Python

liczba – definicja
zmienna – zasady nazewnictwa i definiowania
podstawowe typy danych
separatory
funkcje – budowa, przykłady
funkcje konwertujące
definicja funkcji własnej
działania i ich oznaczenia
operatory porównania, logiczne
wartości logiczne
instrukcja warunkowa if
****

Pliki stworzone w Pythonie mają rozszerzenie py (*.py). Plik stworzony w Pythonie nazywamy Skryptem.
Plik zapisany w języku programowania zawiera kod źródłowy.

Liczba-ciąg cyfr rozdzielony maksymalnie jednym separatorem liczb dziesiętnych (kropla (lub przecinek w programach użytkowych np. Excel)), na początku którego może stać znak minus(-). Liczba posiada wartość.

definicja zmiennej: nazwa_zmiennej = *dana (wartość)*
dana może być podana wprost np. „Ala”, 13, 14.25, działanie lub z zewnątrz za pomocą funkcji input().

zasady nazewnictwa zmiennej (oraz funkcji własnej):
- TYLKO podstawowy alfabet łaciński (litery bez znaków diakretycznych), cyfry i podkreślnik (podłoga);
- NIE WOLNO UŻYWAĆ : spacji, znaków specjalnych, znaków interpunkcyjnych;
- ważna jest wielkość liter: ala ≠ Ala;
- długość do 24 znaków (im krótsza tym lepsza);

komentarz – program nie wykonuje wpisanego kodu:
tekst - jednowierszowy
”””tekst””” - wielowierszowy

Dane (podstawowe)**:**
typy danych:
liczba:
- l. całkowite (intiger - int)
- l. zmiennoprzecinkowe, l. rzeczywiste z rozwinięciem dziesiętnym (floating - float)-
  SPERATOR LICZB DZIESIĘTNYCH – **kropka (.)** np.: 1.5

tekst:
- łańcuch znaków (string - str)

logiczne:
- wartości logiczne: **True** – prawda (1); **False** – fałsz (0) (boolean - bool)

Daną typu tekstowego (string) UMIESZCZAMY W CUDZYSŁOWIE lub apostrofach np.: ”tekst” ,’tekst’

Funkcje:
budowa funkcji: nazwa_funkcji(argumenty,parametry) – nazwy (funkcji wbudowanych) pisane małą literą
funkcja wbudowana – funkcja pierwotnie umieszczona w języku programowania

funkcja print()
funkcja **print()**– drukowanie na ekranie (wyświetlanie)
argumenty w funkcji rozdzielane są przez przecinek (,)
print(),print("Dzien dobry :-)",3,3.25, 4+12.5)
parametry funkcji print()
print (end=”tekst po wykonaniu instrukcji”) – domyślnym znakiem wstawianym po wykonaniu funkcji jest znak końca akapitu - Enter
print (sep=”*tekst* rozdzielający argumenty po wykonaniu”) – domyślnym znakiem po wykonaniu funkcji jest spacja
”” – puste (dwa cudzysłowa obok siebie”
”\n” – w funkcji print() powoduje przejście do nowego wiersza(linii)

funkcja input()
funkcja **input()** - pobiera dane z zewnątrz (dane podaje użytkownik), jest to zawsze dana typu **string**.
input("opis do wpisania odpowiedniego ciągu znaków")

funkcje do zmiany typów danych (konwersja):
int()- l.całkowite (intiger)
float()-l. zmiennoprzecinkowe, l.rzeczywiste z rozwinięciem dziesiętnym (floating)
str() - tekst: łańcuch znaków (string)
bool() - logiczne

np.: licz_1=int(input(”wpisz liczbę całkowitą”)) – efekt: wprowadzona odpowiednio wpisana dana jest zapisana w postaci liczby całkowitej.
WPROWADZANA DANA MUSI BYĆ ODPOWIEDNIO WPISANA, żeby być przypisana odpowiedniemu typowi danych.

Python zachowuje kolejność działań: nawiasy, potęgowanie, mnożenie, dzielenie, dodawanie i odejmowanie

Działania:
nawiasy: ()
dodawanie: +
odejmowanie: -
mnożenie:*
dzielenie: /
potęgowanie:**
dzielenie całkowite: //
modulo (reszta z dzielenia): %

Działania mogą być dokonywane tylko na danych tego samego typu (odpowiednie działania).

