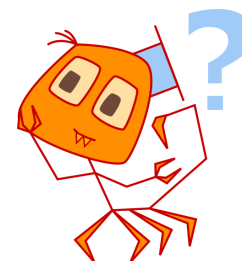


```
cenaOryginalna = float(input("Podaj cenę przed obniżką: "))
obnizka = float(input("Podaj wartość procentową obniżki: "))
cenaPoObnizce = cenaOryginalna - (cenaOryginalna * obnizka / 100)
print("Cena po obniżce = ", cenaPoObnizce)
```

Zadanie

Napisz program, który sprawdza czy z podanych długości można zbudować trójkąt.

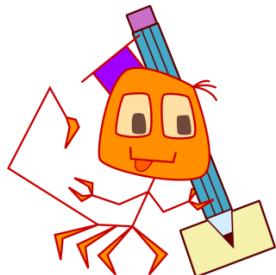
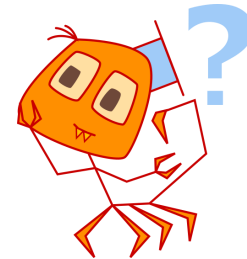
Aby zbudować trójkąt, suma długości dwóch krótszych boków musi być większa od najdłuższego boku.



Zadanie z odpowiedzią

Napisz program, który sprawdza czy z podanych długości można zbudować trójkąt.

Aby zbudować trójkąt, suma długości dwóch krótszych boków musi być większa od najdłuższego boku.



```
bokiTrojkata = []

for i in range(1,4):
    bokiTrojkata.append(float(input("Bok nr " + str(i) + ": ")))

bokiTrojkata.sort()
bokiTrojkata.reverse()

if(bokiTrojkata[0] < (bokiTrojkata[1] + bokiTrojkata[2])):
    print("Można zbudować trójkąt o podanych bokach.")
    print("Obwód trójkąta wynosi: " , sum(bokiTrojkata))
else:
    print("Nie można zbudować trójkąta")
```

Zadanie



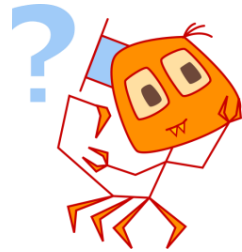
Stwórz program do losowanie 6 numerów lotto.
Tym razem zrób to w ten sposób, by wybrane liczby nie mogły się powtarzać.

Podpowiedź.

Na początek możemy stworzyć tablicę wszystkich liczb od 1 do 49.

Następnie losujemy jedną z tych liczb i dodajemy do nowej tablicy o nazwie `wybraneLiczby`. Po dodaniu usuwamy element z tablicy wszystkich numerów za pomocą metody `pop()`.

Zadanie z odpowiedzią



Stwórz program do losowanie 6 numerów lotto.
Tym razem zrób to w ten sposób, by wybrane liczby nie mogły się powtarzać.

Podpowieź.

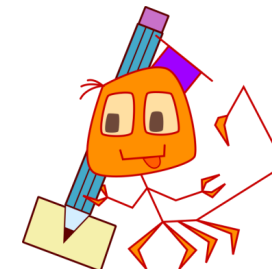
Na początek możemy stworzyć tablicę wszystkich liczb od 1 do 49.

Następnie losujemy jedną z tych liczb i dodajemy do nowej tablicy o nazwie wybraneLiczby. Po dodaniu usuwamy element z tablicy wszystkich numerów za pomocą metody pop().

```
import random
wszystkieLiczby = []
for i in range(1, 50, 1):
    wszystkieLiczby.append(i)

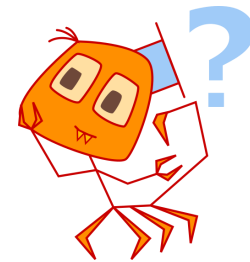
wybraneLiczby = []
for i in range(1, 7, 1):
    losowaLiczba = random.randrange(1, len(wszystkieLiczby))
    wybraneLiczby.append(wszystkieLiczby[losowaLiczba - 1])
    wszystkieLiczby.pop(losowaLiczba - 1)

tekstLiczby = ""
for i in wybraneLiczby:
    tekstLiczby += str(i) + ", "
print(tekstLiczby)
```



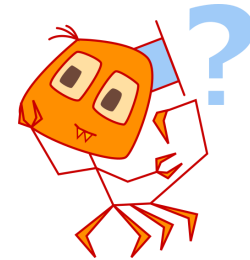
Zadanie - Oblicz, który produkt jest bardziej opłacalny

Często robiąc zakupy zastanawiamy się, który produkt opłaca się kupić, np: 1000g po 34.99zł czy 500g po 18.99zł?
Napiszmy program, który porówna ceny za 1g i wyświetli tą bardziej opłacalną opcję.

