MY JOURNAL TO PYTHON

4110E233 - 高瑞夫

AGENDA

1. LEARNING PYTHON
2. INPUT AND OUTPUT
3. DATA TYPES
4. OPERATORS
5. CONTROLS
6. LOOP
7. FUNCTION

LEARNING PYTHON

CONTENT

CONTENT

LEARNING PYTHON

Introduction

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Good to know

- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
- In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Python Jobs

Python is very high in demand and all the major companies are looking for great Python Programmers to develop websites, software components, and applications or to work with Data Science, AI, and ML technologies.

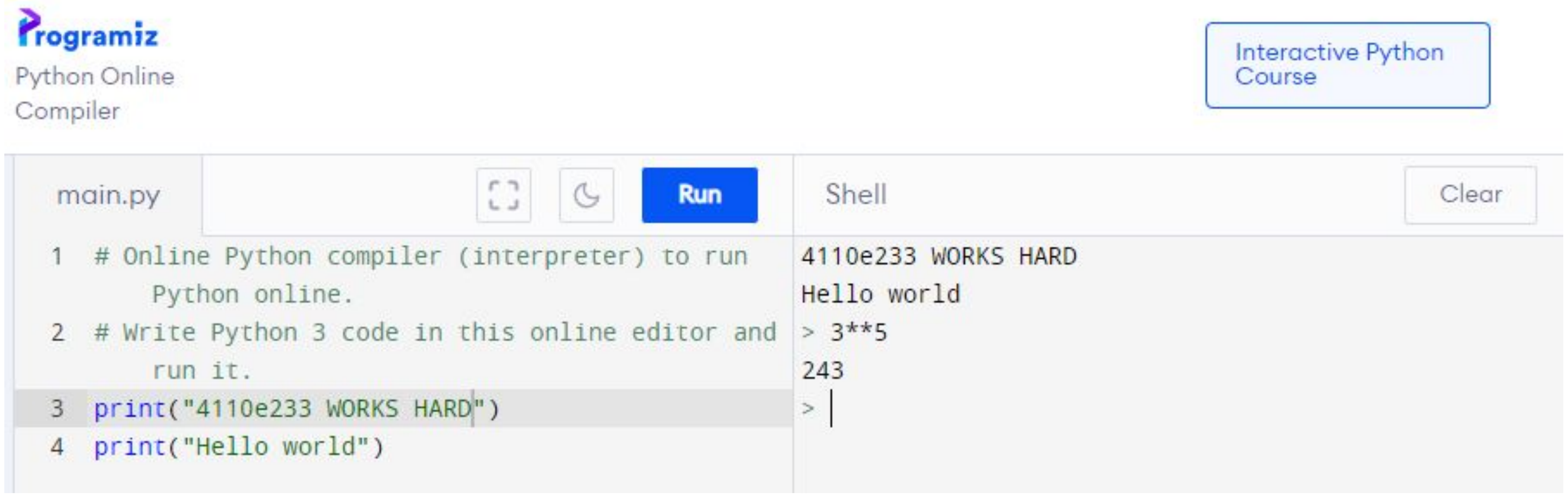
Today a Python Programmer with 3-5 years of experience is asking for around $150,000 annual package and this is the most demanding programming language in America. Though it can vary depending on the location of the Job. It's impossible to list all of the companies using Python, to name a few big companies are Google, Intel, NASA, PayPal, IBM, and many more...

Why to Learn Python?

Python is consistently rated as one of the world's most popular programming languages. Python is fairly easy to learn, so if you are starting to learn any programming language then Python could be your great choice. Today various Schools, Colleges and Universities are teaching Python as their primary programming language. There are many other good reasons which makes Python as the top choice of any programmer:

- Python is Open Source which means its available free of cost.
- Python is simple and so easy to learn
- Python is versatile and can be used to create many different things.
- Python has powerful development libraries include AI, ML etc.
- Python is much in demand and ensures high salary

