Najczęstsze polecenia

Minecraft z Pythonem

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

Zmienne

Zmienna jest to taki element języka, który pozwala na przechowywanie różnego typu wartości. Można inaczej powiedzieć, że zmienna jest to takie pudełeczko lub szufladka, do której możemy wkładać i wyjmować różne dane.

Stwórzmy w takim razie pierwsza zmienną i wyświetlamy jej zawartość na konsole za pomocą poznanej wcześniej funkcji:

```
imie = "Andrzej"
print(imie)
```

Zwracamy uwagę na to, że wartość zmiennej w tym przypadku jest zapisana w cudzysłowie. To znaczy, że zmienna ta jest typu tekstowego (***string***). Zmienne mogą być też innego typu: liczby całkowite (***int***), zmiennoprzecinkowe (***float***), logiczne (***bool***) przykładowe zmienne.

```
kolor = "Czerwony"
wiek = 5
liczbaPi = 3.14
czyPadaDeszcz = False

print(kolor)
print(wiek)
print(liczbaPi)
print(czyPadaDeszcz)
```

Pętle

Pętla for:

Pętla jest to taki element języka, który pozwala powtarzać kilkukrotnie pewne fragmenty skryptów. Wyobraźmy sobie, że ktoś poprosił nas o napisanie skryptu, który wyświetli liczby od 1 do 100. Teoretycznie moglibyśmy napisać sto razy funkcje ***print()***, ale nie jest to raczej wygodny sposób. Możemy do tego wykorzystać wspomnianą pętlę ***for***:

```
for i in range(100):
```

```
    print(i)
```

range() jest to funkcja, która zwraca kolejne liczby z podanego zakresu, domyślnie zaczynając od zera. W każdym wykonaniu pętli do zmiennej **“i”** przypisywana jest ta liczba.
Do funkcji ***range()*** możemy przekazać dwa parametry (oddzielone przecinkiem), wtedy określamy zakres liczb od-do.

```
for i in range(5, 10):

    print(i)

# wynik: 5, 6, 7, 8, 9
```

Podany zakres 5 – 10, wypisze liczby od 5 do 9. Liczba podawana jako górna część zakresu nie jest do niego uwzględniona.
Istnieje też wersja funkcji ***range()*** z trzema parametrami, a wtedy trzeci parametr określa krok (czyli o ile ma być zwiększana wartość zmiennej w każdej iteracji):

```
for i in range(1, 10, 2):

    print(i)

# wynik: 1, 3, 5, 7, 9
```

Pętla while:

W przypadku pętli while nie podajemy zakresu, a jej działanie opiera się na spełnieniu warunku pętli. Dopóki warunek jest spełniony to pętla się wykonuje.

```
x = 1
while x <= 10:
    print(x)
    x = x + 1
```

lub

pętla działająca zawsze

```
x = 1
while True:
    print(x)
    x = x + 1
```

Instrukcja warunkowa

Instrukcje warunkowe - pozwalają wykonywać bądź ominąć wykonanie fragmentów kodu w zależności od tego czy zadany warunek jest spełniony.

Konstrukcja:

if warunek:
** Ten kod wykona się, gdy warunek spełniony**
elif warunek2:
** Ten kod wykona się, gdy warunek2 spełniony**
else:
** Ten kod wykona się, gdy żaden z warunków nie spełniony**

```python
a = 1
b = 4
if a > b:
    print("b jest mniejsze")
elif a < b:
    print("a jest mniejsze ")
else:
    print("a jest równe b")
# wynik: a jest mniejsze
```

Listy

Lista jest to rodzaj zmiennej za pomocą, której możemy przechowywać wiele wartości. Po co nam zatem listy? Wyobraźmy sobie, że chcemy przechować w programie trzy imiona. Jak to zrobimy? W najprostszym przypadku stworzymy trzy zmienne tekstowe:

```python
imie1 = "Jan"

imie2 = "Karol"

imie3 = "Franek"
```

Co w przypadku, jeżeli teraz zmieniło się wymaganie co do naszego skryptu i chcemy mieć możliwość zapisania dziesięciu imion? Stworzymy kolejne zmienne? A co jak będzie potrzeba przechowania tysiąca imion? Zdajemy są sprawę, że nie do końca ma to sens. I właśnie w takiej sytuacji przydają się listy.

