
Python For Engineers I



Why Python?

- Used in every software industry.
 - From web development to even embedded devices & Raspberry Pi's.
 - Easy-to-pick-up language that only gets as complex as one requires.
 - Writing Python code feels like Pseudocode.
 - Great language to prototype/test out software concepts and ideas.
 - Nicer to use than Java and MATLAB.
 - Complete Python environment can be embedded into a C(++) program.
 - Allows for faster development time.
 - Wider audience reach to make plugins for an application.
-

Little bit of History

- Brainchild of **Guido Van Rossum** in the Netherlands.
- Conceived in the **late 1980s**.
- Created as a successor of the ABC Programming Language.
- **Python version 1.0 came out January 26, 1994**
 - Python is technically older than Java!
 - Java version 1.0 was released January 23, 1996
- Boomed in popularity in 2003+
- *Core Philosophy*:
 - Beautiful is better than ugly.
 - Explicit is better than implicit.
 - Simple is better than complex.
 - Complex is better than complicated.
 - Readability counts.



Our first Python program

```
print('hi')
```

Starting a Python program

- Starting Python programs are extremely simple.
 - Open up a Python IDE on replit.com or onlinegdb.com
 - Python programs are segmented into individual files, containing Python code, called *scripts*.
 - Python scripts always (and should) end with a **.py** file extension.
 - As Python is mostly a scripting language, Python programs work from a starter script which either contains all the code required or kicks off code that's organized in other Python scripts.
 - All code in Python is sequential* and flows from top to bottom as the order of execution/interpretation.
-

Basic Data Types of Python

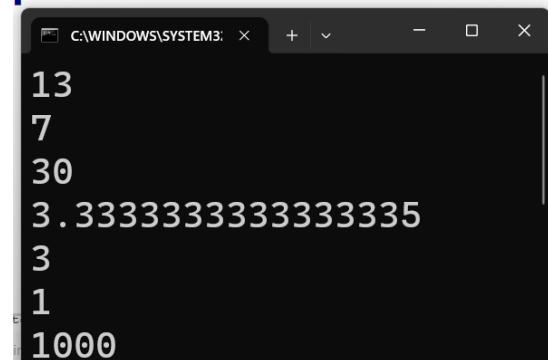
```
# this is a comment
''' single-quote documentation comment '''
""" double-quote documentation comment """
i = 1 # int
b = True # bool
f = 3.33333 # float
s = 'text' # str | string
l = [i, b, f, s] # list
t = (i, b, f, s) # tuple
d = { 'l': l, 't': t } # dict | dictionary
```

Type of Numbers

- Integers:
 - Python's integers has no maximum limit!
 - Be aware, the larger your integer values, the more memory required to represent the value.
 - Floats:
 - Has a limit of 64-bits.
 - Complex:
 - Can be made as a value by using the 'j' suffix like: **1j**.
 - Real and Imaginary parts can be retrieved like an object!:
 - **1j.real**
 - **1j.imag**
 - Can convert other numbers to complex.
-

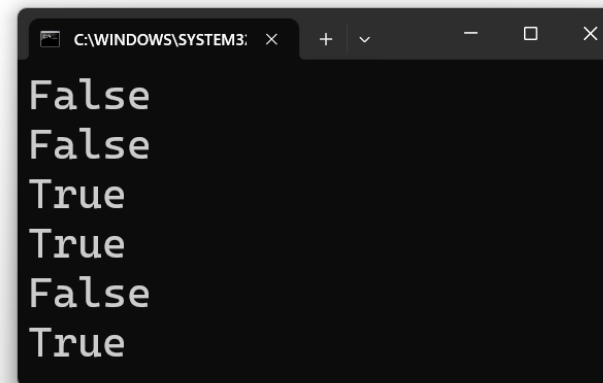
Numbers and Expressions

```
print(10 + 3)
print(10 - 3)
print(10 * 3)
print(10 / 3)
print(10 // 3) # '/' means integer division.
print(10 % 3)  # '%' is modulo.
print(10 ** 3) # '**' means power!
```



```
C:\WINDOWS\SYSTEM32>python
Python 3.10.0 Shell
>print(10 + 3)
13
>print(10 - 3)
7
>print(10 * 3)
30
>print(10 / 3)
3.3333333333333335
>print(10 // 3)
3
>print(10 % 3)
1
>print(10 ** 3)
1000
```

```
print(10 < 3)      # 10 less than 3
print(10 <= 3)     # 10 less than-equal to 3
print(10 > 3)      # 10 greater than 3
print(10 >= 3)     # 10 greater than-equal 3
print(10 == 3)     # 10 equal to 3
print(10 != 3)     # 10 not equal to 3
```



```
C:\WINDOWS\SYSTEM32>python
Python 3.10.0 Shell
>print(10 < 3)
False
>print(10 <= 3)
False
>print(10 > 3)
True
>print(10 >= 3)
True
>print(10 == 3)
False
>print(10 != 3)
True
```