Ważna jest struktura zapisu – wcięcie (zmiana poziomu) uzależnia wykonanie instrukcji od instrukcji zapisanej poziom wyżej (bliżej lewej strony), na końcu której stoi : (dwukropek) .

W interpretatorze języka Python (np. Thony) jeżeli chcemy przesunąć cały blok instrukcji o poziom w głąb można zamarkować (podświetlić) cały ten blok, a następnie wcisnąć znak tabulacji (Tab).

Definiowanie funkcji (na górze skryptu) przed kodem właściwym
Zasady nazewnictwa f-cji są takie same jak zmiennej

```
def nazwa_funkcji_wlasnej(argumenty):
    blok instrukcji
    return efekt działania funkcji #return z ang. zwrócić
```

np.
```
def reszta_z_dziel(a,b):
    return a%b
```

Do zapisu warunków:

Operatory porównania*:*
> większy
>= większy równy
< mniejszy
<= mniejszy równy
!= różny
== równy

Operatory *logiczne*:
and – logiczne i (część wspólna – wszystkie warunki spełnione jednocześnie)
or – logiczne lub (suma zbiorów – przynajmniej jeden warunek spełniony)
not – logiczne negacja (zaprzeczenie)

Wartości logiczne:
True – prawda (1)
False – fałsz (0)

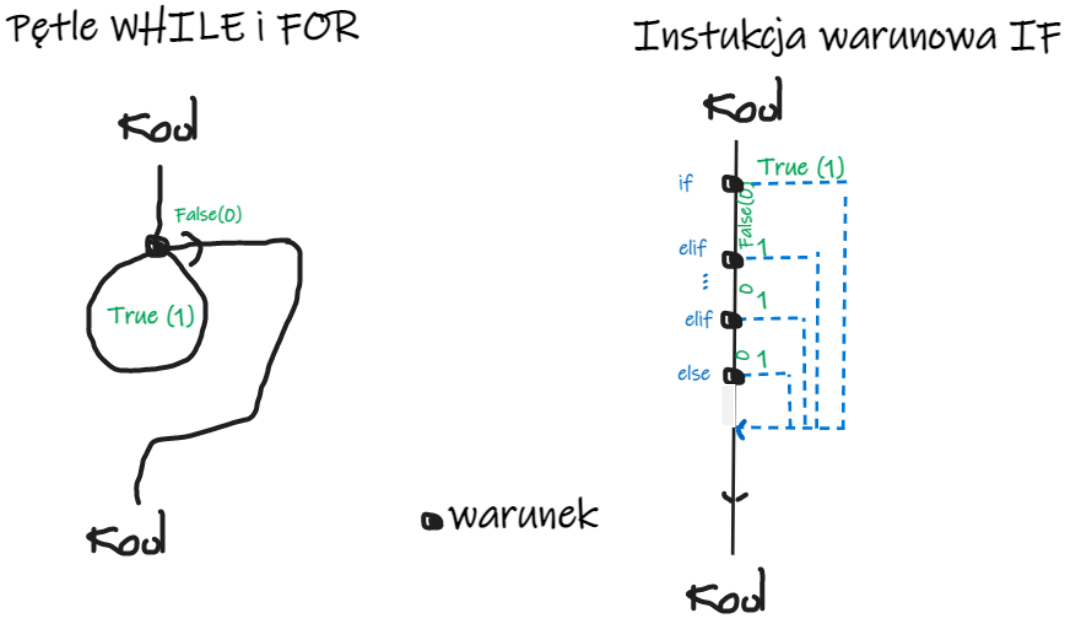
Instrukcja warunkowa **if** – (z ang. jeśli) warunek musi mieć wartość **True** aby była wykonana instrukcja. Instrukcja posiada podinstrukcje, które mogą, ale nie muszą wystąpić elif – kolejny warunek, else (z ang. w przeciwnym razie) – wykonuje się gdy powyższe warunki przy if i elif nie zostały spełnione.

```
if warunek:
    blok instrukcji
```
---
warunek i wszystko inne (przeciwieństwo):

```
if warunek:
    blok instrukcji
else:
    blok instrukcji
```
---
wiele warunków (warunki nie mogą się pokrywać czyli mieć części wspólnej) – pełna istrukcja

```
if warunek:
    blok instrukcji
elif warunek:
    blok instrukcji
.
.
.
else:
    blok instrukcji
```

pętla while
pętla for



****

pętla **while –** (z ang.podczas gdy) działa dopóki warunek jest spełniony (ma wartości True)