W listach elementy są ponumerowane i możemy je dowolnie modyfikować. Deklarujemy jak zmienną z tym, że wartości wymieniamy po przecinku w nawiasach kwadratowych:

```
imiona = ["Jan", "Karol", "Michał"]

print(imiona)
```

Jak dostać się do konkretnego elementu np. jak pobrać z listy imię „Karol”? Elementy w listach mają przypisany odpowiedni indeks. Numerowanie zaczyna się od zero, więc „*Jan*” znajduje się pod indeksem zero, „*Karol*” pod indeksem jeden itd. Pobierzmy element z indeksu jeden, zapiszmy do zmiennej, a następnie wyświetlmy jej wartość:

```
imiona = ["Jan", "Karol", "Michał"]

x = imiona[1]

print(x)
```

Wykorzystując funkcje ***len()*** możemy sprawdzić jak długa jest lista, czyli ile zawiera elementów:

```
imiona = ["Jan", "Karol", "Michał"]

iloscImion = len(imiona)

print(iloscImion) # wynik 3
```

Poznane przez nas wcześniej pętle pozwalają na przeglądanie oraz modyfikowanie w prosty sposób elementów listy. Napiszmy skrypt, którego zadaniem jest przywitać wszystkich znajdujących się w liście „imiona”

```
for i in range(len(imiona)): # pętla wykona się dla każdego elementu li
sty

    print("Witaj " + imiona[i])
```

W języku Python listy mogą przechowywać różne typy danych:

```
lista = ["Hello", 3, True, 3.45]

print(lista)
```

Funkcje

Własne funkcje:
Funkcje to kawałki kodu, które ktoś napisał, tak aby można było ich wielokrotnie używać. Taka podręcznikowa definicja funkcji mówi, że są to „bloki kodu, które uruchamiają się, gdy są wywoływane”.
Funkcje tworzymy używając słowa ***def*** podając nazwę oraz nawiasy okrągłe.

```
import time

def powitanie():

    print("Witaj Adam!")
```

```
    time.sleep(1)

    print("Właśnie stworzyłeś swoją pierwszą funkcję w języku Python")
```

Wywołanie funkcji

```
powitanie()
```

Funkcje z parametrami

Funkcje mogą także przyjmować jakieś dane, funkcja *print()* jest dobrym przykładem, jej zadaniem jest wyświetlenie tekstu na konsoli, ale żeby wiedziała co ma wyświetlić to najpierw musi jej tą informację przekazać. Mówimy wtedy, że funkcja przyjmuje parametry. Parametry wymieniamy w nawiasach okrągłych oddzielając je przecinkami.

```
def powitanie(imie):

    print("Witaj! "+imie)

powitanie("Adam")

powitanie("Darek")
```

Najpopularniejsze funkcje używane podczas zajęć


```
powiedz ":)"
```

```
#wyświetlamy wiadomość na czacie w  grze
nazwaKursu="Python z Minecraftem"
player.say("Witaj Gigancie na kursie "+nazwaKursu)
```

```
teleportuj do to
```

```
#kryjówka na drzewie kordy x=125 y=79 z=50, teleportujemy gracza do wskazanej lokalizacji
player.teleport(world(125, 79, 50))
```

```
uruchom kod przy śmierci gracza
```

giganci programowania

MINECRAFT

pozycja świata gracza

```
#kiedy gracz zginie zostaje teleportowany do pozycji początkowej,
która została zapisana do zmiennej po uruchomieniu programu
aktualnaPozycjaOdrodzenia=player.position()
def smierc():
    player.teleport(aktualnaPozycjaOdrodzenia)
player.on_died(smierc)
```