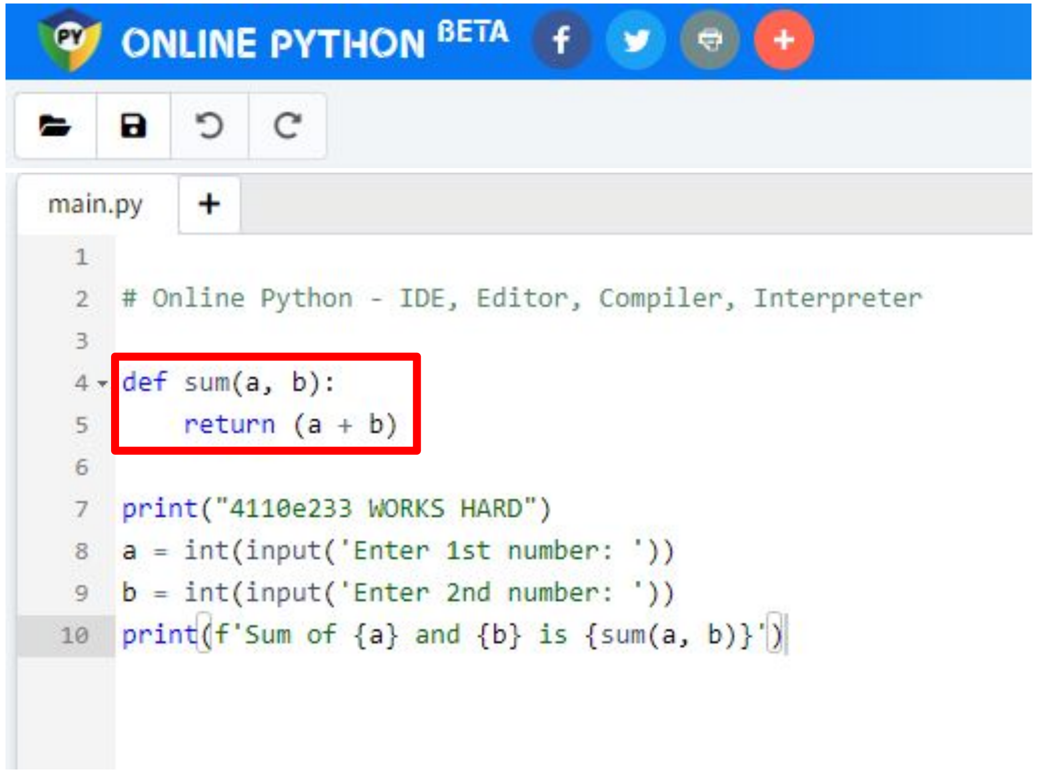
Python Online Interpreter: One



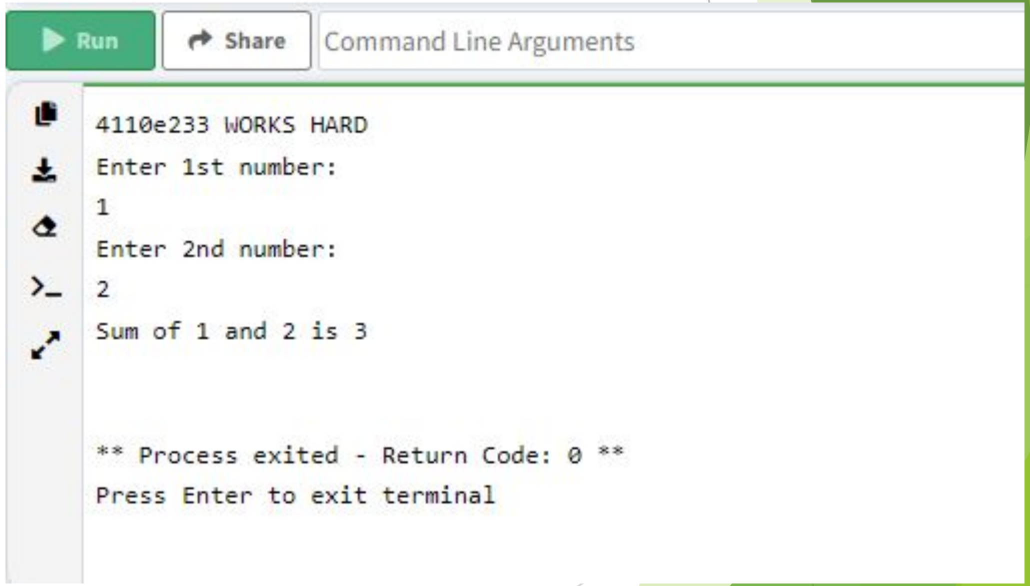
```
# Online Python compiler (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it.
print("4110e233 WORKS HARD")
print("Hello world")
```

```
4110e233 WORKS HARD
Hello world
> 3**5
243
>
```

