Pojęcia do programowania w python.

**Ważne pojęcia**

**Zmienne** - pozwalają zapisać wartości do siebie, można je wykorzystać/zmienić za pomocą etykiet (ich nazw). W przenośni są to pojemniki do przechowywania wartości.

Do stworzenia zmiennej potrzebujemy etykietę (nazwę), znak równości oraz wartość jaką chcemy jej przypisać np.

wiek = 28

aby wydrukować wartość zmiennej piszemy:

|  |
| --- |
| print(wiek) |

**Shell** - powłoka interaktywna mająca za zadanie interakcje między Tobą, a interpreterem, który analizuje i wykonuje instrukcje. Pozwala na interakcję ze skryptem (kodem) napisanym i wywołanym z zewnątrz np. przez Ciebie.

Zmiana stanu powłoki zachodzi w momencie, gdy uruchomisz skrypt lub gdy np. stworzysz zmienną z pomocą interpretera. Zmiana ta istnieje dopóki nie zrestartujemy powłoki. Tzn. “stan” powłoki wraz z jej zawartością jest cały czas niezmienny. Co oznacza, że raz stworzone zmienne istnieją dopóki nie zamkniemy całego IDLE lub nie zrestartujemy powłoki np. uruchamiając ponownie nowy skrypt.

**Skrypt** - kod który zapisujemy, można go wywołać. Komputer wywołuje skrypt z komendami (instrukcjami) od góry do dołu.

Zapis zmiennych w Pythonie:

1. **liczb całkowitych**

a = 6 #l. całkowita z angielskiego int (integer)

b)  **liczby z ułamkiem** zapisujemy z **kropką**

b = 5.3 #l. zmiennoprzecinkowa (float)

c)  **ciąg znaków tzw. string** np. imię, nazwisko, tekst zapisujemy za pomocą podwójnego lub pojedynczego cudzysłowia (apostrofu)

imie = "Arkadiusz" lub imie = 'Arkadiusz'

Należy pamiętać, że przy pisaniu zdania np. w języku angielskim używając apostrofa np.

|  |
| --- |
| zdanie = 'I'm from Poland' |

musisz zastosować znak” \” (slash):

|  |
| --- |
| zdanie = ('I\'m from Poland'), |

który mówi komputerowi, że kolejny znak jest w tym wypadku apostrofem jako znak, a nie jako zakończenie stringa.

**Sprawdzenie typu zmiennych - za pomocą słowa type**

**np. type(a)**

|  |
| --- |
| type (zdanie)   type (imie) |

**Nazewnictwo zmiennych jest bardzo ważne :**

* Zmienne powinny być w programowaniu samoopisujące się np.

imie = "Arkadiusz", nazwisko = "Kowalski" , wiek = 34

przez samoopisujące się rozumiemy, że czytając nazwę zmiennej wiemy co ona przechowuje

* Nie korzystamy z polskich znaków, przy pisaniu kodów
* Nie zaczynamy nazewnictwa zmiennej od liczb
* Dla czytelności warto powiększać pierwsze litery kolejnych słów w nazwie zmiennej np.: nazwaUzytkownika

String, czyli ciąg znaków **zawsze musi** być pomiędzy cudzysłowami, aby komputer zinterpretował ciąg znaków jako String.

tzn.

"Arkadiusz" - poprawnie

Arkadiusz - niepoprawnie

Aby połączyć ze sobą dwie zmienne typu **string**, aby był pomiędzy nimi odstęp:

|  |
| --- |
| imie = "Arkadiusz" nazwisko = "Włodarczyk" |

należy zrobić sztuczną spacje poprzez użycie “ “.

|  |
| --- |
| calosc = imie + " " + nazwisko |

Znak **+** służy do łączenia zmiennych typu string.

Aby wybrać znak ze zmiennej typu string pamiętamy o tym, że numerujemy znaki poczynając od **zera**.

# N A T A L I A

**0**  1 2 3 4 5 6 - INDEKS - pierwszy element zawsze 0

-7 -6 -5 -4 -3 -2 **-1**  ostatni element zawsze -1

np. print (imie[:-1]) bez ostatniego elementu

print (imie[1:3]) od indeksu nr 1 do 2 (element o indeksie 3 nie jest wliczany)

Funkcja - To kod, zapisany i stworzony przez innego programistę, który można wywołać. Gdy po nazwie funkcji dodamy nawiasy, następuje jej wywołanie. Przez wywołanie rozumiemy odpalenie (wykonanie) kodu, czyli instrukcji napisanych przez innego programistę.

