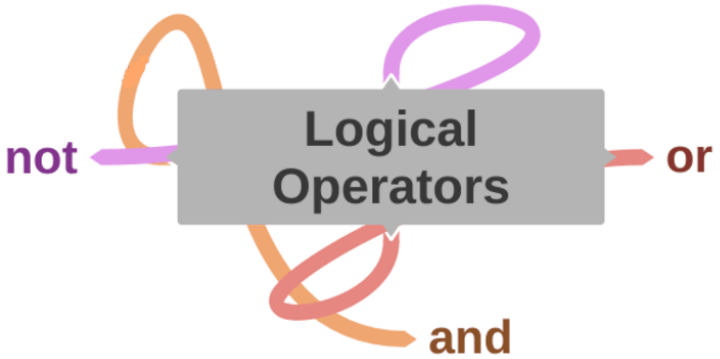
Operatory logiczne i bitowe

Podstawy programowania w języku Python



Złożone wyrażenia logiczne

- operatory logiczne pozwalają budować złożone wyrażenia logiczne

jeżeli będzie dodatnia temperatura, i będzie słonecznie, to…
pójdziemy na spacer; a jeżeli nie, to zostaniemy w domu

```
1  temperature = 12
2  is_sun_shining = False
3
4  if (temperature > 0 and is_sun_shining):
5      print("Idziemy na spacer")
6  else:
7      print("Zostaniemy w domu")
```

Operatory logiczne

- służą do budowania wyrażeń logicznych
- wykonują operacje na wartościach logicznych **True** i **False**
- wynikiem tych operacji są wartości **True** lub **False**

Operator	Znaczenie	Przykład	Wynik
and	koniunkcja	True **and** False	False
or	alternatywa	True **or** False	True
not	negacja	**not** True	False

Tabele prawdy

operator **and**

A	B	A **and** B
False	False	False
False	True	False
True	False	False
True	True	True

operator **or**

A	B	A **or** B
False	False	False
False	True	True
True	False	True
True	True	True

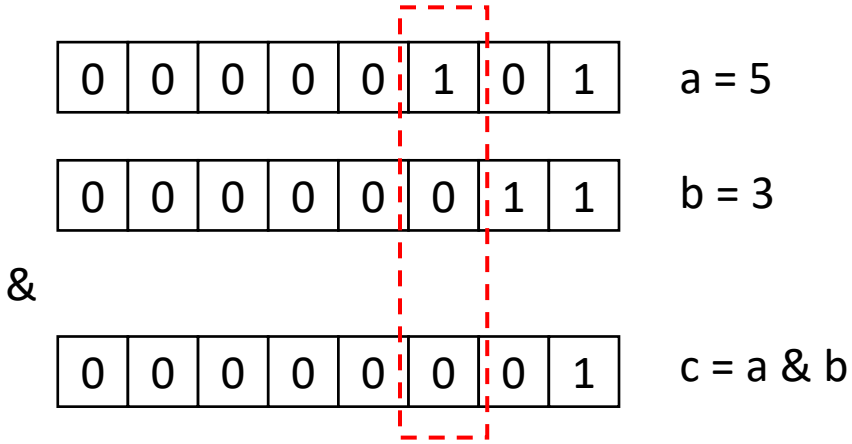
operator **not**

A	**not** A
False	True
True	False

Operacje na bitach

- liczby reprezentowane są przez serie bitów
- operatory bitowe ingerują w każdy bit z osobna
- każda para bitów poddawana jest ewaluacji operatora logicznego

```
1    a = 5
2    b = 3
3    c = a & b
4
5    print(c) #wynik: 1
```



Operatory bitowe

- pozwalają manipulować pojedynczymi bitami
- przyjmują wyłącznie argumenty w postaci liczb całkowitych

Operator	Znaczenie	Przykład	Wynik
&	koniunkcja bitowa	2 **&** 3	2
|	alternatywa bitowa	2 **|** 3	3
~	negacja bitowa	**~** 3	-4
^	alternatywa rozłączna bitowa	2 **^** 3	1
>>	przesunięcie bitowe w prawo	2 **>>** 1	1
<<	przesunięcie bitowe w lewo	2 **<<** 1	4

Tabele prawdy

operatory bitowe **&**, **|**, **^**

bit A	bit B	A **&** B	A **|** B	A **^** B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

operator bitowy **~**

bit A	~ A
0	1
1	0

Przesunięcia bitowe w prawo

- operator **>>** wykona bitowe ”przesunięcie w prawo”, w którym wartość lewego operandu jest przesuwana w prawo o liczbę bitów podaną przez prawy operand

```
wartość >> bity
```

```
1  # 8 = 0b1000
2  8 >> 2
3  # wynik: 2
4  # 2 = 0b10
5
6  bin(8 >> 2)
7  # wynik: 0b10
```

Przesunięcia bitowe w lewo

- operator **<<** wykona bitowe ”przesunięcie w lewo”, w którym wartość lewego operandu jest przesuwana w lewo o liczbę bitów podaną przez prawy operand

```
wartość << bity
```

```
1  # 2 = 0b10
2  2 << 2
3  # wynik: 8
4  # 8 = 0b1000
5
6  bin(2 << 2)
7  # wynik: 0b1000
```

Priorytety operatorów

Priorytet	Operator	
1	+, −	jednoargumentowe
2	**	
3	*, /, //, %	
4	+, −	dwuargumentowe
5	<<, >>	
6	<, <=, >, >=	
7	==, !=	
8	&	
9	|	
10	=, +=, −=, *=, /=, %=, &=, ^=, |=, >>=, <<=	

Pytanie

Które z wyrażeń najlepiej opisuje poniższy warunek złożony:

”Jeżeli liczba będzie parzysta lub większa od pięciu to…”

a) `number % 0 == 2 and number > 5`
b) `number % 2 == 0 or number > 5`
c) `number % 2 == 0 and number > 5`
d) `number % 2 == 0 or number >= 5`

Odpowiedź: b)

Pytanie

Co wyświetli się na ekranie po wykonaniu poniższego skryptu?

a) nic się nie wyświetli
b) Tak
c) Nie
d) wystąpi błąd

```
1  a = 7
2  b = 4
3  cond = a > b
4  if (not cond):
5      print("Tak")
6  else:
7      print("Nie")
```

Odpowiedź: c)

Pytanie

Jaki będzie wynik poniższej operacji?

```
a) 4
b) 1
c) 0
d) wystąpi błąd
```

```
1    a = 4
2    print(a >> a)
```

Odpowiedź: c)
4 to binarnie 0100, po przesunięciu bitowym w prawo o 4 pozycje otrzymamy 0.