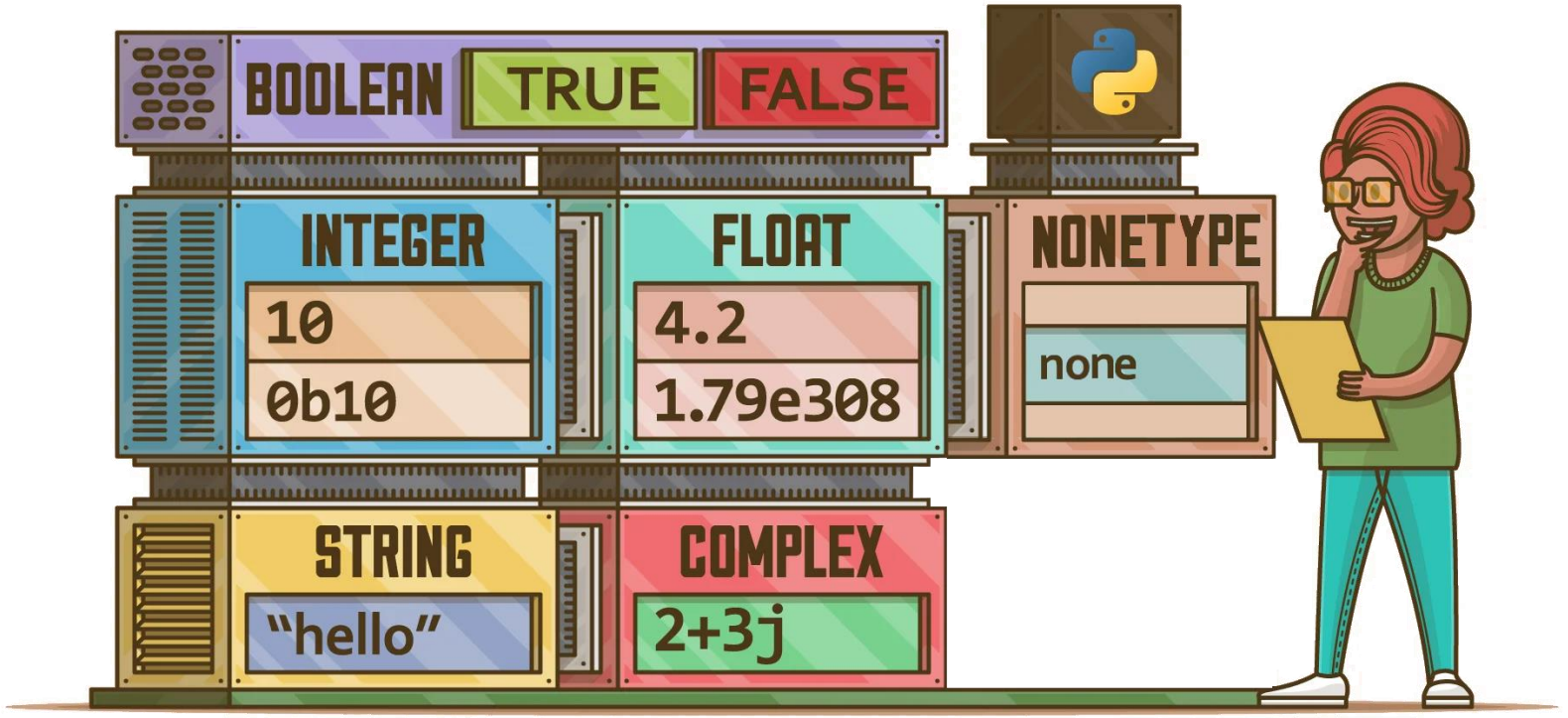
Typy proste


BOOLEAN
TRUE
FALSE
INTEGER
10
0b10
FLOAT
4.2
1.79e308
NONETYPE
none
STRING
“hello”
COMPLEX
2+3j
Real Python

Wartości

W Pythonie posługujemy się wartościami, np. 5, 7, 'ala'.

Widać, że wartość 5 i 7 różnią się od siebie. Dokładnie o dwa.

Widać, że wartości 5 i 'ala' też różnią się od siebie. Ale tym razem w jakiś bardziej fundamentalny sposób. Nie potrafimy ich nawet ze sobą porównać. Czy te wartości różnią się od siebie?

Typem, każda wartości ma jakiś typ.

5 i 7 to wartości tego samego typu - całkowitoliczbowego, 'ala' to wartość typu napisowego.

A ile wyróżniamy typów wartości w Pythonie?

