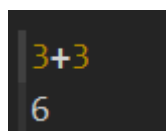


## 4.4 Obliczenia w tabeli atrybutowej (język programowania Python)

Obliczenia w polach tablicy atrybutowej są realizowane w GIS za pomocą wyrażeń i skryptów (krótkich programów) w języku programowania Python. Python jest obecnie podstawowym językiem programowania służącym do przetwarzania danych (data science). Jego niewątpliwą zaletą jest prostota, dzięki czemu łatwo go opanować i używać.

Język programowania to język komunikacji użytkownika z komputerem. Tak jak każdy inny język używa się w nim słów i odpowiedniej składni, z których buduje się instrukcje do wykonania przez komputer. Python realizuje instrukcję po instrukcji, zamieniając je kolejno do postaci zrozumianej przez komputer (na tzw. język maszynowy) i wykonując. Do pisania programów używa się specjalnego programu pełniącego rolę środowiska programowania zwanego IDE (od Interactive Development Environment). GIS zawiera szereg gotowych miejsc gdzie będziemy mogli programy w Pythonie tworzyć i wykonywać.

Najprostszym rodzajem instrukcji jest **wyrażenie** (expression). Przykładem takiego wyrażenia jest:



Podobnie jak w kalkulatorze wyrażenie **3+3** zostanie sprowadzone do pojedynczej wartości **6**. Wyrażenia składają się z **wartości** (3,3) i **operatorów** (+) i mogą być zawsze sprowadzone do pojedynczej wartości. W związku z tym wyrażenia mogą być używane w każdym miejscu naszego programu w którym oczekiwania jest wartość.

Najczęściej będziemy używali operatorów matematycznych. W tabeli zestawiono je w kolejności wykonywania.

Operator	Operacja	Przykład	Wynik
**	potęgowanie	2 ** 3	8
%	reszta z dzielenia	22 % 8	6
//	dzielenie całkowite	22 // 8	2
/	dzielenie	22 / 8	2.75
*	mnożenie	3 * 4	12
-	odejmowanie	7 - 5	2
+	dodawanie	2 + 3	5

Kolejność wykonywania może być zmieniona za pomocą wykorzystania nawiasów.

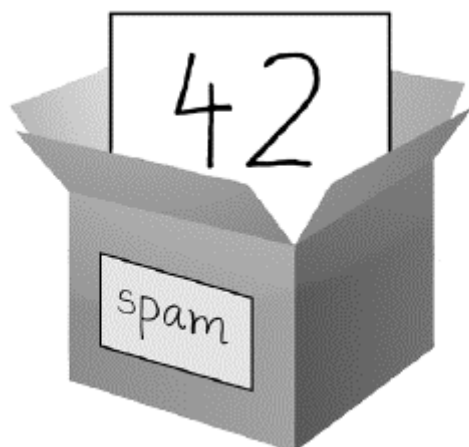
Każda wartość należy do określonego **typu danych**. Podstawowe typy danych w Pythonie to:

Typ danych	Przykład
Całkowite (integer)	-12, 0, 3, 234, 17
Rzeczywiste (float)	-1.30, 0.0, 12.01, 13.8
Tekstowe (string)	'klasa', 'A', 'koniec programu nr. 3',

Operatory mogą mieć różne działanie w zależności od typu danych. Przykład poniżej pokazuje jak działa operator dodawania w przypadku dwóch stringów (tekstów). Wyrażenie **str(3)** dokonuje **konwersji** (zamiany) liczby całkowitej **3** na tekst **'3'**. A operator **+** służy w tym przypadku do łączenia tekstów (concatenate).

```
'Punkt '+str(3)
'Punkt 3'
```

Kolejnym podstawowym pojęciem w programowaniu jest **zmienna**. Można ją rozumieć jako pudełko z nazwą do którego możemy włożyć pojedynczą wartość, a następnie wywoływać tą wartość za pomocą nazwy zmiennej (pudełka).