print (“lalalala”)

print - nazwa funkcji

( ) - wywołanie funkcji

“lalalala” - argument funkcji

**Operatory Arytmetyczne**

Operatory arytmetyczne operują na liczbach, dzięki nim możemy

* dodawać
* odejmować

\* mnożyć

/ dzielić (podczas dzielenia liczby całkowite zamieniają się w liczbę zmiennoprzecinkową (float)

() jak w matematyce kierują kolejnością wykonywania działań

\*\* potęgowanie np. 2\*\*3 = 8

/ dzielenie np. 9/4 = 2.25

// dzielenie w “dół” - z ang. floor np. 9//4 = 2

% modulo (reszta z dzielenia) np. 9%3 = 0

**Operatory logiczne**

1. **and** - spójnik “ i ”

**KONIUNKCJA** - jest prawdziwa wtedy i tylko wtedy, gdy oba wyrażenia są prawdziwe np.

|  |
| --- |
| wartosc = 0  if(wartosc > 1 and wartosc < 10): |

Jeżeli nasza przykładowa wartość jest większa od 1 **i** mniejsza od 10 w tym samym momencie nasza koniunkcja jest prawdziwa.

2. **or** - “lub”

**ALTERNATYWA** - jest fałszywa, wtedy i tylko wtedy gdy oba wyrażenia są w tym samym momencie są fałszywe. np

|  |
| --- |
| a = 5 b = 2  if(a == 5 or b == 3):  print("tak") |

W tym przykładzie a jest równe 5. Wyrażenie jest prawdziwe **(true)** ponieważ mamy tutaj operator logiczny “**or**” , czyli wystarczy że jedno z wyrażeń jest prawdziwe.

3. z ang **not** - nie (zaprzeczenie)

np. Jeżeli wartość **nie** jest w zakresie 1-10

|  |
| --- |
| if (not(wartosc >= 1 and wartosc <= 10)):  print("wartosc nie jest od 1 do 10") |

**Operatory porównania - to operatory, które porównują wartości po lewej i prawej stronie**

**5 > 4**

**jeżeli równanie jest prawdziwe tak jak w powyższym przykładzie (5 jest większe od 4) to operator zwraca nam na wynik True, czyli prawdę.**

**3 > 3 # zwraca False, ponieważ 3 jest równe 3 - nie jest większe**

**PAMIĘTAJ, ŻE**

**- aby sprawdzić czy dwie wartości są ze sobą równe używamy tylko i wyłącznie znaku == np. 2 == 2 (czy 2 jest równe 2?)**

**Jeżeli użyjesz do porównania dwóch wartości pojedynczego znaku równości wystąpi błąd, ponieważ = odpowiada za przypisanie wartości.**

**Operatory porównania:**

**> większy niż**

**< mniejszy niż**

**== równy**

**!= nierówny**

**>= większy bądź równy**

**<= mniejszy bądź równy**

**OPERATORY PRZYPISANIA**

Stwórz zmienną x i przypisz jej wartość 4

x = 4

Jeżeli chcesz:

- powiększyć x o jakąkolwiek inną wartość np. 3 stwórz kod:

x = x + wartość którą chcesz dodać do x np.

x = x + 3

Interpreter czyta to co dzieje się pierw po prawej stronie, czyli szuka co znajduje się

pod zmienną x, następnie dodaje 4 + 3 i otrzymujesz wynik x równa się 7

Dużo szybszym i czytelniejszym zapisem jest

x += 3 #dodaj do x liczbę 3

Wynik jest ten sam x równa się 7

- pomnożyć zmienną \* 2

x = 4

x \*= 2 #x\*2

Wynik to 4 \* 2 = 8

W taki sam sposób możesz korzystać z innych operatorów

+= x += 5 to skrócony zapis: x = x + 5

-= x -= 5 to skrócony zapis: x = x - 5

\*= x \*= 5 to skrócony zapis: x = x \* 5

/= x /= 5 to skrócony zapis: x = x / 5

%= x %= 5 to skrócony zapis: x = x % 5

//= x //= 5 to skrócony zapis: x = x // 5

\*\*= x \*\*= 5 to skrócony zapis: x = x \*\* 5

**input ( ) z ang. wejście**

**( ) - wywoływanie funkcji**

**Gdy funkcja wraca do miejsca wywołania - zwraca wynik, który został w tej funkcji obliczony.**

np.

a = input() # użytkownik podał '5**'**

b = input() # użytkownik podał '10'

a+b = '510' (zmienne interpretowane są jako ciąg znaków, które łączą się ze sobą za pomocą plusa)

Jeżeli chcemy aby zmienne były uważane za liczby trzeba wykonać **rzutowanie**

**Rzutowanie - to zmiana typu z jednego na drugi np.**

int(a) - rzutowanie do zmiennej całkowitej

float(a) - rzutowanie do liczby zmiennoprzecinkowej

str(a) - rzutowanie do stringa (ciągu znaków)