Kategorie typów

Wyróżniamy dwie podstawowe kategorie typów:

- **typy proste** – pojedyncze wartości
- **typy złożone** – złożone z wielu elementów (typów prostych)

Typy proste

Do typów prostych zaliczamy:

- integer (typ całkowitoliczbowy)
- float (typ zmiennoprzecinkowy)
- complex (typ zespolony)
- string (typ napisowy)
- boolean (typ logiczny)
- none (typ pusty)

Przykładowe literały całkowitoliczbowe
(integer literals)

```
0
235462
-34
0b10101
0o350
0x12b
```

Przykładowe literały zmiennoprzecinkowe
(floating-point literals)

```
0.0
4.235
-3.53
4341.
.412
32.32e4
-.35e-9
```

Przykładowe literały zespolone
(complex literals)

```
0.0+0j
0j
1+4j
4.235-3.24j
1+32.32e4j
-.35e-9+2j
```

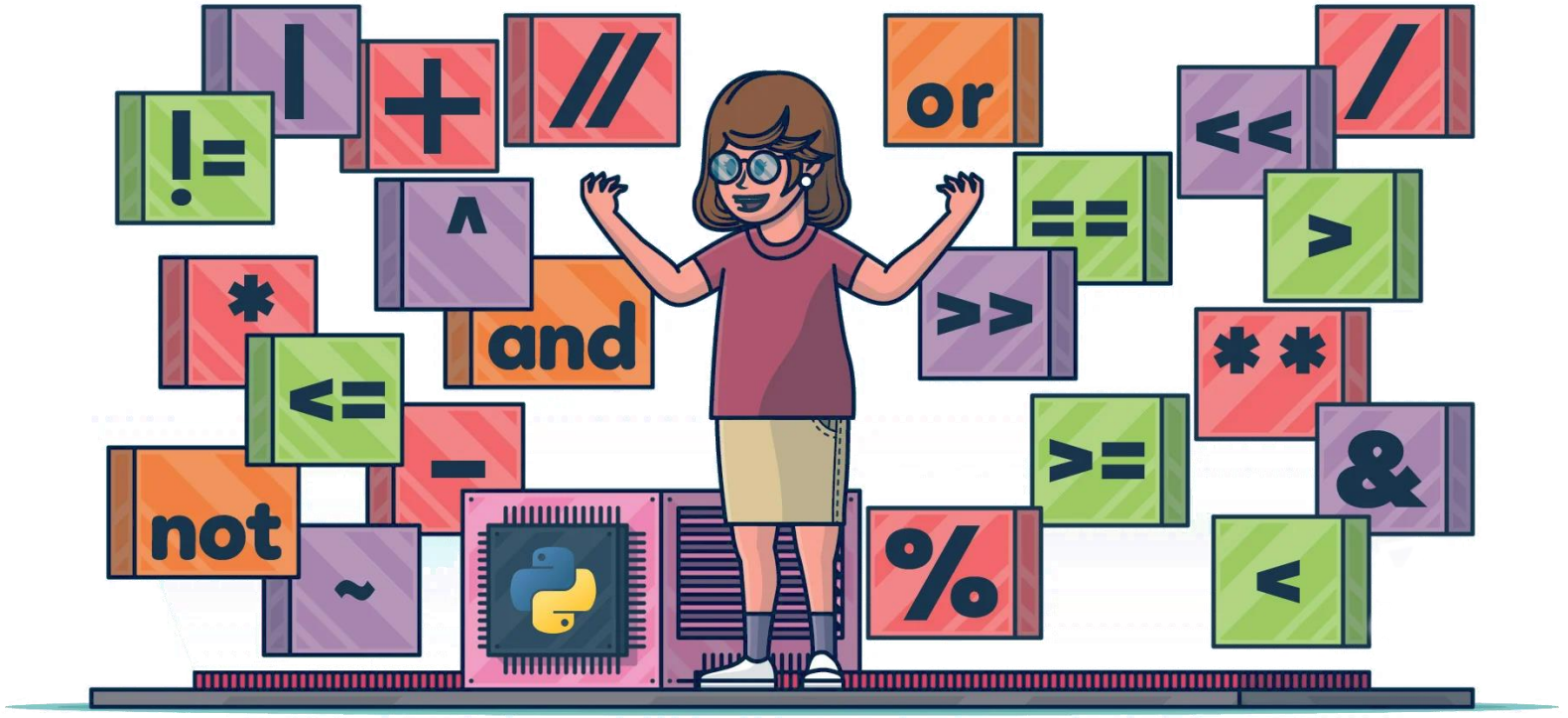
Przykładowe literały napisowe
(string literals)

```
""
''
"ala ma kota"
'दिलचस्प'
"有趣的 интересный"
'😀'
"3.23"
r"napis \n"
f'sth'
"""napis wielolinijkowy"""
```

Literały logiczne
(boolean literals)

Literał pusty
(none literal)

Operatory (operacje)



Real Python

Wyróżniamy 7 kategorii operatorów

Operatory arytmetyczne
(arithmetic operators)

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Floor division
%	Modulus
**	Exponentiation

Operatory porównania
(relational operators)

Operator	Meaning
>	Greater than
<	Less than
==	Equal to
!=	Not equal to
>=	Greater than or equal to
<=	Less than or equal to