- z góry nie jest określona ilość wykonania pętli
- należy pamiętać, aby wewnątrz jej była zmiana wartości zmiennej, na której oparty jest warunek, do którego jest ona zależna
- pod **zmienną** pamiętana jest ostatnia wartość jej nadana
- polecenia przerywające działanie pętli: **break** (przerywa i wychodzi z pętli) oraz **continue** (przerywa i wraca na początek pętli)

while *warunek*:
    blok instrukcji

Przykład pętli while zawsze otwartej:

#skrypt, który wyświetla liczby parzyste od podanej wartości do zera:

```
print("Wydrukowanie wszystkich parzystych liczb z przedziału od 0 do x")
x=int(input("Podaj koniec zakresu - liczba naturalna:"))
if x>=0:
   if x%2==0:
      print(x)
   if x>0:
      while True:
         x=x-1
         if x==0:
            print(x)
            break # przerywa i wychodzi z pętli
         elif x%2==0:
            print(x)
         else:
            continue # przerywa i wraca na początek pętli
         print("Szukamy kolejnej liczby parzystej")
else:
   print("Nie podałeś liczy naturalnej")
print("koniec programu")
```

****

Pętla **for** – (z ang. dla) działa dopóki zmienna wewnętrzna ma wartości z sekwencji **–** pętla for ma określoną z góry liczbę wykonań

```
for zmienna_wew in sekwencja:
   blok instrukcji
```

string

```
for x in "Grażyna":
   print(x,end="")
```

lista

```
for x in [5,"ala",67]:
   print(x,end=" ")
```

range() – funkcja generuje ciąg liczb naturalnych (indeksy) od 0 - zakres.

```
for x in range(6):
   print(x,end=" ")
```

efekt: 0 1 2 3 4 5 (liczby od 0 z 5 - 6 indeksów)

```
for x in range(1,6):
   print(x,end=" ")
```

efekt: 1 2 3 4 5 (liczby od wartości 1 do 5 z 6 indeksów)

```
for x in range(1,25,3):
   print(x,end=" ")
```

efekt: 1 4 7 10 13 16 19 21 24 (liczby od wartości 1 do 24 co 3 krok (wartość) z 25 indeksów)

Część II

"\n" – w funkcji print() powoduje przejście do nowego wiersza(linii)

SEKWENCYJNE typy danych:
string (str)– łańcuch znaków
typy złożone:
tuple - krotki - (el1,el2,…,eln) – funkcja konwertująca: tuple()
list - listy – [el1,el2,…,eln] - funkcja konwertująca: list()
dictionary (dict) **-** słowniki - {el1,el2,…,eln} - funkcja konwertująca: dict()

PYTHON JEST OBIEKTOWYM JĘZYKIEM PROGRAMOWANIA
class (klsa) – np. list (listy)
lista1=[…] - obiekt klasy list, **tekst**="Ala ma kota." - obiekt klasy string, **liczba**= 3.6 - obiekt klasy floating
lista1.append("Ala") -> obiekt.metoda() – metoda jest przypisana do danej klasy, może się zdarzyć, że jedna metoda występuje w kilku klasach.

tuple(krotka) – „krotka jest to lista, której nie można modyfikować"

list - listy – [el1,el2,…,eln]

-7	-6	-5	-4	-3	-2	-1	index -
G	R	A	Ż	Y	N	A	string
el1	el2	el3	el4	el5	el6	el7	tuple, list, dictonary
0	1	2	3	4	5	6	index +

index – nr porządkowy elementu w danej sekwencyjnej – zaczynamy od 0.
"" – pusty string
[]-pusta lista