BLOKI

umieść **block** w **pos**

```
#dodajemy jeden blok TNT w pobliżu gracza
blocks.place(TNT, pos(3, 0, 0))
```

wypełnij **block** od **from** do **to**
operator

```
#bryłę bloków w pobliżu gracza
blocks.fill(43, pos(0, 0, 0), pos(4, 4, 4))
```

giganci programowania

MINECRAFT

uruchom kod gdy **block** umieszczono

```
#zdarzenie na zniszczenie konkretnego bloku
def on_block_broken():
    gameplay.set_game_mode(CREATIVE, mobs.target(NEAREST_PLAYER))
    player.teleport(pos(50, 0, 50))
blocks.on_block_broken(GLOWSTONE, on_block_broken)
```

testuj **block** w **pos**

```
#sprawdzamy blok pod graczem, jeśli pod graczem jest dany blok warunek zwraca True
    if blocks.test_for_block(SEA_LANTERN, pos(0, -1, 0)):
```

wydrukuj **HELLO** **block** w **position** wzdłuż **direction**

```
pozycja=positions.add(player.position(), pos(10, 0, 0))
x=pozycja.get_value(Axis.X)
y=pozycja.get_value(Axis.Y)
z=pozycja.get_value(Axis.Z)
```

giganci programowania

```
#tworzymy napis runda
blocks.print("RUNDA", SEA_LANTERN, world(x+35,y+10,z+45), WEST)
```

MOBY

daj **target** blok lub przedmiot **block**
liczba **1**

```
#gracz otrzymuje broń oraz zbroję
def ekwipunek():

    mobs.give(mobs.target(NEAREST_PLAYER), DIAMOND_PICKAXE, 1)
    mobs.give(mobs.target(NEAREST_PLAYER), DIAMOND_SWORD, 1)
    mobs.give(mobs.target(NEAREST_PLAYER), DIAMOND_CHESTPLATE, 1)
    mobs.give(mobs.target(NEAREST_PLAYER), DIAMOND_HELMET, 1)
    mobs.give(mobs.target(NEAREST_PLAYER), DIAMOND_LEGGINGS, 1)
    mobs.give(mobs.target(NEAREST_PLAYER), DIAMOND_BOOTS, 1)
    mobs.give(mobs.target(NEAREST_PLAYER), BOW, 1)
```


```
    mobs.give(mobs.target(NEAREST_PLAYER), ARROW, 64)
    mobs.give(mobs.target(NEAREST_PLAYER), TRIDENT, 1)

#ta sama funkcja oparta na listach
```

```
def ekwipunek():

    #tworzymy listę według schematu: podajemy co chcemy otrzymać i w ja
kiej liczbie

    przedmioty = [IRON_SWORD,1,IRON_AXE,1,BOW,1,ARROW,64,FIREBALL,10,LA
VA_BUCKET,5,IRON_HELMET,1,IRON_CHESTPLATE,1,IRON_LEGGINGS,1,IRON_BOOTS,
1,SPRUCE_DOOR,20,TORCH,30,LADDER,64,ENCHANTED_APPLE,2]

    #pętla przypisująca dla i tylko przedmioty(id), przeskok co 2

    for i in range(0,len(przedmioty),2):

        #dodanie przedmiotów, dane pobieramy z listy

        mobs.give(mobs.target(NEAREST_PLAYER), przedmioty[i], przedmiot
y[i+1])


player.on_chat("ekwipunek", ekwipunek)
```

giganci programowania

zespawnuj **CHICKEN** w **destination**

```
mobs.spawn(CHICKEN, pos(5, 0, 5))
```

apply **effect** to **target** duration **10** amplifier **1**

```
#gracz otrzymuje efekty typu siła, regeneracja, szybkość
mobs.apply_effect(SPEED, mobs.target(NEAREST_PLAYER),20)
mobs.apply_effect(REGENERATION, mobs.target(NEAREST_PLAYER),20)
mobs.apply_effect(STRENGTH, mobs.target(NEAREST_PLAYER),20
```

zaczaruj **target** za pomocą **infinity** poziomu **1**

```
def zaczarowanyLuk():
    #maksymalne obrażenia
    mobs.enchant(mobs.target(NEAREST_PLAYER), "Power", 5)
    #płonące strzały
```

```
    mobs.enchant(mobs.target(NEAREST_PLAYER), "Flame", 1)

player.on_chat("łuk", zaczarowanyLuk)
```