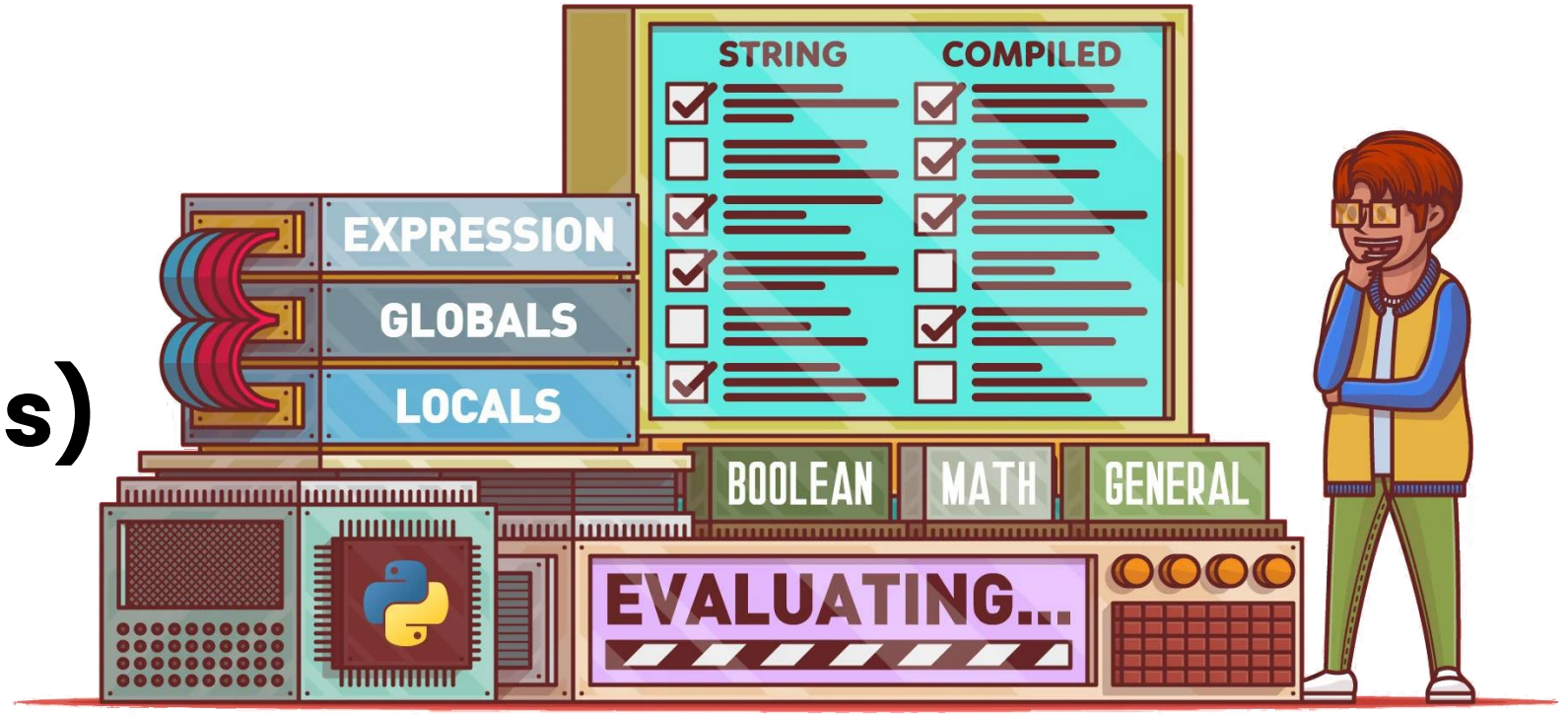
Operatory logiczne
(logical operators)

Operator	Meaning
and	True if both the operands are true
or	True if either of the operands is true
not	True if operands is false (complements the operand)

Operatory bitowe
(bitwise operators)

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
~	Bitwise NOT
^	Bitwise XOR
>>	Bitwise right shift
<<	Bitwise left shift

Wyrażenia (expressions)



Alternatywna klasyfikacja operatorów

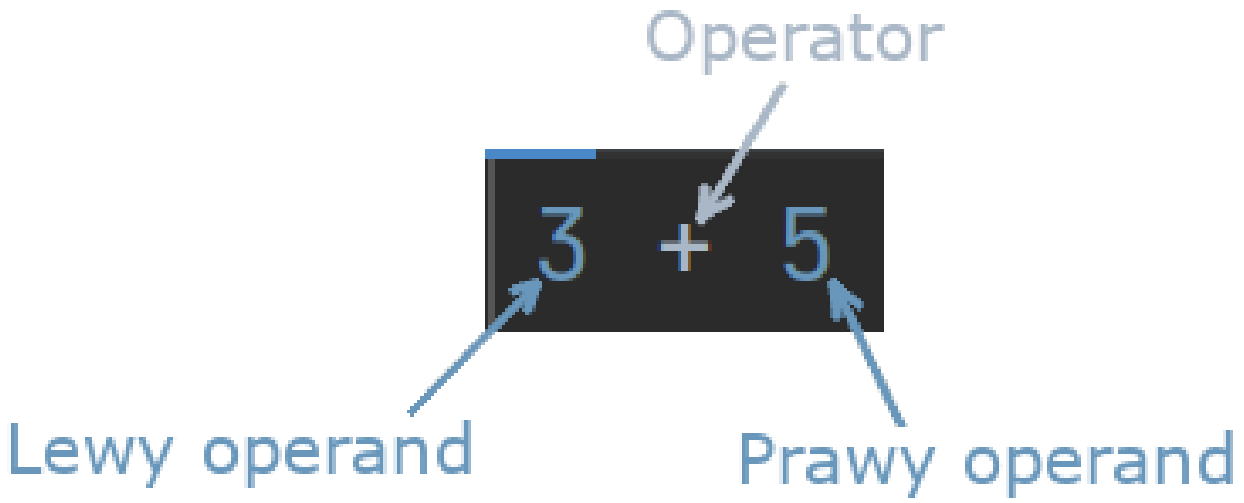
- operatory jednoargumentowe

Operator	Meaning
not	True if operands is false (complements the operand)

