giganci programowania

python

Najczęstsze komendy

Wstęp do programowania

Python

Witaj w materiałach pomocniczych przygotowanych specjalnie dla Ciebie. Znajdziesz tutaj najczęstsze kody i komendy potrzebne przy programowaniu. Zapisz, wydrukuj i korzystaj! Powodzenia!

Najczęściej używane komendy:

Zmienne

Wartości liczbowe i tekstowe w programie przechowywane są w zmiennych. Każda zmienna posiada swoją unikalną nazwę i aby się pobrać z niej wartość lub przypisać jej nową, należy odwołać się do jej nazwy. Każda zmienna musi zostać zainicjalizowana, czyli należy ją zadeklarować i przypisać jej wartość początkową:

Deklaracja zmiennej:
nazwa zmiennej = wartość zmiennej, np:

```
liczba = 12
tekst = "Giganci Programowania"
wartosc_logiczna = True
```

Raz stworzoną zmienną, możemy modyfikować i przypisać jej nową wartość:

```
zmienna = 12
zmienna = 15
```

Operatory matematyczne

Na zmiennych można wykonywać operacje matematyczne, tak jak na liczbach:

```
liczba_a = 10
liczba_b = 15

suma = liczba_a + liczba_b
roznica = liczba_a - liczba_b
dzielenie = liczba_a / liczba_b
mnozenie = liczba_a * liczba_b
```

Operatory logiczne

Na zmiennych można wykonywać operacje logiczne. Wynikiem tych operacji jest zawsze wartość logiczna, czyli prawda lub fałsz. Używa się ich do sprawdzenia prawdziwości jakiegoś zdania, inaczej czy spełniony został jakiś warunek.

Przykłady:
Liczba 12 jest większa od liczby 5, da nam prawdę, czyli **True.**
Liczba 20 jest mniejsza od liczby 5, da nam fałsz, czyli **False**.

Możemy to zapisać w programie za pomocą znaków porównania
> większę
>= większę lub równe
< mniejsze
<= mniejsze lub równe
== równe (nie mylić ze znakiem = -jest to operator przypisania)
!= różne (przeciwieństwo ==)

Kod python:

```
liczba_a = 10
liczba_b = 15

wynik = liczba_a > liczba_b  # False
wynik = liczba_a < liczba_b  # True
wynik = liczba_a != liczba_b # True
wynik = liczba_a == liczba_b # False
```

Dodatkowymi operacjami logicznymi są słowa **and** i **or** i oznaczają one **i** oraz **lub.**

Wykorzystuje się je do łączenia ze sobą kilku różnych warunków np:
liczba a jest większa liczby b, lub jest równa liczbie c.
Kod python:

```python
liczba_a = 10
liczba_b = 15
liczba_c = 10

wynik = liczba_a > liczba_b or liczba_a == liczba_c # True
```

Instrukcje warunkowe

Instrukcje warunkową, stosujemy do sterowania przebiegiem programu. Po sprawdzeniu warunku (wartość logiczna), zależnie czy został on spełniony czy nie, wykonujemy inny fragment kodu.

Przykład:
Jeżeli zmienna a jest większa od 5 to przypisz do zmiennej b wartość 5, w przeciwnym razie przypisz do zmiennej c wartość 5.
Kod python:

```python
liczba_a = 7
liczba_b = 0
liczba_c = 0

if liczba_a > 5:
    liczba_b = 5
else:
    liczba_c = 5
```

Możemy łączyć ze sobą kilka różnych instrukcji warunkowych, wtedy jeżeli jeden warunek nie będzie spełniony w danym ciągu warunków, program przechodzi do kolejnego.:

```
liczba_a = 7
liczba_b = 0
liczba_c = 0

if liczba_a < 2:
    liczba_b = 2
elif liczba_a < 4:
    liczba_c = 2
elif liczba_a < 6:
    liczba_b = 4
else:
    liczba_c = 5
```

Pętle

W programowaniu wykorzystujemy pętle aby powtórzyć jakiś fragment kodu kilka razy.
W języku python rozróżniamy 2 rodzaje pętli: **while** i **for**.

Pętlę **while** stosujemy jeżeli chcemy powtarzać coś, dopóki jakiś warunek(wartość logiczna) nie został spełniony. Prosty przykład programu, w którym trzeba odgadnąć liczbę:

```
sekretna_liczba = 13
wpisana_liczba = 0

while sekretna_liczba != wpisana_liczba:
    wpisana_liczba = int(input("Zgadnij liczbę: "))

print("Gratulacje!")
```

A oto wygląd konsoli:

```
Zgadnij liczbę: 2
Zgadnij liczbę: 4
Zgadnij liczbę: 9
Zgadnij liczbę: 12
Zgadnij liczbę: 13
Gratulacje!
```

Pętle **while** wykorzystuje się np. w grach, aby cały czas rysować elementy gry na ekranie, dopóki jej nie wyłączymy

Pętlę **for** należy użyć gdy chcemy wykonać fragment programu, określoną liczbę razy, lub chcemy wykonać fragment programu dla każdego elementu w jakimś kontenerze(np. liście).