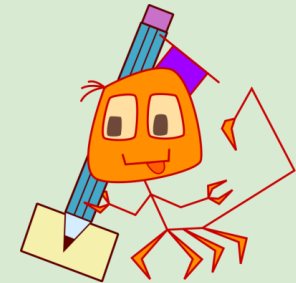


Zadanie z odpowiedzią - Oblicz, który produkt jest bardziej opłacalny

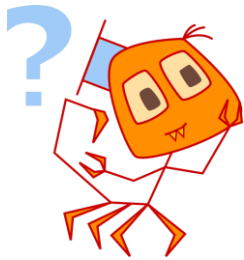
Często robiąc zakupy zastanawiamy się, który produkt opłaca się kupić, np: 1000g po 34.99zł czy 500g po 18.99zł?
Napiszmy program, który porówna ceny za 1g i wyświetli tą bardziej opłacalną opcję.



```
cena1 = float(input("Podaj cenę produktu 1 w zł: "))
waga1 = float(input("Podaj wagę produktu 1 w gramach: "))
cena2 = float(input("Podaj cenę produktu 2 w zł: "))
waga2 = float(input("Podaj wagę produktu 2 w gramach: "))
cenaZaGram1 = cena1 / waga1
cenaZaGram2 = cena2 / waga2
oplacalnaOpcja = ""
if(cenaZaGram1 < cenaZaGram2):
    oplacalnaOpcja = "Kup " + str(waga1) + " gram za " + str(cena1) + "zł."
else:
    oplacalnaOpcja = "Kup " + str(waga2) + " gram za " + str(cena2) + "zł."
print(oplacalnaOpcja)
```



Zadanie - Test matematyczny 5 pytań



Napisz program, który wyświetli pewną ilość zadań matematycznych.

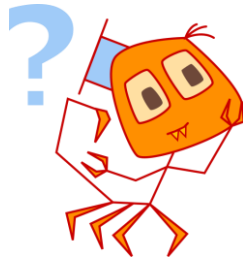
Ilość tych zadań może wynosić np.5, ale powinna być łatwo zmieniana.

Po podaniu odpowiedzi, przechodzimy do kolejnego zadania.

Przy udzieleniu poprawnej odpowiedzi dostajemy punkt.

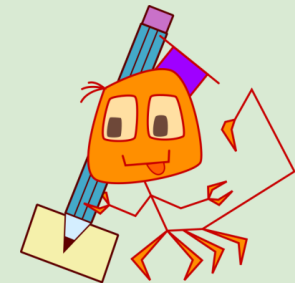
Po udzieleniu wszystkich odpowiedzi, na koniec testu pokazuje nam się wynik poprawnie udzielonych odpowiedzi.

Zadanie z odpowiedzią - Test matematyczny 5 pytań

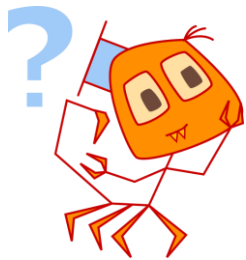


Napisz program, który wyświetli pewną ilość zadań matematycznych. Ilość tych zadań może wynosić np.5, ale powinna być łatwo zmieniana. Po podaniu odpowiedzi, przechodzimy do kolejnego zadania. Przy udzieleniu poprawnej odpowiedzi dostajemy punkt. Po udzieleniu wszystkich odpowiedzi, na koniec testu pokazuje nam się wynik poprawnie udzielonych odpowiedzi.

```
import random
liczbaPunktow = 0
iloscZadan = 3;
def GenerujLosowaLiczbe():
    return random.randrange(1,21,1)
def TworzZadanie():
    l1 = GenerujLosowaLiczbe()
    l2 = GenerujLosowaLiczbe()
    zadanie = str(l1) + " + " + str(l2) + " = "
    odp = int(input(zadanie))
    if(odp == l1 + l2):
        global liczbaPunktow
        liczbaPunktow += 1
for i in range(1,iloscZadan+1):
    TworzZadanie()
print("Poprawnych odpowiedzi: " + str(liczbaPunktow) + " / " + str(iloscZadan))
```