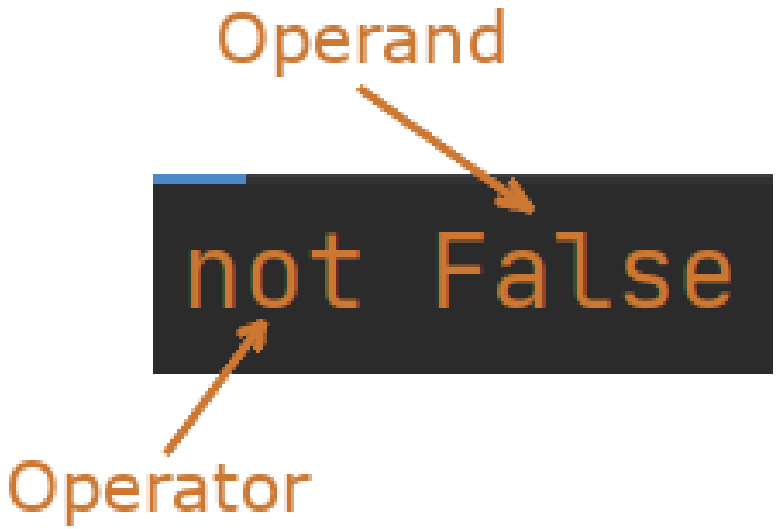
- operatory dwuargumentowe

Operator	Meaning
+	Addition

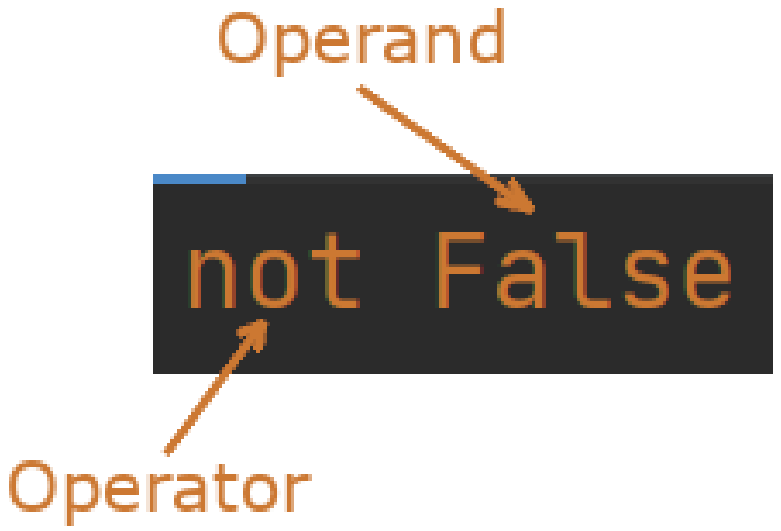
Anatomia wyrażenia
(dla operatorów dwuargumentowych)



Anatomia wyrażenia
(dla operatorów jednoargumentowych)



Składanie wyrażeń



Składanie wyrażeń

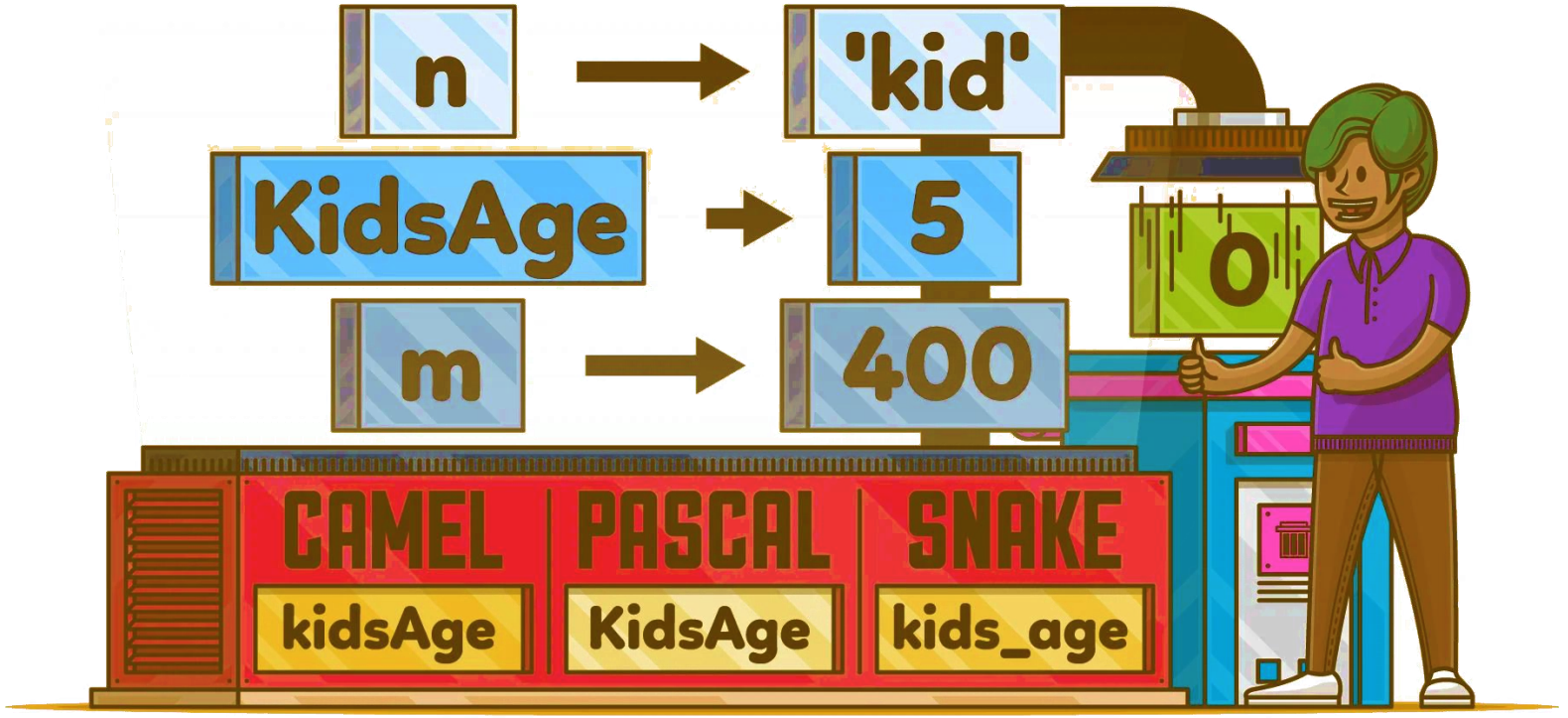
Wyrażenia możemy ze sobą składać.

```
2 + 3 - 4 * 2 // 3
```