Python Online Interpreter: Two



```python

# Online Python - IDE, Editor, Compiler, Interpreter

def sum(a, b):
    return (a + b)

print("4110e233 WORKS HARD")
a = int(input('Enter 1st number: '))
b = int(input('Enter 2nd number: '))
print(f'Sum of {a} and {b} is {sum(a, b)}')
```



```
4110e233 WORKS HARD
Enter 1st number:
1
Enter 2nd number:
2
Sum of 1 and 2 is 3


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

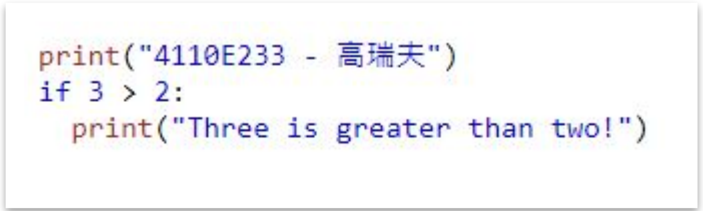
LEARNING PYTHON

Syntax

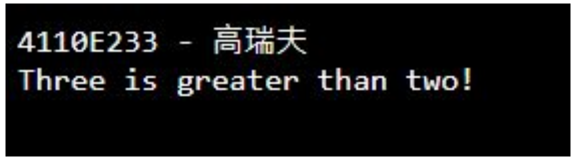
Python Indentation

- Indentation refers to the spaces at the beginning of a code line.
- Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.
- Python uses indentation to indicate a block of code.

Correct Indentation

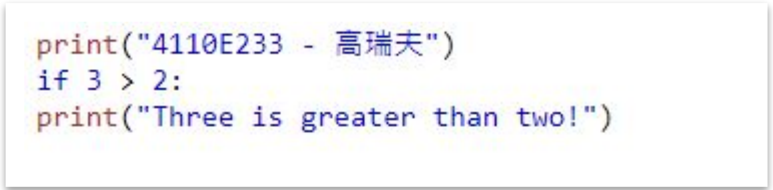


```
print("4110E233 - 高瑞夫")
if 3 > 2:
  print("Three is greater than two!")
```

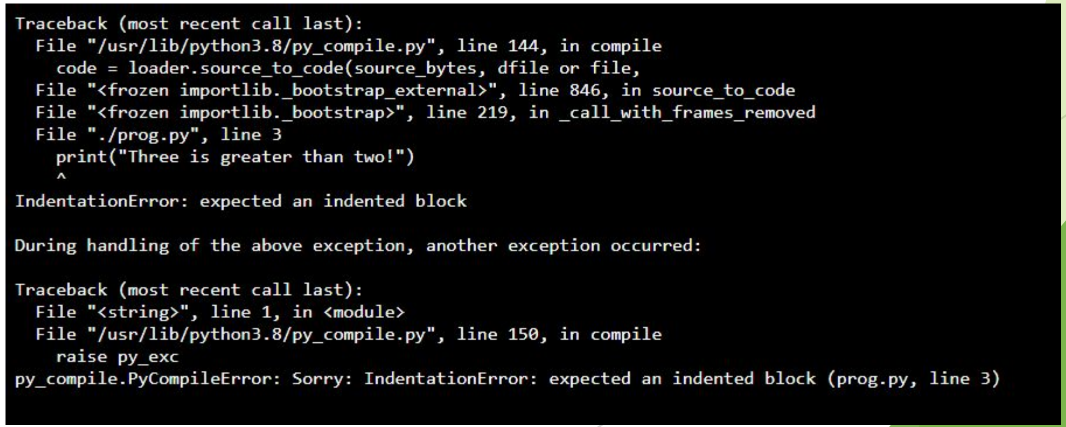


