WIZUALIZACJA DANYCH

Ćwiczenie 2

Python 3: Zapisywanie i wykonanie skryptów. Instrukcje sterujące i obsługa błędów

Wprowadzanie danych

Do wprowadzania danych możemy użyć komendy input.

Przykład

```
a = input("Tu jest jakiś komunikat np. Podaj liczbę\n")
print(a)
```

Możemy użyć też komend readline() i write(s), które są w module sys

Przykład

```
import sys

print("Podaj jakiś tekst")
s = sys.stdin.readline() #Wczytuje wiersz
print("Twój tekst to: " + s)

#Do wydruku można użyć też komendy write np.
sys.stdout.write(s)
```

Zad. 1

Napisz skrypt, który pobiera od użytkownika zdanie i liczy w nim spacje. Wynik wyświetla na ekranie (użyj instrukcji input)

Zad. 2

Napisz skrypt, który pobiera od użytkownika dwie wartości i mnoży je przez siebie. Wynik wyświetla na ekranie (użyj instrukcji readline() i write()).

Instrukcja warunkowa

Składnia:

```
if warunek_1:
      Instrukcje_1
[elif warunek_2:
      Instrukcje_2:
…
elif warunek_n:
      Instrukcje_n
else:
      Inne_instrukcje]
```

***Przykład*:**

```
#Pobieramy od użytkownika liczbę
#Chcemy sprawdzić czy jest dodatnia czy ujemna

liczba = input("Podaj jakąś liczbę")

#liczba jest typu string musimy ją rzutować na całkowitą

liczba = int(liczba)

if liczba > 0:
    print("Podano liczbę dodatnią")
elif liczba < 0:
    print("Podano liczbę ujemną")
else:
    print("Podano liczbę równą zero")
```

Przykład

```
#Sprawdzamy, która liczba jest większa

liczba = input("Podaj pierwszą liczbę")
k = input("Podaj drugą liczbę")

#liczba, k są typu string musimy ją rzutować na całkowitą

liczba = int(liczba)
k = int(k)

#przy wyświetlaniu zmieniamy liczbę na typ string

if liczba > k:
    print("Liczba=" + str(liczba) + " jest większa")
else:
    print("Liczba=" + str(k) + " jest większa")
```

Zad. 3

Odszukaj w dokumentacji, jakie operatory można używać w instrukcjach warunkowych (np. równe, różne, mniejsze bądź równe itp.)

Zad. 4

Napisz skrypt, który pobiera od użytkownika liczbę i wypisuje na ekran wartość bezwzględną tej liczby

Zad. 5

Napisz skrypt, który pobiera od użytkownika trzy liczby a, b i c. Sprawdza następujące warunki:

czy a zawiera się w przedziale (0,10> oraz czy jednocześnie a>b lub b>c. Jeśli warunki są spełnione lub nie to ma się wyświetlić odpowiedni komunikat na ekranie.

Instrukcja iteracyjna for

Składnia:

```
for licznik in sekwencja:
     Instrukcje
[else:
     inne_instrukcje]
```

Sekwencją może być łańcuch, lista lub krotka. Od obliczenia sekwencji zaczyna się działanie instrukcji iteracyjnej. Licznik przyjmuje wartość pierwszego elementu wykonuje instrukcje, następnie przyjmuje wartość kolejnego elementu itd.

Do utworzenia sekwencji możemy użyć funkcji range:

```
range([start], stop[, krok])
```

Przykład

```
#Chcemy wyświetlić liczby od 1 do 5

for x in range(1, 6, 1):
    print(str(x) + " ")


#Tworzymy swoją listę i chcemy jej użyć jako sekwencji do
wyświetlania wartości

lista = [3, 4, 2, 1, 6]

for x in lista:
    print(str(x) + " ")


#Wyświetlamy elementy za pomocą pętli ale na koniec odpowiedni
komunikat

lista = [3, 4, 2, 1, 6]

for x in lista:
    print(str(x) + " ")
else:
    print("Koniec wyświetlania")
```

Zad. 6

Napisz pętlę, która wyświetla liczby podzielne przez 5

Zad. 7

Napisz pętle, która pobiera liczby od użytkownika i wyświetla ich kwadraty na ekranie.

Instrukcja iteracyjna while

Składnia:

```
while warunek:
      instrukcje
[else:
      inne_instrukcje]
```

Przykład:

```
#skrypt wyświetla losowe liczby całkowite aż napotka 5

import random #biblioteka z funkcjami do losowania

random.seed() #inicjowanie generatora

z = random.randint(1, 15) #losowanie pierwszej liczby

while(z != 5):
    print(str(z))
    z = random.randint(1,15)
else:
    print("Znalazłem 5 koniec działania")
```

Zad. 8

Napisz skrypt, który odczytuje liczby od użytkownika i umieszcza je na liście. Wykorzystaj pętle while.

Zad. 9

Napisz skrypt, który odczytuje od użytkownika liczbę wielocyfrową i sumuje jej cyfry. Wynik wyświetla na ekranie. Wykorzystaj pętle while.

Instrukcje break i continue

Instrukcje umieszczamy w pętlach. Sterują działaniem pętli.