Ale co z pierwszeństwem?

Operators	Definitions	Precedence
()	Parenthesis	Highest
**	Exponentiation	
~	Bitwise NOR	
+, -	Sign (positive, negative)	
*, /, //, %	Multiplication, division, floor division, modulus division	
+, -	Addition, subtraction	
&	Bitwise AND	
^	Bitwise XOR	
|	Bitwise OR	
<, <=, >, >=, ==, !=, is, is not	Relational operators, membership operators	
not	Boolean (Logical) NOT	
and	Boolean (Logical) AND	
or	Boolean (Logical) OR	Lowest

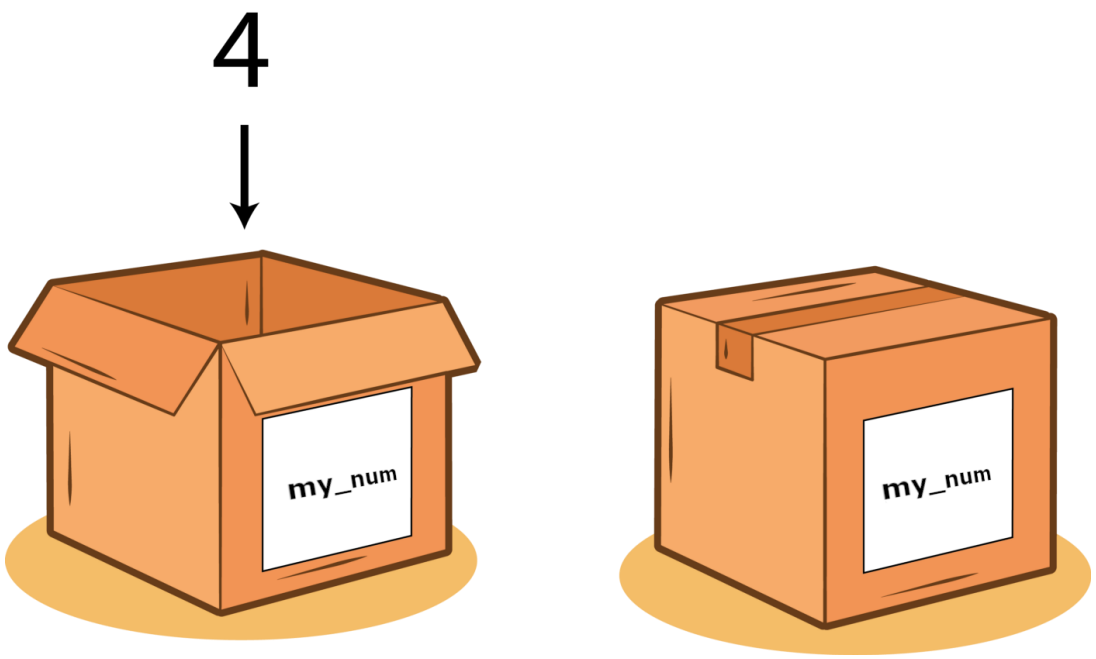
Zmienne (variables)



Real Python

Po co zmienne?

Żebyśmy mogli przechowywać gdzieś wartości do użycia w dalszych miejscach naszego programu.