Przykładowe użycie:

|  |
| --- |
| a = int(input()) b = int(input())  print = (a+b) a + b = 15 |

**Instrukcja warunkowa** - to instrukcja, która warunkuje, sprawdza czy coś jest prawdą czy fałszem.

1. z ang . **if** - jeśli

**if (5 > 3):          przy użyciu “if”  po nawiasach należy użyć  dwukropek**

**print(“lalala”)** zaznaczona niebieska linia to **WCIĘCIE,** które oznacza że instrukcja    która jest pod “if” ma dotyczyć warunku. **Wcięcia muszą być równe**, aby je stworzyć w Pythonie wystarczy klawisz “tab” .

     b) z ang. **elif** - od ang. else if - **jeśli jeszcze co innego**

|  |
| --- |
| elif(3 < 5):            print("3 jest mniejsze od 5") |

     c) z ang. **else** - całkowicie inny

|  |
| --- |
| else:            print("całkowicie inny wynik") |

Jeśli porównujemy liczby należy pamiętać o rzutowaniu np.

|  |
| --- |
| a = int(input()) b = int(input()) |

**Pętla (z ang. while to “podczas”, “gdy”) służy do zapętlania, czyli powtarzania danych instrukcji wielokrotnie. Pętla będzie się wykonywać, dopóki warunek będzie prawdziwy np.**

liczba = 0

while liczba < 5:

print(liczba)

ponieważ liczba się nie zmienia i cały czas jest równa 0 to ta pętla jest tzw. pętlą nieskończoną. Dzieje się tak ponieważ warunek będzie zawsze prawdziwy, 0 zawsze będzie mniejsze od 5.

Aby przerwać pętle nieskończoną naciskamy Ctrl + C ( przerwanie pętli)

Aby upewnić się, że pętla się zakończy należy zmienić zmienić warunek pętli w taki sposób, aby warunek kiedyś stał się fałszywy. Zmianę dokonujemy wewnątrz pętli np.

liczba = 0

while liczba < 5:

print(liczba)

liczba += 1

**Pętla for** będzie się wykonywać dopóki ma w zasięgu elementy, przez które może przejść.

Przykładowa pętla for

|  |
| --- |
| **i = 0   for i in range(0,4):   x = int(input("Podaj liczbe": ))  wynik += x** |

z ang.  **for**- dla

**range** - zakres

**in** - w

**Range** można wywołać na dwa sposoby:

1. z jednym argumentem
2. z dwoma argumentami
3. Gdy wyślemy jeden argument to range pozwoli nam wygenerować liczby od 0 do podanej wartości **minus** 1 np.

range(4) - wartości od 0 do **3**

1. Gdy wyślemy dwa argumenty to range pozwoli nam wygenerować liczby od pierwszego argumentu (włącznie) do podanej wartości **minus** 1 np.:

range(1,4) - wartości od 1 do 3

Ilość możliwych do wygenerowania elementów w nawiasach, określa ile razy wykona się pętla:

|  |
| --- |
| for i in range (4):  print("ta instrukcja wykona się 4 razy") |

z ang. **break** - przerywać, to instrukcja która służy przerwaniu wykonywania pętli

z ang. **continue** - kontynuować, to instrukcja która przerywa tylko aktualne przejście pętli. Pętla kontynuuje swoją pracę od następnego “przejścia”.

**Lista - pojemnik do przechowywania elementów, która jest numerowana od 0**

Przykład listy :

**imiona** = **[** **“Arkadiusz”, “Wioletta”, “Karol”, “Kuba”, “Adrian” ]**

**0 1 2 3 4**

**imiona - nazwa listy**

**[ ] - klamra kwadratowa, która oznacza że tworzymy listę**

**“Arkadiusz”, “Wioletta”, “Karol”, “Kuba”, “Adrian” - elementy listy**

**0,1,2,3,4 - indeksy (numery) konkretnej pozycji elementu w liście**

W liście możemy mieszać wartości. Do elementów listy możemy się szybko odwołać np.

a) aby wypisać wszystkie elementy z listy:

|  |
| --- |
| print(imiona) |

b) aby wybrać dany element z listy

print(nazwa\_listy[indeks\_elementu\_do\_którego\_chcemy\_się\_odwołać])

np.ostatni element:

|  |
| --- |
| print(imiona[-1]) |
|  |
|  |

**IN, NOT IN**

Za pomocą wyrażenia **“in”** (z ang. “w”) możemy sprawdzić, czy dany element znajduje się w liście np.

|  |
| --- |
|  |