Zadanie - Gra Zgadnij liczbę



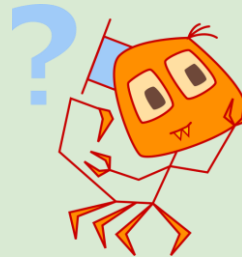
Napisz grę w której będziemy starali się odgadnąć losowo wybraną liczbę z przedziału od 1 do 100.

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.

Zadanie z odpowiedzią - Gra Zgadnij liczbę

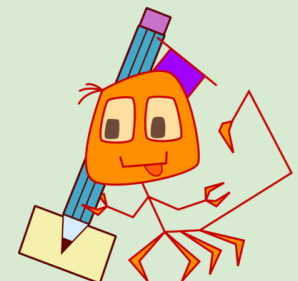
```
import random
szukanaLiczba = random.randrange(1,101)
ostatniaOdpowiedz = 0
iloscProb = 0;
trescPytania = ""
def ZwiekszIloscProb():
    global iloscProb
    iloscProb += 1
def GenerujTrescPytania():
    global trescPytania
    global ostatniaOdpowiedz
    if(ostatniaOdpowiedz == 0):
        trescPytania = "Podaj liczbę od 1 do 100: "
    elif(ostatniaOdpowiedz > szukanaLiczba):
        trescPytania = "Podaj mniejszą liczbę: "
        ZwiekszIloscProb()
    else:
        trescPytania = "Podaj większą liczbę: "
        ZwiekszIloscProb()
def PodajOdpowiedz():
    global ostatniaOdpowiedz
    GenerujTrescPytania()
    ostatniaOdpowiedz = int(input(trescPytania))
while(ostatniaOdpowiedz != szukanaLiczba):
    PodajOdpowiedz()
print("Brawo! Zgadłeś! Ilość prób: " + str(iloscProb))
```



Napisz grę 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.

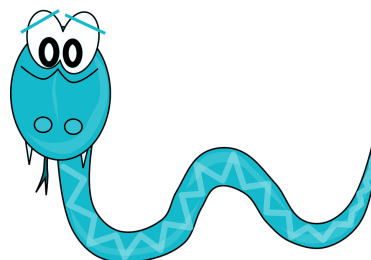


To już jest koniec....

ale tylko treści zawartych w tym ebooku.

Nasza przygoda dopiero się rozpoczęła, a Ty drogi czytelniku możesz dalej rozwijać swoją wiedzę i umiejętności. Jeśli miałbym Ci cokolwiek doradzić to powiedziałbym, tak:

- skup swoją uwagę na nauce języka angielskiego
- pomyśl nad projektami (problemami), które można rozwiązać za pomocą programowania. To mogą być aplikacje, które są już dostępne na rynku, ale chodzi o to, byś sam nauczył się je tworzyć, bo dzięki temu nauczyć się bardzo wiele.
- spróbuj poszukać projektów w internecie (fora internetowe, youtube, grupy na fb)



Simba, remember!

Programowanie to długi proces, nie załamujcie się tym, że tak wiele jest jeszcze do nauki. Nie porównujcie się do osób, które już wymiatają w kodowaniu. Oni przeszli już długą drogę, by być w tym miejscu.

Doceniajcie to, co udało Wam się już nauczyć. Zobaczcie, dzięki przeczytaniu tej publikacji wiecie o programowaniu znacznie więcej od osób, które nie miały z programowaniem styczności. Każdy przeczytany artykuł i ukończony projekt sprawia, że stajecie się coraz lepsi.

Życzę Wam wytrwałości i tego, by programowanie, było dla Was fantastyczną przygodą.

Zaglądajcie do Nas czasami

www.pikademia.pl

Do usłyszenia :)