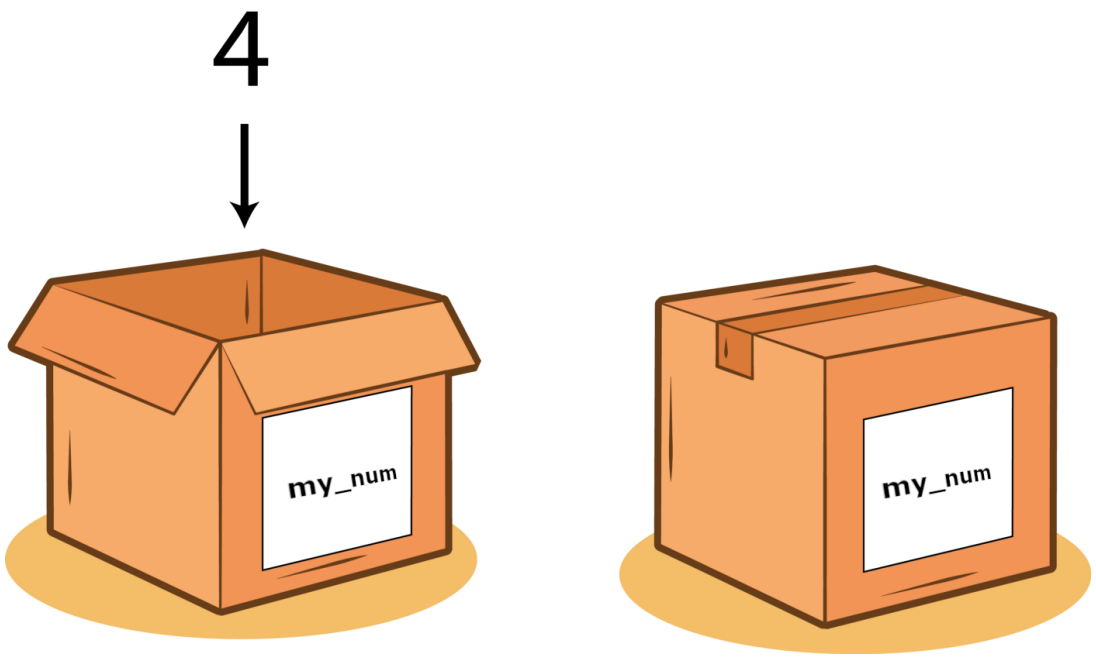


Zmienne

```
my_num = 4
```

Zmienne

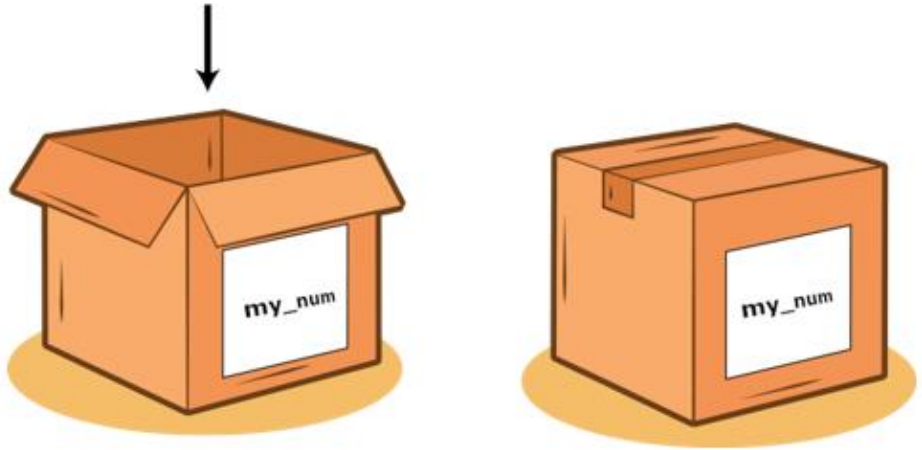
```
my_num = 4
```



Zmienne

```
my_num = 4 + 3 // 2
```

3 + 4 // 2 → 5



Operator przypisania

Operator przypisania

Weź to co po prawej stronie, oblicz i wynik przypisz do tego co po lewej stronie. Po prawej stronie operatora mamy wyrażenie, a po lewej nazwę zmiennej.

```
my_num = 4 + 3 // 2
```

Operatory przypisania

Operator	Meaning
=	Assign a value
+=	Adds and assign the result to the variable
-=	Subtracts and assign the result to the variable
*=	Multiplies and assign the result to the variable
/=	Division and assign the result to the variable
//=	Floor division and assign the result to the variable
%=	Find modulus and assign the result to the variable
**=	Find Exponentiation and assign the result to the variable
&=	Find Bitwise AND and assign the result to the variable
|=	Find Bitwise OR and assign the result to the variable
^=	Find Bitwise XOR and assign the result to the variable
>>=	Find Bitwise right shift and assign the result to the variable
<<=	Find Bitwise left shift and assign the result to the variable

Instrukcje sterujące (control flow statements)