ROZGRYWKA

zmień tryb gry na **SURVIVAL** dla **player**

```
#przełączamy tryb gry na przetrwanie
gameplay.set_game_mode(SURVIVAL, mobs.target(NEAREST_PLAYER))
```

ustaw czas **DayTime.Day**

```
#przełączamy tryb gry na przetrwanie
gameplay.time_set(DayTime.NIGHT)
```

giganci programowania

MINECRAFT

POZYCJE

~ x ~ y ~ z

```
#tworzymy zombie nad graczem
mobs.spawn(ZOMBIE, pos(0, 5, 0))
```

świat x y z

```
#kryjówka na drzewie kordy x=125 y=79 z=50
player.teleport(world(125, 79, 50))
```

p1 + p2

```
#do aktualnej pozycji gracza dodajemy przesunięcie o 10 bloków na osi
x
pozycja=positions.add(player.position(), pos(10, 0, 0))
```

wybierz pozycję losową od p1 do p2

```
mobs.spawn(ZOMBIE, randpos(world(x+5,y+5,z+5), world(x+35,y+5,z+35)))
```

giganci programowania

MINECRAFT

PĘTLE

while

```
while True:
    #sprawdzamy blok pod graczem
    if blocks.test_for_block(SEA_LANTERN, pos(0, -1, 0)):
        #pobieramy aktualną pozycję z i przekazujemy do funkcji
        pozycja=player.position()
        z=pozycja.get_value(Axis.Z)
        etap1(z)
        loops.pause(3000)
```

for

```
for i in range(6):
    mobs.spawn(CREEPER, randpos(world(x-8,y+1,z+3), world(x+8,y+1,z+20)))
```

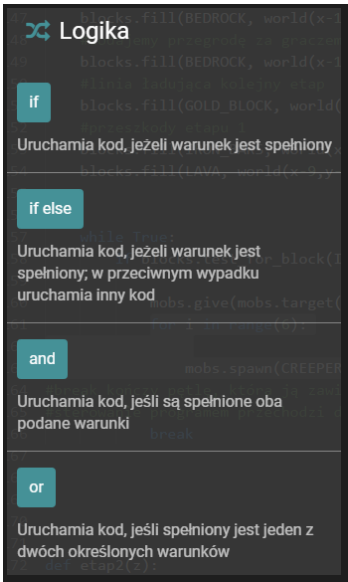
pauza (ms) 100

```
#1 sekunda to 1000 milisekund(ms)
loops.pause(1000)
```

giganci programowania

MINECRAFT

LOGIKA




```
poziom=int(input("Jaki potrzebujesz poziom światła?"))

if poziom == 15:
    print("Możesz wykorzystać blok jasnogłazu")


elif poziom < 15 and poziom >=10:
    print("Wykorzystaj pochodnię albo eteryczna latarenkę")
else:
```

```
    print("Prawdopodobnie podałeś nieprawidłowe dane zakres światła to 1
-15")
```

ZMIENNE

```
#tworzymy zmienną deski o przypisujemy jej wartość którą poda gracz
deski=int(input("ile masz desek?"))
#zwiększamy wartość zmiennej o 1
aktualnaLiczbaTorow += 1
```

KSZTAŁTY

rząd block od p0 do p1

```
#tworzymy schody
for i in range(15,21):
    shapes.line(STONE_BRICK_STAIRS, world(x-4,y,z+i), world(x,y+4,z+i))
```

kula block środek center promień 5 operator


```
#tworzymy kulę ze szlamu
shapes.sphere(SLIME_BLOCK, pos(4, -30, 0), 1, ShapeOperation.REPLACE)
```