Other Operators

logical operators

a, b = True, False

print(a and b) # False

print(a or b) # True

print(not a) # False

print(not b and (a or b)) # True

identity operators

'is' checks if same object!

x = 'a'

x += 'bc'

y = 'abc'

print(x == y) # True - Same Value

print(x is y) # False - NOT same object!

print(x is not y) # True

bitwise operators

only works for int objects.

AND operator - &

print(1 & 3) # prints 1

OR operator - |

print(1 | 4) # prints 5

XOR operator - ^

print(1 ^ 3) # prints 2

NOT operator - ~

print(~1) # prints -2

Compound Assignment Operators

compound assignment structure.

`<var> <op> = <expression>`

same as doing:

`<var1> = <var1> <op> <expression>`

compound assignment.

`a = 10`

`a += 1` # add - 'a' is now 11

`a -= 4` # sub - 'a' is now 7

`a *= 3` # mul - 'a' is now 21

`a /= 2` # div - 'a' is now 10

`a %= 4` # mod - 'a' is now 2

`a **= 5` # pow - 'a' is now 32

`a |= 1` # OR - 'a' is now 33

`a &= 1` # AND - 'a' is now 1

`a ^= 1` # XOR - 'a' is now 0

Strings - Basics

- Can be created using single or double quotes!
 - This allows using the other quote when using a specific one to indicate the string.
- Strings can also be indexed like an array.

```
name = input('tell me your name: ')\nprint(f"hi, {name}, pleasure to meet you.")
```

```
my_name = 'peter'\nprint(my_name[3]) # gives 'i'.
```

```
# you can add strings together!\nmy_name += ' griffin'\nprint(my_name)
```

```
# multiplying with a number repeats the string.\nmultiplied_name = my_name * 4\nprint(multiplied_name)
```

Strings – Formatting with F-Strings

- Python provides many ways to format data and strings.
- For this workshop, we're going to use f-strings.
- F-strings provide a fast and convenient way to take multiple, existing data and put it all together in a single string for whatever purpose.

```
# structure
# put expression between '{' and '}'
f'characters {<expression>}'
f"characters {<expression>}"

# f-strings allow you to inline format
# variables into a string without having to
# convert them or use weird specifiers.
a, b, c = 1, 1.0, '1!'
print(f"a = {a}, b = {b}, c = {c}")
pwr = 3
s = f' {(a+a)}^{pwr} = {(a+a)**pwr}'
print(s)
```

Strings – Slicing

- Sometimes you only want to access a portion of a string.
- How do we do that without doing a lot of string operations?
- Answer: Slicing.
- Basic idea: gives you only a portion of a table-based object that you need.
- Slicing can be done with other kinds of objects we'll see later.

items start through stop-1

a[start:stop]

items start through the rest of the array

a[start:]

items from the beginning through stop-1

a[:stop]

```
name = 'peter griffin'
```

```
print(name[1:])      # eter griffin
```

```
print(name[:7])      # peter g
```

```
print(name[0:10:2])  # ptrgi
```

a copy of the whole array

a[:]

a[start:stop:step]

Control Flow – If Statements

```
if <expression> :  
    → <statements>
```

```
if <expression> :  
    → <statements>  
else:  
    → <statements>
```

```
if <expression> :  
    → <statements>  
elif <expression> :  
    → <statements>  
else:  
    → <statements>
```

```
a, b = 100, 10 * 5  
if a < b:  
    → print('a is less than b')  
elif a > b:  
    → print('a is greater than b')  
else:  
    → print('a is equal to b')  
  
a = 100  
if (a & 1) == 0:  
    → print('a is odd')  
else:  
    → print('a is even')
```