Przykłady:

Wypisanie 10 liczb na ekranie:

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

Wypisze na ekranie liczby 0 - 9:

```
0
1
2
3
4
5
6
7
8
9
```

Możemy także odnieść się to listy:

```
owoce = ["banan", "jabłko", "kiwi", "arbuz", "mandarynka"]

for owoc in owoce:
    print(owoc)
```

Ten program wypiszę na ekranie wszystkie elementy listy

```
banan
jabłko
kiwi
arbuz
mandarynka
```

Listy

Lista to specjalny rodzaj zmiennej w języku python, może ona przetrzymywać więcej niż jedną wartość, a każda z nich ma swój unikalny indeks, zaczynający się od 0 i rosnący.
Aby zapisać różne wartości do tablicy, należy po przecinku wpisać je w nawiasy kwadratowe:

```
nieparzyste = [1,3,5,7,9,11]
samogloski = ["a", "e", "i", "o", "u", "y"]
```

Aby wyświetlić konkretny element tablicy, przy odnoszeniu się do zmiennej, należy podać indeks elementu w nawiasie kwadratowym:

```
print(nieparzyste[0])
print(nieparzyste[3])
print(samogloski[0])
print(samogloski[2])
print(samogloski[5])
```

```
1
7
a
i
y
```

Należy pamiętać aby nie podawać indeksu większego niż ilość elementów, ponieważ spowoduje to błąd

```
print(samogloski[12])
```

```
IndexError: list index out of range
```

Aby dodać nowy element tablicy, należy użyć metody **append**.

```
print(nieparzyste)
nieparzyste.append(19)
print(nieparzyste)
```

Aby usunąć element tablicy o konkretnym indeksie należy użyć metody **pop.**

```
print(samogloski)
samogloski.pop(2)
print(samogloski)
```

```
['a', 'e', 'i', 'o', 'u', 'y']
['a', 'e', 'o', 'u', 'y']
```

Metody

Metody to specjalne kawałki kody wykonujące jakąś operację, jednak muszą one zostać wywołane w kodzie. Stosuje się je zazwyczaj, jeśli dany kod ma zostać wywołany więcej niż jeden raz, np. do jakichś obliczeń.

Tworząc metodę zawsze zaczynamy od napisania słowa **def** następnie nazwy metody i parę nawiasów okrągłych. Nawiasy mogą być puste lub posiadać dowolną ilość parametrów. Parametr działa jak zmienna, która istnieje tylko wewnątrz danej funkcji i nie można się do niej odnieść poza nią. W ciele funkcji piszemy kod, funkcja może ale nie musi zwracać jakąś wartość. Do zwrócenia wartości służy słowo **return**.

Przykład, funkcja obliczająca pole Koła:

```
def pole_kola(r):
    pole = 3.14 * r * r
    return pole

print(pole_kola(3))
print(pole_kola(5))
print(pole_kola(21))
print(pole_kola(1))
```

Klasy

Klasa to bardziej zaawansowany element programowania. Jest to pewnego rodzaju szablon, na podstawie którego tworzy się później obiekty. W klasie definiujemy właściwości takie jak zmienne czy metody obiektu i każdy utworzony obiekt danej klasy będzie posiadał te właśnie elementy unikalne dla danego obiektu.

Aby stworzyć klasę należy użyć słowa **class**, następnie podać nazwę klasy i nawiasy okrągłe i dwukropek:
Poniżej można stworzyć metody danej klasy. Klasy posiadają wbudowane metody, które działają trochę inaczej niż zwykłe metody. Jedną z nich jest konstruktor. Ta metoda wywołuje się automatycznie przy tworzeniu obiektu. Aby stworzyć konstruktor należy stworzyć metodę o nazwie **__init__**. Jako pierwszy parametr każdej metody w klasie, należy podać słowo kluczowe **self**.
Przykład klasy:

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

    def daj_glos(self):
        print(f"{self.imie} ma {self.wiek} lat i szczeka")
```

Tworzenie obiektu i wywołanie metody dla każdego obiektu:

```
pies1 = Pies("Azor", 5)
pies2 = Pies("Kajtek", 12)

pies1.daj_glos()
pies2.daj_glos()
```

```
Azor ma 5 lat i szczeka
Kajtek ma 12 lat i szczeka
```

Przydatne metody:

```
#komentarz rozpoczynamy znakiem #

#wypisanie tekstu na ekranie
print("tekst")

#wczytanie tekstu z klawiatury
zmienna = input("Podaj liczbę: ")

#zamiana tekstu na liczbę:
calkowita = int(zmienna)
```

Pygame

Wczytanie modułu

```
import pygame
```

Inicjalizacja modułu:

```
pygame.init()
```

Wczytanie obrazu z pliku:

```
obraz = pygame.image.load("ścieżka_do_pliku/obraz.png")
```

Utworzenie obiektu ekranu o danych rozmiarach w pixelach(szerokość x wysokość:

```
ekran = pygame.display.set_mode([800, 600])
```

Rysowanie obrazu na ekranie w danych współrzędnych (x, y):

```
ekran.blit(obraz_tla, (12, 100))
```

Czyszczenie ekranu:

```
pygame.display.flip()
```

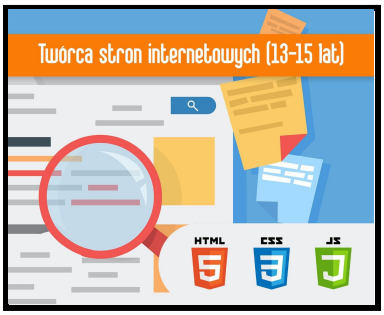
Wyłączenie moduły pygame:

```
pygame.quit()
```

giganci programowania

python™

Chcesz stworzyć jeszcze więcej gier i wypróbować nowe możliwości? Zapraszam Cię na krótkie kursy, w których możesz uczestniczyć w dowolnym czasie. Nowe gry, kolejne wyzwania - dla każdego, kto w grach i programowaniu chce być mistrzem. Zajrzyj, sprawdź, skorzystaj!



Zapisz się już dziś!

https://www.giganciprogramowania.edu.pl/

sekretariat@giganciprogramowania.edu.pl

tel: 22 112 10 63