Przykład:

```
$ python3
>>> x=78
>>> hex(x)
'0x4e'
>>> oct(x)
'0o116'$
```

Operacje arytmetyczne na liczbach dwójkowych

Arytmetyka liczb dwójkowych jest podobna do dziesiętnych (stosujesz reguły „słupkowe" poznane w szkole podstawowej!).

Popatrz na operacje elementarne (dwójka w indeksie dolnym oznacza umownie, że mamy do czynienia z liczbą binarną, dziesiątka — z liczbą w systemie dziesiętnym):

- $0_2+0_2 = 0_2$, $0_2+1_2 = 1_2$, $1_2+0_2 = 1_2$, $1_2+1_2 = 10_2$ (przeniesienie),
- $0_2-0_2 = 0_2$, $1_2-0_2 = 1_2$, $1_2-1_2 = 0_2$, $0_2-1_2 = 1_2$ oraz pożyczka z kolejnej pozycji.

Pożyczka oznacza nic innego jak dodatkowe odjęcie 1 od wyniku odejmowania cyfr w następnej kolumnie.

Kilka prostych przykładów znajdziesz w tabeli 2.3.

TABELA 2.3. Arytmetyka liczb dwójkowych na przykładzie

	PRZYKŁAD	KOMENTARZ
Dodawanie	A = 11011 = 27_{10} B = 11001 = 25_{10} A+B = 110100	Pamiętaj o przeniesieniu 1 dla (1+1)!
Odejmowanie	A = 11010 = 26_{10} B = 10001 = 17_{10} A-B = 1001	W przypadku odejmowania (0-1) musimy wykonać „zapożyczenie" 1 na następnej pozycji liczby.
Mnożenie	A = 101 = 5_{10} B = 010 = 2_{10} 000 101 A·B = 1010 = 10_{10}	Tak jak w systemie dziesiętnym (mnożysz przez kolejne cyfry i przesuwasz się w lewo).
Dzielenie	110 1100:10 = $12_{10}:2_{10} = 6_{10}$ -10 0100 10 000	Tak jak w systemie dziesiętnym (odejmujesz przesunięty na lewo dzielnik od dzielnej aż do uzyskania 0 lub reszty).

Jeśli w programach Pythona zajdzie potrzeba wykonywania **operacji na pojedynczych bitach** pewnych liczb binarnych, które uczestniczą w wyrażeniu, to możesz użyć dedykowanych *operatorów bitowych*, które pokazałem w tabeli 2.4.

TABELA 2.4. Operatory bitowe w języku Python

SYMBOL	ZNACZENIE	ZNACZENIE
&	AND (koniunkcja)	Ustawia wynikowy bit na *1*, gdy oba bity w wyrażeniu są równe *1*, w przeciwnym razie *0*.
|	OR (alternatywa)	Zwraca *0*, gdy oba bity są równe *0*, w przeciwnym razie *1*.
~	Negacja	Zamienia *0* na *1*, *1* na *0*.
^	XOR	Równa się *1*, gdy oba bity są różne, w przeciwnym razie *0*.
>>	Przesunięcie bitowe w prawo	Skrajne *prawe* bity „wypadają", a z lewej strony są dodawane zera. Przykład: dla `a=5` (dwójkowo: 101), wyrażenie `a>>1` jest równe *2* (dwójkowo: 10).
<<	Przesunięcie bitowe w lewo	Skrajne *lewe* bity „wypadają", a z prawej strony są dodawane zera. Przykład: dla `a=5` (dwójkowo: 101), wyrażenie `a<<1` jest równe *10* (dwójkowo: 1010).

W celu głębszego zrozumienia operacji bitowych możesz użyć gotowego skryptu zapisanego w pliku *binarnie.py*. Stworzyłem go jako szkielet kodu umożliwiający zapoznanie się z operatorami stosownymi w Pythonie. Jego składnia jest na tyle prosta, że możesz równie dobrze przetestować wybrane wyrażenia bezpośrednio w konsoli Pythona!

```
print("bin(46)=", bin(46))
print("Liczba 5 << 1 to 10 (operacje są wykonywane na bitach!):\n", \
      5<<1, "- binarnie: ", bin(5<<1) )
a=0b1011     # Liczba 11, prefiks 0b oznacza liczbę binarną
b=0b0010     # Liczba 2, wiodące zera są nieznaczące
print("a=", a, ", binarnie", bin(a))
print("b= ", b, ", binarnie", bin(b))
print("a^b=", bin(a^b))
print("a|b=", bin(a|b))
print("a&b=", bin(a&b))
print("a<<2:", a<<2, ", binarnie:", bin(a<<2))
print("b>>1:", b>>1, ", binarnie:", bin(b>>1))
```

