Basic Python Programming

for POSN Computer

Pana Wanitchollakit

2024

Outline

Outline

Basic

Data types

Data types

```
s = "Hello World" # String (can use either '' or "")
i = 42 # Integer
d = 3.14159 # Float
b = True # Boolean
```

Type conversion

```
s = "42"
i = int(s) # i = 42
f = float(s) # f = 42.0
k = str(571) # k = "571"
```

Input

A function named **input** is used for only **string** input.
The **input** function will receive input until you press the <Enter>.

Input

```
s = input() # string input
"""
If you type 42 then enter
s is equal to "42" (string) not 42(int)
"""
```

If you want an integer input, you must convert it to an integer, i.e.,

```
i = int(input()) # make sure that the input is "Integer"
```

Output

A **print** function is used to show output on your commandline.
Note: **print** will add an extra character, namely a newline character "\n" at the end of your output.

Output

```
s = "Hello"
i = 42

print(s) # Hello\n
print(i) # 42\n
print(s, i) # Hello 42\n
# ^ "Hello" and 42 are separated by a space character (" ")
```

String operator

Plus operator (+)

We can concatenate the string using + operator

```
s1 = "Hello"
s2 = "World"

s3 = s1 + " " + s2 # s3 = "Hello World"
```

Multiplication operator (*)

Just like multiplying in arithmetic, we concatenate (plus) a string multiple times instead.

```
s = "Hello" * 3 # s = "HelloHelloHello"
```

Arithmetic Operators

Arithmetic

```
a = 5 + 2 # 7
b = 5 - 2 # 3
c = 5 * 2 # 10
d = 5**2 # 25
e = 5 / 2 # 2.5
f = 5 // 2 # 2 --> floor(5/2) = 2
g = 5 % 2 # 1 --> 5 = 2*2 + 1 <-- remainder = 1
h = -15 % 4 # 1 --> -15 = 4*-4 + 1 <-- remainder = 1
```

Arithmetic Operators Precedence

Order	Operator	Associativity Type
1	()	-
2	**	right to left
3	*, /, //, %	left to right
4	+, -	left to right

Example

```
10 + 3 * 4 + 2 ** 3 * 4 - (4*5-25/5**2) + 5
= (10) + (3 * 4) + (2 ** 3 * 4) - ((4*5) - (25/5**2)) + (5)
= 10 + (3 * 4) + ((2**3)*4) - ((4*5) - (25/(5**2))) + 5
```

$$10 + (3 \times 4) + (2^3 \times 4) - \left((4 \times 5) - \frac{25}{5^2}\right) + 5 = 40$$

Example

```
4 * 5 ** 3 ** 2 * 7
= 4 * 5 ** (3 ** 2) * 7
= 4 * (5 ** (3**2)) * 7
```

$$4 \times 5^{3^2} \times 7 = 4 \times 5^9 \times 7$$

Example

```
5 * 3 // 4 * 10 // 7
```

$$\left\lfloor \frac{\left\lfloor \frac{5\times 3}{4} \right\rfloor \times 10}{7} \right\rfloor$$

Note:

Floor function ▸ Definition : $\lfloor x \rfloor$

Example: $\lfloor -2.5 \rfloor = -3$, $\lfloor 2.5 \rfloor = 2$

String and List

List

In short, it's a box of variables.
String is a special type of list (all elements of a list are characters).

Example

```
l = [1, "A", "B", 10, 0.5]
l2 = list() # Empty list
l3 = [] # Also Empty list
```

List methods (function)

```
l.append(120) # l = [1, "A", "B", 10, 0.5, 120]
l.pop() # [1, "A", "B", 10, 0.5]
```

String & List Operators

Length

The **len** function can be used to count the length of a string.

```
s = "Hello World"
ls = len(s) # ls = 11 (space character (" ") is also counted.)

l = [123, None, 4, "Hello", 3.14159]
ll = len(l) # ll = 5
```

Indexing

You can access a character in string or a element in list with a [ ].
Note: Indexing in Python starts with 0 and can be accessed with a negative index (reversed index).

idx	0	1	2	3	4	5	6	7	8	9	10
str	*H*	*e*	*l*	*l*	*o*		*W*	*o*	*r*	*l*	*d*
idx_r	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1

```
s = "Hello World"
s[0] # H
s[2] # l
s[-1] # d
s[11] # // Error
s[-12] # // Error
```

Control flow

Boolean operators

Relation operators

The relation operators yield *boolean* values, i.e., **True** or **False**.