Instrukcje sterujące

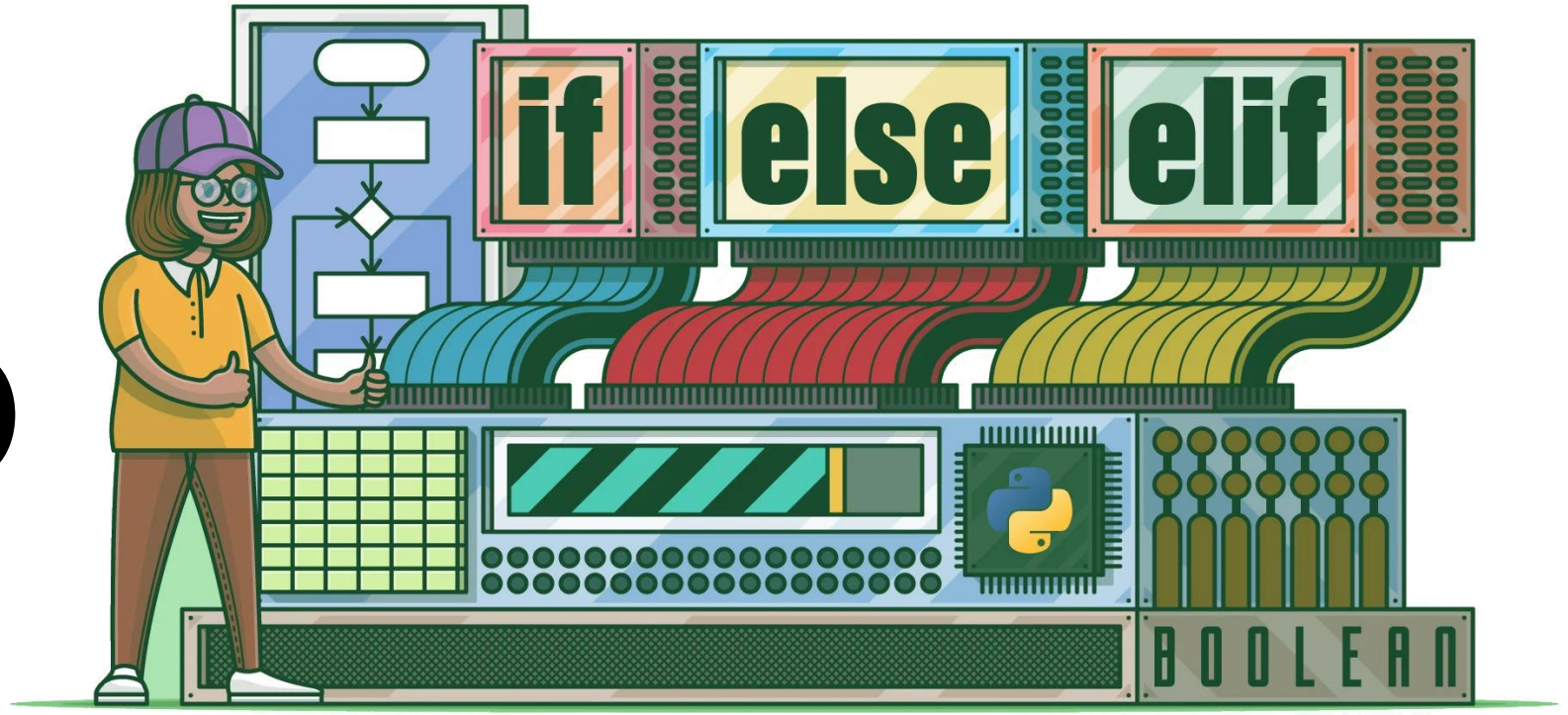
Do instrukcji sterujących przepływem kontroli w programie zaliczamy:

- następowanie
- rozgałęzienia
- powtórzenia

W Pythonie:

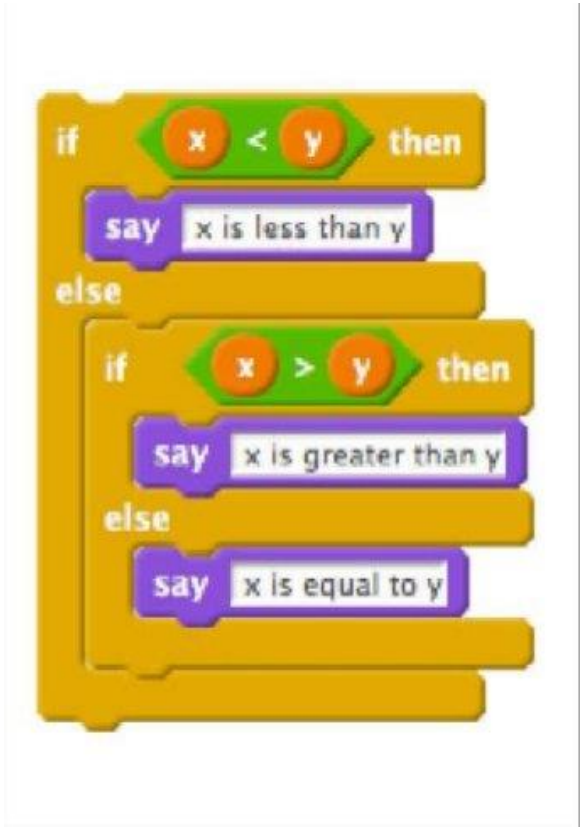
- następowanie jest domyślnie realizowane przez interpreter
- rozgałęzienia są implementowane przez **warunki**
- powtórzenia są implementowane przez **pętle**

Warunki (conditions)



Real Python

Warunki




```
if x < y:
    print("x jest mniejsze od y")
elif x > y:
    print("x jest większe od y")
else:
    print("x jest równe y")
```