Proces przypisania wartości do zmiennej odbywa się za pomocą **instrukcji przypisania** np. `spam = 42`. Oznacza to, że aktualnie zmienna *spam* ma wartość całkowitą 42.

Wykorzystanie zmiennej i instrukcji przypisania może mieć postać.

```
spam=42
spam=spam+10
spam+0
52
```

Najpierw przypisano 42 do zmiennej *spam*. Następnie do zmiennej *spam* przypisano wartość otrzymaną z dodania aktualnej wartości jaką posiada zmienna *spam* i liczby 10. Do nowej wartości *spam* dodano zero. Istnieją pewne restrykcje co do tworzenia nazw. Przyjęto, że nazwy zaczynają się literą (bez polskich), a po nich mogą następować litery, cyfry oraz znak podkreślnika `_`. Nie można w nazwach używać takich znaków jak: `#`, `&`, `@` i `%` oraz spacji. Python jest językiem „case sensitive” oznacza to, że Temperatura i temperatura to dwie różne nazwy. Istnieje także pewien zbiór słów zarezerwowanych, które nie mogą być używane jako nazwy zmiennych.

Klasycznym pierwszym programem jest wydrukowanie tekstu „Witaj świecie!”. W Pythonie taki program będzie miał postać.

```
print('Witaj świecie!')
Witaj świecie!
```

**Print** to funkcja wykonująca pewną operację na argumencie znajdującym się wewnątrz nawiasów. Wewnątrz nawiasów znajduje się tekst (string) „Witaj świecie!”. Funkcja wydrukuje go na ekranie. Język Python zawiera wiele różnych funkcji. Na przykład funkcje **int( )**, **float( )**, **str( )** służą do zamiany typu danych, a funkcja **round(x,2)** zaokrągla wartość x do 2 miejsc po kropce.

Program (czasem nazywany skryptem) składać się może z szeregu linii. Kolejność ich wykonywania określają bloki instrukcji opisujące tzw. **flow control**. Istotnym ich elementem są zmienne typu logicznego (Boolowskie) przyjmujące tylko dwie możliwe wartości PRAWDĘ (True) albo FAŁSZ (False). Wartości logiczne są wynikiem działania operatorów porównania oraz operatorów logicznych:

Operator	Znaczenie / wynik
==	równa się
!=	różne
<	mniejsze
>	większe
<=	mniejsze lub równe

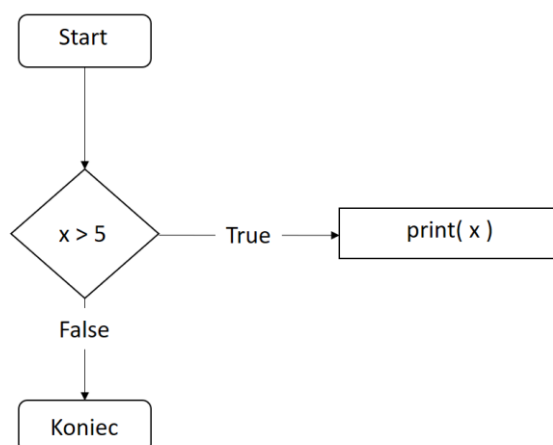
<code>&gt;=</code>	większe lub równe
<code>True and True</code>	True
<code>True and False</code>	False
<code>False and True</code>	False
<code>False and False</code>	False
<code>True or True</code>	True
<code>True or False</code>	True
<code>False or True</code>	True
<code>False or False</code>	False

W naszych zastosowaniach podstawową konstrukcją określającą *flow control* będzie blok instrukcji oparty o **if**. W najprostszej wersji przyjmie on postać.

```
x=7
if x > 5:
    print(x)
7
```

Zmiennej *x* została przypisana wartość 7. Instrukcja **if** zawiera warunek (*x* > 5) jeśli warunek jest spełniony (True) wykonane zostaną linie znajdujące się pod **if** (blok linii zaznaczony wcięciem). W tym przykładzie warunek jest spełniony i na ekranie zostanie wypisana wartość *x*.

