Operators and Variables

Operators

Type	Notation	Example	Remarks
Addition	+	5 + 6 = 11	
Subtraction	-	5 - 6 = -1	
Multiplication	×	5 × 6 = 30	
Division (float)	/	5 / 6 = 8.33	
Modulus (Remainder after division)	%	5 % 6 = 5, 6 % 5 = 1	For x % y, if x < y, = x
Exponential	**	5 ** 6 = 15625	right to left associativity
Floor Division	//	5 // 6 = 0, 6 // 5 = 1	For x // y, if x < y, =0 (always round down to nearest integer)

Precedence

Precedence Level	Operators	Explanation
1 (highest / comes first)	( )	Parentheses
2	**	Exponent
3	+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
4	*, /, //, %	Multiplication, Division, Floor division, Modulus
5	+, -	Addition, Subtraction
6	<<, >>	Bitwise shift operators
7	&	Bitwise AND
8	^	Bitwise XOR
9	|	Bitwise OR
10	(*) is, is not, in, not in, ==, !=, >, >=, <, <=	Comparisons, Identity, Membership operators
11	not	Logical NOT
12	and	Logical AND
13 (lowest, comes last)	or	Logical OR

(*) The is, is not, in, not in have a higher precedence than the ==, !=, >, >=, <, <=

Variable Naming

- Start with a-z or A-Z or _
- Contain only alphanumeric characters or _
- Case sensitive, X_1 != x_1
- Avoid reserved keywords e.g. if
- Python convention: lower case letters separated by, e.g. count_change

Global vs Local Variable

- A variable which is defined in the main body of a file is called a **global variable**. It will be **visible throughout the file**, and also inside any file which imports that file.
- A variable which is defined inside a function is **local** to that function. It is accessible **from the point at which it is defined until the end of the function**, and exists for as long as the function is executing.
- The parameter names in the function definition behave like local variables, but they contain the values that we pass into the function when we call it.

```
x = 0
def foo_printx():
        print(x) # This 'x' refers to the outer 'x'

foo_printx()
print(x) # This 'x' also refers to the outer 'x'
```

```
x = 0
y = 999
def foo_printx(y): # This 'y' refers to the parameter
        print(y)       # pass-by-value

foo_printx(x)
print(x) # This 'x' refers to the outer 'x'
```

```
x = 0 # Global scope
def foo_printx():
        x = 999 # Local scope, This 'x' is created of new assignment
        print(x) # Local 'x' is born here, will die when the function ends here

foo_printx()
print(x) # This 'x' still refers to the outer 'x'
            # The two 'x' will be different 'x'
            # '999' will only be available within function
```

```python
def foo(x):
        bar(x + 1)
        print(x)

def bar(x):
        x = x + x
        print(x)

print(foo(3))
```

The 'x' in bar is different from the 'x' in foo

Variable Type

Type	Example
int	8, 45, 123
float	2.71828, 3.14159 , 1.0
bool	True, False
str	"POGG", 'poggers'
None	

Get Variable Type

```python
print(type(123))
print(type(123.123))
print(type('123'))
print(type(None))
```

Type casting

```python
print(str(123))
print(float('45.2'))
print(int(23.8))
```

Integer Truncation

The `round` function uses "round half to even" strategy, also known as "bankers' rounding" or "unbiased rounding." When a number is exactly halfway between two possible rounded values, it rounds to the nearest even number.

```python
import math

print("ROUND")
print(round(3)) # 3
print(round(3.49)) # 3
print(round(3.5)) # 4
print(round(2.5)) # 2

print("CEIL (ROUND UP)")
print(math.ceil(3)) # 3
print(math.ceil(3.49)) # 4
print(math.ceil(3.5)) # 4

print("FLOOR (ROUND DOWN)")
print(math.floor(3)) # 3
print(math.floor(3.49)) # 3
print(math.floor(3.5)) # 3
```

Comparing Strings

In Python, when comparing strings, the comparison is done `lexicographically` (i.e., dictionary order). The comparison is performed character by character from left to right.

```python
print('abc' > 'abbbbbbbb') # True
```

1. At the first position, `'a'` is the same in both strings.
2. At the second position, `'b'` is the same in both strings.
3. At the third position, `'c'` is greater than the corresponding character `'b'` in the second string.

```python
print('123' > '11111111') # True
```

Binary Operator

Operator	Name	Description	Syntax
&	Bitwise AND	Result bit 1 if both operand bits are 1; otherwise results bit 0.	x & y
\ (straight line)	Bitwise OR	Result bit 1 if any of the operand bit is 1; otherwise results bit 0.	x \ y
~	Bitwise NOT	Inverts individual bits	~x
^	Bitwise XOR	Result bit 1 if any of the operand bit is 1 but not both; otherwise results bit 0.	x ^ y
>>	Bitwise right shift	The left operand’s value is moved toward right by the number of bits specified by the right operand.	x >>
<<	Bitwise left shift	The left operand’s value is moved toward left by the number of bits specified by the right operand.	x <<

ASCII

dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char
0	0	000	NULL	32	20	040	space	64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(	72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051	)	73	49	111	I	105	69	151	i
10	a	012	LF	42	2a	052	*	74	4a	112	J	106	6a	152	j
11	b	013	VT	43	2b	053	+	75	4b	113	K	107	6b	153	k
12	c	014	FF	44	2c	054	,	76	4c	114	L	108	6c	154	l
13	d	015	CR	45	2d	055	-	77	4d	115	M	109	6d	155	m
14	e	016	SO	46	2e	056	.	78	4e	116	N	110	6e	156	n
15	f	017	SI	47	2f	057	/	79	4f	117	O	111	6f	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1a	032	SUB	58	3a	072	:	90	5a	132	Z	122	7a	172	z
27	1b	033	ESC	59	3b	073	;	91	5b	133	[	123	7b	173	{
28	1c	034	FS	60	3c	074	<	92	5c	134	\	124	7c	174	|
29	1d	035	GS	61	3d	075	=	93	5d	135	]	125	7d	175	}
30	1e	036	RS	62	3e	076	>	94	5e	136	^	126	7e	176	~
31	1f	037	US	63	3f	077	?	95	5f	137	_	127	7f	177	DEL


```
print(ord('A'))
print(ord('a'))
print(ord('Z'))
print(ord('z'))
print(chr(65))
print(chr(97))
print(chr(90))
print(chr(122))
```