Uruchom ten skrypt w terminalu linii poleceń:

```
$ python3 binarnie.py
bin(46)= 0b101110
Liczba 5 << 1 to 10 (operacje są wykonywane na bitach!):
10 - binarnie:  0b1010
a= 11 , binarnie 0b1011
b=  2 , binarnie 0b10
```

TABELA 2.6. Kod ASCII — kody sterujące

DEC	HEX	ZNAK	SKRÓT
0	00	*Null*	NUL
1	01	*Start of Heading*	SOH
2	02	*Start of Text*	STX
3	03	*End of Text*	ETX
4	04	*End of Transmission*	EOT
5	05	*Enquiry*	ENQ
6	06	*Acknowledge*	ACK
7	07	*Bell*	BEL
8	08	*Backspace*	BS
9	09	*Horizontal Tab*	HT
10	0A	*Line Feed*	LF
11	0B	*Vertical Tab*	VT
12	0C	*Form Feed*	FF
13	0D	*Carriage Return*	CR
14	0E	*Shift Out*	SO
15	0F	*Shift In*	SI
16	10	*Data Link Escape*	DLE
17	11	*Device Control* 1 (XON)	DC1
18	12	*Device Control* 2	DC2
19	13	*Device Control* 3 (XOFF)	DC3
20	14	*Device Control* 4	DC4
21	15	*Negative Acknowledge*	NAK
22	16	*Synchronous Idle*	SYN
23	17	*End of Transmission Block*	ETB
24	18	*Cancel*	CAN
25	19	*End of Medium*	EM
26	1A	*Substitute*	SUB
27	1B	*Escape*	ESC
28	1C	*File Separator*	FS
29	1D	*Group Separator*	GS
30	1E	*Record Separator*	RS
31	1F	*Unit Separator*	US
32	20	*Spacja*	
127	7F	*Delete*	DEL

TABELA 2.7. Kod ASCII — cyfry, litery alfabetu, znaki interpunkcyjne

DEC	HEX	ZNAK
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-

DEC	HEX	ZNAK
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M

DEC	HEX	ZNAK
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n

TABELA 2.7. Kod ASCII — cyfry, litery alfabetu, znaki interpunkcyjne — *ciąg dalszy*

DEC	HEX	ZNAK
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@

DEC	HEX	ZNAK
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	`

DEC	HEX	ZNAK
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~

Siedem bitów w kodzie ASCII stanowiło pewną zaszłość historyczną i ponieważ komputery operowały już słowami ośmiobitowymi, szybko uzupełniono kod ASCII o kody powyżej 127, gdzie wstawiono znaki semigraficzne, np. służące do tworzenia obramowań w trybie znakowym (np. ╔,╤) oraz znaki narodowe. Rozszerzona część tablicy ASCII zawiera zestaw zależny od kraju i czasami producenta sprzętu komputerowego (drukarki, terminale znakowe). W celu obsługi polskich znaków utworzono wariant kodu ASCII o nazwie ISO 8859-2 (tzw. ISO Latin-2). Wadami standardu ASCII są zamieszanie wprowadzone przez producentów oprogramowania i sprzętu (w zasadzie tylko część kodów 0 – 127 jest jednoznaczna) oraz brak jednoczesnej obsługi wielu języków (do dyspozycji w wersji rozszerzonej standardu ASCII mamy tylko 8 bitów, co daje 256 możliwości). Aby utrudnić programistom życie, podobną rolę w systemie Windows pełnił standard o nazwie Windows-1250 (CP-1250), używany do reprezentacji tekstów w głównych językach środkowoeuropejskich...

Od tego czasu powstało kilkadziesiąt innych standardów, o czym łatwo się przekonać, oglądając różne programy i strony internetowe (czasami w miejsce polskich znaków pojawiają się dziwne symbole). Świetne tabelaryczne zestawienie standardów kodowania polskich znaków znajduje się na stronie *http://pl.wikipedia.org/wiki/Sposób_kodowania_polskich_znaków*.