Realizacja kodu (flow control) może być przedstawiona w postaci graficznej jako:

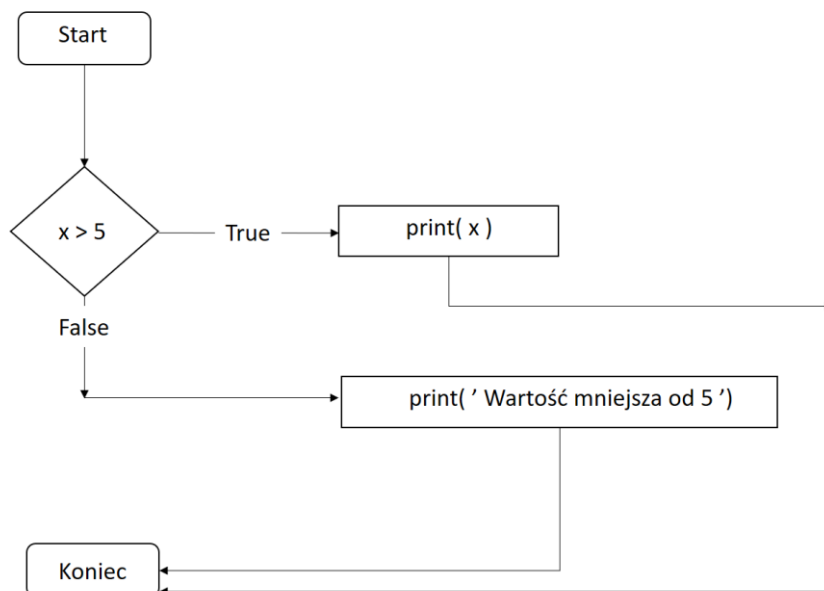


Rysunek 4.19 Realizacja bloku instrukcji IF.

Rozwinięciem tej konstrukcji jest zastosowanie **else**, które oznacza w *przeciwnym razie*.

```
x=4
if x > 5:
    print(x)
else:
    print('Wartość mniejsza od 5')
Wartość mniejsza od 5
```

Realizacja bloku instrukcji if-else można przedstawić graficznie jako:



Rysunek 4.20 Realizacja bloku instrukcji IF-ELSE.

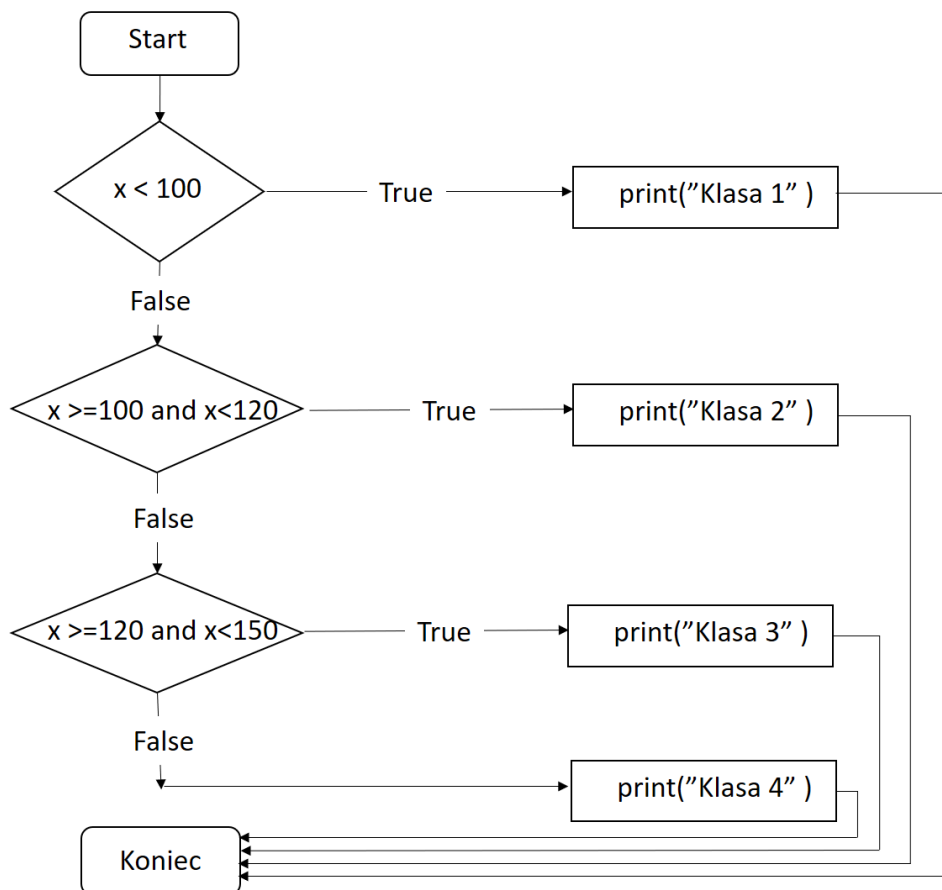
Kolejnym rozwinięciem tej instrukcji jest użycie **elif**.

```

x=122
if x < 100:
    print('Klasa 1')
elif x >= 100 and x < 120:
    print('Klasa 2')
elif x >= 120 and x < 150:
    print('Klasa 3')
else:
    print('Klasa 4')
Klasa 3