Python	<	>	<=	>=	!=	==
Math	$<$	$>$	$\leq$	$\geq$	$\neq$	$=$

Logical operators

Apparently, boolean is just a **proposition** in logic.

Python	**and**	**or**	**not**
Logic(Math)	$\wedge$	$\vee$	$\neg$ or $\sim$

De Morgan's Law

- $\neg(p \wedge q) \equiv \neg p \vee \neg q$
- $\neg(p \vee q) \equiv \neg p \wedge \neg q$

Negation of Relation operators

p	$a < b$	$a > b$	$a \leq b$	$a \geq b$	$a \neq b$	$a = b$
$\neg$p	$a \geq b$	$a \leq b$	$a > b$	$a < b$	$a = b$	$a \neq b$

Boolean operator precedence

Note: Every boolean operator has lower precedence than all arithmetic operators.

Order	Operator
1	$==, !=, <=, >=, >, <$
2	**not**
3	**and**
4	**or**

Example

```
p = True
q = False
r = True

not p or q # ~p or q == p -> q
not p or q and r # ~p or (q and r)
```

if-elif-else statement

if else-if else block

- **if** start condition(proposition).
- **elif** another condition if **if** is rejected
- **elif** another condition if the above **elif** is rejected
- **elif** another condition if the above **elif** is rejected
- ⋮
- **else** if all the conditions above are rejected.

The **if-else** blocks can contain many (or none) **elif** block and not be nesscessary to have an **else**.
Note: Each **if-else** block performs action *once* or *none* (no **else**).

Example

```
x = int(input())
if x <= 10:
    print("do if")
elif x <= 25:
    print("do elif 1")
elif x <= 50:
    print("do elif 2")
else:
    print("do else")
```

- if $x \leq 10$: do if
- if $10 < x \leq 25$: do elif 1
- if $25 < x \leq 50$: do elif 2
- if $x > 50$: do else

Many if vs if-else

```
x = int(input())
if x <= 10:
    print("f")
if x <= 25:
    print("ef1")
if x <= 50:
    print("ef2")
if x > 50:
    print("e")
```

- if $x \leq 10$: f ef1 ef2
- if $10 < x \leq 25$: ef1 ef2
- if $25 < x \leq 50$: ef2
- if $x > 50$: e

Loop

For Loop

range ▸ Documentation

range is an iterable object (can be converted into a **list**).

Example

```
# range(stop)
list(range(5)) # [0, 1, 2, 3, 4]
# range(start, stop)
list(range(2, 6)) # [2, 3, 4, 5]
# range(start, stop, step)
list(range(0, 8, 3)) # [0, 3, 6]
list(range(10, -11, -5)) # [10, 5, 0, -5, -10]
```

Note: **stop** exclusive.

for keyword

The **for** keyword is used to iterate an iterable object with the keyword **in**.

Example

```
for i in range(5):
    print(i, end=" ") # 0 1 2 3 4

for n in [3, 10, 20]:
    print(n, end=" ") # 3 10 20

for c in "Hello":
    print(c, end=" ") # H e l l o
```

Note: **end** keyword in **print** change "\n" to the specified string.

While Loop

while keyword

The **while** keyword is like the **if-else** block, but **while** loop does action until the condition is rejected.

Variable assignment

You can assign new value in declared variable.

```
i = 2

i = i + 1 # i=(2)+1 => 3
i += 1 # i=(3)+1 => 4 [equivalent to i=i+1]
i = 2 * i + 1 # i=2*(4)+1 => 9
i *= 3 # i=(9)*3 => 27 [equivalent to i=i*3]
```

Example

```
i = 2
while i < 5:
    print(i, end=" ") # 2 3 4
    i += 1

i = 4
while i >= 2:
    print(i, end=" ") # 4 3 2
    i -= 1

i = 1
while i < 32:
    print(i, end=" ")
    i = 2 * i + 1 # 1 3 7 15 31
```

Nested Loop

Loop can be nested.

Example

```
n = 10
i = 0
while i < n:
    j = 0
    while j < i + 1:
        print("*", end="")
        j += 1
    print()
    i += 1
```

Output

```
*
**
***
****
*****
******
*******
********
*********
```

Example

```
n = 10
i = 0
while i < n:
    j = 0
    while j < n:
        if j < i + 1:
            print("y", end="")
        else:
            print("x", end="")
        j += 1
    print()
    i += 1
```

Output

```
yxxxxxxxxx
yyxxxxxxxx
yyyxxxxxxx
yyyyxxxxxx
yyyyyxxxxx
yyyyyyxxxx
yyyyyyyxxx
yyyyyyyyxx
yyyyyyyyyx
yyyyyyyyyy
```