Control Flow – While Loops

```
while <expression> :  
    →<statements>
```

```
while <expression> :  
    →<statements>
```

```
else:  
    →<statements>
```

```
a = int(input('enter a: '))  
b = int(input('enter b: '))  
while a < b:  
    →print(b - a)  
    →a += 1  
else:  
    →print("loop is done.")
```

- Best to use for data that has an unknown end.
- Example is called 3n+1.
- The 'else' part runs after the loop finishes.

```
n = int(input('enter a number: '))  
while n != 1:  
    →print(n)  
    →if n % 2 == 0:  
        →→n //= 2  
    →else:  
        →→n = 3 * n + 1
```

Control Flow – For Loops

```
for <var> in <expression> :  
    → <statements>
```

when loop ends, the 'else' part executes!

```
for <var> in <expression> :  
    → <statements>  
else:  
    → <statements>
```

loop 'a' from 0 to 99 or [0, 100)

```
for a in range(100):  
    → print(a)
```

loop 'b' starting at 80 | [80, 100)

```
for b in range(80, 100):  
    → print(b)
```

loop 'c' starting at 72 | [72, 100)

increment 'c' by 3 instead of 1

```
for c in range(72, 100, 3):  
    → print(c)
```

Control Flow – Loop Controls

'continue' – skips current iteration of loop

'break' – stops the loop entirely.

they can be used in ANY type of loop.

```
for i in range(30):
```

```
    > # skip printing multiples of 5
```

```
    > # and move onto the next iteration.
```

```
    > if i % 5 == 0:
```

```
        > continue
```

```
    > print(i)
```

- **break** and **continue**.

- Can ONLY be used in a loop, any kind of loop.

```
# break stops the 'else' part from executing.
```

```
n, factorial = 5, 1
```

```
for i in range(1, n + 1):
```

```
    > factorial *= i
```

```
    > if factorial > 50:
```

```
        > print("Factorial exceeds 50!")
```

```
        > break
```

```
else:
```

```
    > print("Factorial:", factorial)
```

Functions I

```
def <name here> (<parameters here>) :  
    → <statements here>
```

```
def add_mul1(a, b, c):  
    → return a + b * c
```

same function with OPTIONAL type annotations.

```
def add_mul2(a: int, b: int, c: int) -> int:  
    → return a + b * c
```

```
def no_params1():  
    → return 1.0
```

```
# this will return 2 + 4 * 6  
print( add_mul2(2, 4, 6) )
```

```
def no_params2():  
    → pass      # function has no code!
```

Functions II

Python Functions can

return multiple objects!

```
def cube_and_root(num):
```

```
    → if num < 0.0:
```

```
        → → return 0.0, 0.0, False
```

```
    → return num**3, num**(1/3), True
```

ask for a numerical input

```
str_entry = input('enter a positive number: ')
```

convert the input to a decimal point value.

```
num_entry = float(str_entry)
```

```
print(cube_and_root(num_entry))
```

Word Problems Set A

```
a = int(input('enter an int: '))
```

1. Square the value of **a** then print **a**.
 2. if **a** is over 50, add 5 more to it then print **a**.
 3. If **a** is negative, multiply **a** with -1.
 4. While **a** is less than 10, print **a** then increase **a** by 2.
 5. While **a** is less than 20 and **a** is an even number, print **a** and then increase **a** by 4.
-

Word Problems Set B

```
a = int(input('enter an int #1: '))  
b = int(input('enter an int #2: '))
```

1. Print the sum of **a** and **b**.
 2. Print the difference of **a** and **b**.
 3. Print if **a** is greater than **b**.
 4. If **a**, multiplied by 2, is greater than **b**, print that **a** is bigger. Else, print **b** is greater.
 5. Swap the values of **a** and **b**.
 6. Print the multiplication of **a** and **b** if both **a** and **b** are even, print the division of **a** and **b** if they're both odd, print the addition of **a** and **b** if **a** is even but **b** is odd, print the subtraction of **a** and **b** if **a** is odd but **b** is even.
-

Abstract Word Problem #1

- Using ``input()``, try to create a small program that doesn't stop running (basic interactive program) unless a user specifically wants to quit.
 - Don't forget you can give ``input`` a prompt message as a function argument!
 - You don't necessarily need a variable but if it helps, use one.
 - Don't focus on efficiency or think there's only one way to accomplish this.
-

Abstract Word Problem #2

- Make a program that asks the user for 3 coefficients of the values in the quadratic formula:
 $Ax^2 + Bx + C$
- Best to make this as a function so we can return two x values.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$