```

**Elif** pozwala budować złożoną strukturę warunkową, której istotną zaletą jest zatrzymanie działania po napotkaniu prawdy. Graficzne przedstawienie tej konstrukcji będzie miało postać:



Rysunek 4.21 Realizacja bloku instrukcji IF-ELIF-ELSE.

Wszystkie programy w Pythonie mogą korzystać z zestawu podstawowych funkcji (built-in functions). Oprócz tego Python posiada szereg standardowych bibliotek (standard library), które zawierają najczęściej potrzebne funkcje. Na

przykład biblioteka **math** zawiera funkcje matematyczne, a biblioteka **random** generatory liczb losowych. Są one często używane w GIS. Użycie funkcji zawartej w bibliotece wymaga zaimportowania tej biblioteki, wykonuje się to za pomocą instrukcji:

**Import <nazwa\_biblioteki>**

Ten sposób importu biblioteki wymaga poprzedzenia nazwy funkcji nazwą biblioteki i kropką:

**<nazwa\_biblioteki>.<nazwa\_funkcji>( )**

Podstawowe funkcje biblioteki **math** zostały przedstawione poniżej:

Funkcje modułu math	Opis funkcji
ceil(x)	najmniejsza liczba całkowita nie mniejsza od x
floor(x)	największa liczba całkowita nie mniejsza od x
exp(x)	e do potęgi x
log(x)	logarytm naturalny z x
log10(x)	logarytm dziesiętny z x
pow(x,y)	wartość $x^{**}y$
sqrt(x)	pierwiastek kwadratowy z x
pi	stała 3.141599265359
e	stała 2.718281882846
sin(x)	sinus x (x w radianach)
cos(x)	cosinus x (x w radianach)
tan(x)	tangens z (w radianach)
acos(y)	zwraca arcus cosinus (w radianach)
atan(y)	zwraca arcus tangens (w radianach)
degrees(x)	zamienia radiany na stopnie
radians(x)	zamienia stopnie na radiany

Przykładowe zastosowanie funkcji matematycznych może wyglądać w następujący sposób:

```

8 import math
9
10 print(math.sqrt(100))
11
12 fi=123.22 # stopnie
13 rfi=math.radians(fi)
14 print(fi,round(math.cos(rfi),4))

```

Drugą grupą funkcji są funkcje służące do generowania liczb pseudolosowych. Poniżej przedstawiono cztery najczęściej używane, oraz przykładowe ich wykorzystanie.