Listy i krotki (list and tuple) oraz innych danych strumieniowych (stringi i dictonary)
wywoływanie po indeksie: obiekt[index]
wywoływanie przedziału: obiekt[x:y] *(x<y) x- index początek przedziału, y –koniec przedziału o inideksie y-1*
wywoływanie przedziału od początku do danego indeksu obiekt[:y] *y –koniec przedziału o inideksie y-1*
wywoływanie przedziału od danego indeksu do końca: obiekt[x :] *x- index początek przedziału*

funkcje:
len()-zlicza ilość elementów – funkcja ogólna
reversed() – odwrotna kolejność w pętli for

plecenia:
del – usuwanie (ogólnym)

Listy i krotki (list and tuple)

metody dla klasy (class) **list** (el – element) - na tuple (krotki) działają te same metody co na listach, pod warunkiem, że ich **nie modyfikują.**
uwaga – jeśli metody działają obiektach zmieniając, muszą występować samotnie,

.append(el) – dodaje element na końcu listy
.remove(el) – usuwa z listy pierwszy napotkany element
.insert(index,el) – dodaje element do listy na wskazanym indeksie (index)
.reverse() – ustawienie elementów listy w odwrotnej kolejności
.count(el) – zlicza ilość występowania danego elementu w obiekcie (+krotka)

.index(el) – wyświetla nr indeksu, na którym stoi pierwszy szukany element (+krotki)
.pop(index) – wyświetla co występuje na danym index-e w liście, jednocześnie usuwając ten elemetn.
.sort() – sortuje elementy obiektu rosnąco (elementy muszą być tego samego typu)
.sort(reverse=True) - sortuje elementy obiektu malejąco (elementy muszą być tego samego typu)
.extend(lista) – dołącza do obiektu wymienioną listę
.clear()-usuwa wszystkie elementy listy – tworzy pustą listę

Stringi – łańcuch znaków

Kod ASCII – Unicod (UTF-…)

ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

np. € -> 8364
Funkcje od kodów znaków
chr(123) – wyświetla znak o kodzie ASCII 123
ord(”K”) – wyświetla kod znaku w kodzie ASCII dla K (np. € -> 8364)
65 535 – ilość znaków w 2B.

metody dla class string (nie zmieniają obiektu klasy string):
.lower() – zamienia wszystkie duże litery na małe
.upper() - zamienia wszystkie małe litery na duże
.capitalize() – jak w zdaniu – pierwsza litera duża, reszta mała
.swapcase() – zamienia małe litery na duże, duże na małe
.title() – jak w nazwach własnych
.center(szerokość) – centruje napis w danej szerokości (szerokość >=1), jeśli szerokość jest <=il. znaków, nic się nie zmienia w położeniu
.lstrip()- usuwa znaki białe (np. spacja) – na początku tekstu
.rstrip() - usuwa znaki białe (np. spacja) – na końcu tekstu
.replace(stary,nowy,[liczba]) – zamienia stary łańcuch na nowy, liczba oznacza ile pierwszych łańcuchów ma być zamienione, domyślnie jest wszystkie)
.find(co?) – na którym indeksie stoi pierwszy szukany znak

.rfind(co?) – na którym indeksie stoi ostatni szukany znak (od prawej strony)

działania na tekstach:
+ - dołączanie
* - powielanie

Moduły/biblioteki:

Biblioteki załadowuje się do skryptu na samym początku:
from *nazwa_biblioteki* ***import*** *nazwa_funkcji lub **
np.:
from math import sqrt
Znaki globalne
** - dowolny ciąg znaków*
?- dowolny jeden znak

Po załadowaniu (podłączeniu biblioteki) można używać funkcji w niej zdefiniowane tak jak to robimy w przypadku funkcji wbudowanych
inny sposób podłączania biblioteki i wywoływania funkcji w niej zawartej:
from nazwa_biblioteki
.
.
.
nazwa_biblioteki .nazwa funkcji(argumenty)

np. pierwiastek kwadratowy z 4
from math
.
.
.
math.sqrt(4)

Budowa skryptu:

1. biblioteki/moduły
2. definicja:
    - klasy
    - funkcje własne
3. kod właściwy