Anatomia warunku

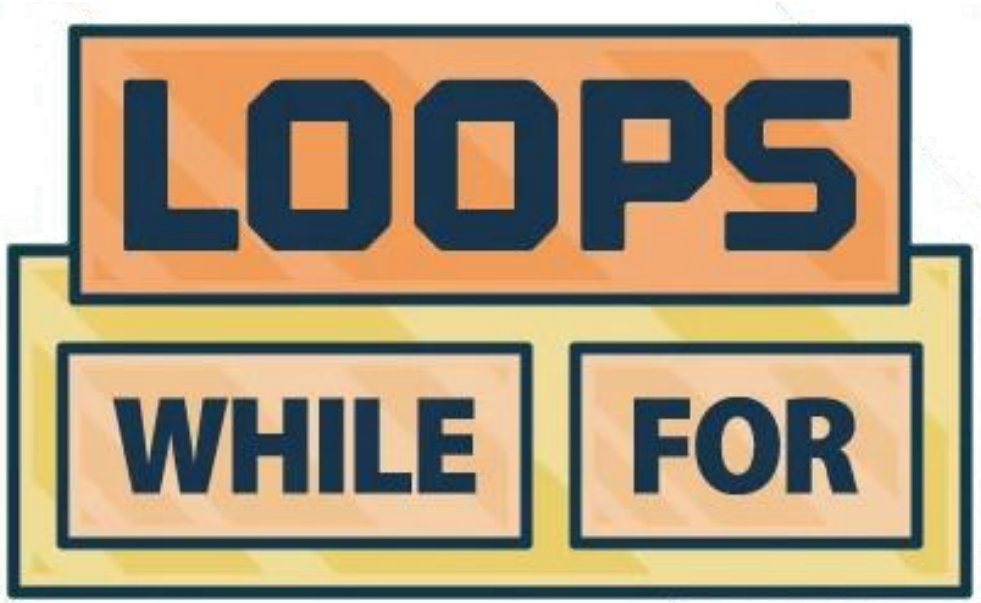
Anatomia warunku

```
if <typ boolean>:
    <blok kodu>
else:
    <blok kodu>
```

Anatomia warunku

```
if <typ boolean>:
    <blok kodu>
elif <typ boolean>:
    <blok kodu>
else:
    <blok kodu>
```

Pętle (loops)



Pętle

W Pythonie wyróżniamy dwa rodzaje pętli:

- pętla while
- pętla for

Na tym etapie poznamy pętle while. Pętle for wprowadzimy w trakcie przeglądu typów złożonych.

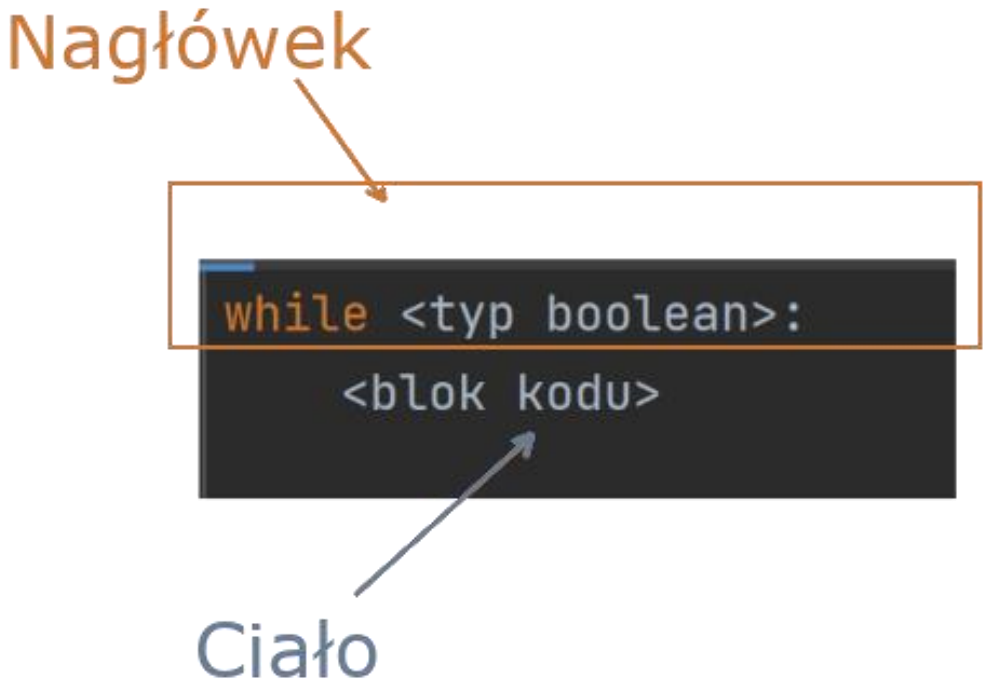
Pętla while

Real Python

Pętla while


```
while True:
    print("hello, world")
```

Anatomia pętli while



Terminatory pętli

Do instrukcji specjalnych używanych **wewnątrz** pętli (*aka terminatory pętli*) należą:

- instrukcja **break** – natychmiast opuszcza pętle
- instrukcja **continue** – natychmiastowo przechodzi do kolejnego obrotu pętli. Ignoruje tą część ciała pętli, która znajduje się pod instrukcją continue

Dziękujemy!

software
development
academy