Funkcje modułu random	Opis funkcji
randint(a,b)	losowa liczba całkowita od a do b
random()	losowa liczba rzeczywista od 0 do 1
gauss(a,b)	losowa liczba rozkładu normalnego o średniej a i odchyleniu standardowym b
uniform(a,b)	losowa liczba rzeczywista od a do b
seed(a)	tworzenie powtarzalnej serii liczb losowych

```

1 import random
2
3 print(random.randint(1,10))
4 print(random.random())
5 print(random.gauss(10,5))
6 print(random.uniform(10,20))

```

W Pythonie stworzono wiele bibliotek z najróżniejszymi funkcjami. Jest to ogromna wartość tego języka. Niemniej można także tworzyć w prosty sposób własne funkcje, co okazuje się bardzo przydatne w procesie programowania. Funkcja pełni rolę mini-programu w naszym programie, ale także pozwala na zawarcie wielu linii instrukcji w jednej linii (sprowadzonej do realizacji funkcji).

Przyjrzyjmy się przykładowej funkcji i metodzie jej wykorzystania.



```
1 - def hallo() :  
2     print('Pozdrowienia')  
3     print('Dla Wszystkich')  
4  
5 hallo()  
6 hallo()  
7 hallo()
```

Pierwsza linia zawiera słowo **def** , które definiuje funkcję o nazwie **hallo( )** . Pod funkcją znajduje się blok kodu (wcięte linie 2 i 3), jest to tzw. *body of function*. Ten kod będzie wykonany kiedy funkcja zostanie wykonana. Nie zostanie on wykonany przy definiowaniu funkcji. Linie 5,6 i 7 to trzykrotne wywołanie funkcji powodujące jej wykonanie. Kiedy wykonamy cały program (linie 1-7) rezultat będzie miał postać.

```
Pozdrowienia  
Dla Wszystkich  
Pozdrowienia  
Dla Wszystkich  
Pozdrowienia  
Dla Wszystkich
```

Podobnie jak w zdefiniowanych już funkcjach np. `round(2.341,1)` w nawiasach możemy umieszczać argumenty funkcji. Aby było to możliwe w procesie definiowania określamy parametry funkcji. Parametr to zmienna zawierająca argument funkcji. Nasza funkcja z parametrem przyjmie postać.

```
1 - def hallo(kto) :  
2     print('Pozdrowienia')  
3     print('Dla Wszystkich ' +kto)  
4  
5 hallo('białych')  
6 hallo('czarnych')  
7 hallo('żółtych')
```

Rezultat działania funkcji będzie miał postać,

```
Pozdrowienia
Dla Wszystkich białych
Pozdrowienia
Dla Wszystkich czarnych
Pozdrowienia
Dla Wszystkich żółtych
```

Jedna istotna uwaga. Nasz parametr (zmienna) **kto** istnieje wyłącznie wewnątrz funkcji. Funkcje przedstawione do tej pory wykonywały pewną czynność (drukowanie), ale aby za pomocą funkcji przypisać wartość w tabeli atrybutowej musi ona zwracać pewną wartość, tak jak na przykład działa funkcja `round()`.

```
1 x=8.3478
2 y=round(x,2)
3 print(y)
```

Rezultat działania programu,

```
8.35
```

Aby funkcja zwracała wartość używamy słowa `return` przy jej definiowaniu. Przyjrzyjmy się funkcji służącej do klasyfikacji jednej zmiennej. Nasza funkcja w zależności od sytuacji zwróci wartość -1, 0 albo 1.

```
1 - def klasyfikacja(x):
2 -     if x>0:
3 -         klasa=1
4 -     elif x<0:
5 -         klasa=-1
6 -     else:
7 -         klasa=0
8 -     return klasa
9
10 print(klasyfikacja(123))
11 print(klasyfikacja(-123))
12 print(klasyfikacja(0))
```

Oczywiście wynikiem działania tego programu będzie,

```
1
-1
0
```