Break – przerywa działanie pętli w której się znajduje (ale nie wszystkich pętli jeśli pętle zagnieżdżamy)

Continue – kończy przebieg aktualnej iteracji pętli i przechodzi do następnego przebiegu.

Przykład

```
#Użytkownik podaje liczbę
#Przeglądamy listę liczb i jeśli znajdziemy tę podaną przez użytkownika
#wyświetlamy komunikat i kończymy działanie pętli

lista = [1, 5, 3, 2, 6, 7, 8, 9, 10]
print("Podaj liczbę a sprawdzę czy jest na liście")
```

```
liczba = input()
licznik = 0
while licznik < 10:
    #Jeśli znajdziemy liczbę przerywamy
    if int(liczba) == lista[licznik]:
        print("Twoja liczba: " + liczba + "znaleziona na pozycji: " +
str(licznik))
        break
    else:
       licznik += 1
```

Instrukcje iteracyjne zagnieżdżone

Pętle, instrukcje warunkowe możemy umieszczać jedna w drugiej. Nazywamy to zagnieżdżaniem.

Przykład

Chcemy wyświetlić wzór na ekranie:
H H
H H
HHHHH
H H
H H
kod:

```
import sys

for i in [1, 2, 3, 4, 5]:
    for j in [1, 2, 3, 4, 5]:
        #sprawdza czy jest w pionowej linii H
        if i == 3:
            sys.stdout.write('H')
        #jeśli nie to rysuje H na brzegach
        else:
            #sprawdzamy czy j zawiera się na liście, czyli jest na brzegach
            if j in [1, 5]:
                sys.stdout.write('H')
            #jeśli nie piszemy spację
            else:
                sys.stdout.write(' ')
    sys.stdout.write('\n')
```

Zad. 10

Napisz skrypt, który rysuje wieżę z literek. Użytkownik podaje wysokość wieży ale nie więcej jak 10.

A
AA
AAA
AAAA
AAAAA
AAAAAA

Zad. 11

Napisz skrypt, który rysuje diament. Użytkownik podaje wysokość nie mniej jak 3 i nie więcej jak 9
wysokość=3

```
  o
ooo
  o
```
wysokość równa 5
```
   o
  ooo
 ooooo
  ooo
   o
```

itd.

Zad. 12

Napisz skrypt, który wyświetla i oblicza tabliczkę mnożenia od 1 do 100

Obsługa błędów

Mamy 3 rodzaje błędów:

1. Błędy składniowe – powstają gdy piszemy program niezgodnie z gramatyką języka np. błędy w definicjach funkcji, niezamknięte nawiasy czy cudzysłów bez pary.
2. Błędy czasu wykonania – powodują przerwanie lub niewłaściwe działanie. Mamy możliwość ich przechwycenia i wymuszenia odpowiedniej reakcji np. użytkownik podaje litery a miał wpisywać liczby.
3. Błędy logiczne – błędy w algorytmie lub programie. Nie wykrywalne przez interpreter ale możliwe przez człowieka po analizie programu

Obsługę błędów realizujemy przez specjalny blok instrukcji.

***Składnia*:**

```
try:
      instrukcje
except nazwa_bledu_1:
      awaryjne_instrukcje_1
…
[except nazwa_bledu_n:
      awaryjne_instrukcje_n]
[else:
      blok_bez_błędu]
```

***Prostsza składnia*:**

```
try:
      instrukcje
finally:
      blok_zamykający
```

Przykład

```
#Uzytkownik podaje liczby do dzielenia
#Chcemy wyłapać moment dzielenia przez zero
print("Proszę podać pierwszą liczbę")
licz1 = input()
print("Proszę podać drugą liczbę")
licz2 = input()
try:
    wynik = int(licz1) / int(licz2)
    print("Wynik= " + str(wynik))
      except ZeroDivisionError: #nazwa błędu dzielenia przez zero
    print("Tylko Chuck Norris może dzielić przez zero!")
```

Zad. 13

Przestudiuj inne błędy jakie mogą się pojawić w trakcie działania programu:
https://pl.python.org/docs/tut/node10.html

Zad. 14

Napisz skrypt, który liczy pierwiastek z liczby podanej przez użytkownika jeśli użytkownik poda wartość ujemną to powinien być wyłapany błąd

Zad. 15*

Napisz skrypt, w którym użytkownik ma podać liczbę i który będzie wyłapywał błąd gdy użytkownik poda literę zamiast cyfry.

Bibliografia

[1] Jacqueline Kazil, [online], Katharine Jarmul, Data Wrangling with Python, wyd. 1, O'Reilly, 2016, dostęp: 13 września 2017, Dostępny w Internecie: http://pdf.th7.cn/down/files/1603/Data%20Wrangling%20with%20Python.pdf

[2] Wes McKinney, [online], Python for Data Analysis, wyd. 1, O'Reilly, 2013, dostęp: 13 września 2017, Był dostępny w Internecie: http://www3.canisius.edu/~yany/python/Python4DataAnalysis.pdf

[3] Marian Mysior, *Ćwiczenia z języka Python*, Warszawa, Mikom, 2003