```
4110E233 - 高瑞夫
Three is greater than two!
```

Incorrect Indentation



```
print("4110E233 - 高瑞夫")
if 3 > 2:
print("Three is greater than two!")
```



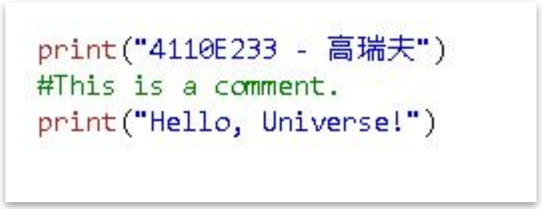
```
Traceback (most recent call last):
  File "/usr/lib/python3.8/py_compile.py", line 144, in compile
    code = loader.source_to_code(source_bytes, dfile or file,
  File "<frozen importlib._bootstrap_external>", line 846, in source_to_code
  File "<frozen importlib._bootstrap>", line 219, in _call_with_frames_removed
  File "./prog.py", line 3
    print("Three is greater than two!")
    ^
IndentationError: expected an indented block

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib/python3.8/py_compile.py", line 150, in compile
    raise py_exc
py_compile.PyCompileError: Sorry: IndentationError: expected an indented block (prog.py, line 3)
```

LEARNING PYTHON

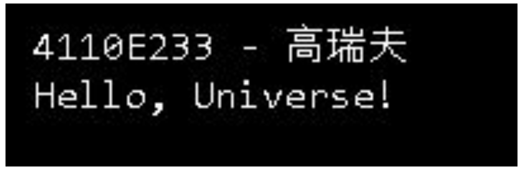
Comments

Creating a Comment

- Comments starts with a #, and Python will ignore them

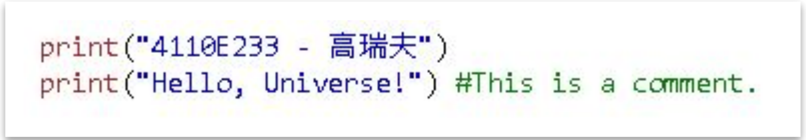


```
print("4110E233 - 高瑞夫")
#This is a comment.
print("Hello, Universe!")
```

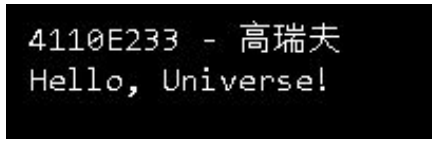


```
4110E233 - 高瑞夫
Hello, Universe!
```

- Comments can be placed at the end of a line, and Python will ignore the rest of the line:

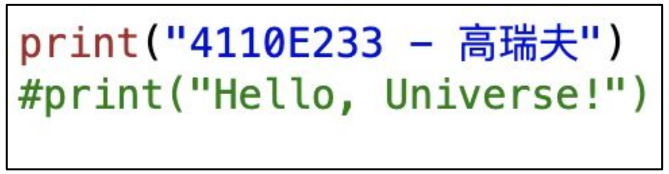


```
print("4110E233 - 高瑞夫")
print("Hello, Universe!") #This is a comment.
```

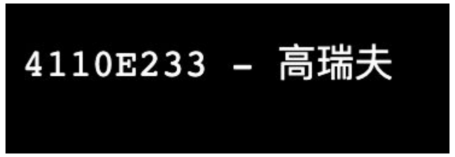


```
4110E233 - 高瑞夫
Hello, Universe!
```

- A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code:



```
print("4110E233 - 高瑞夫")
#print("Hello, Universe!")
```



```
4110E233 - 高瑞夫
```

LEARNING PYTHON

Variables

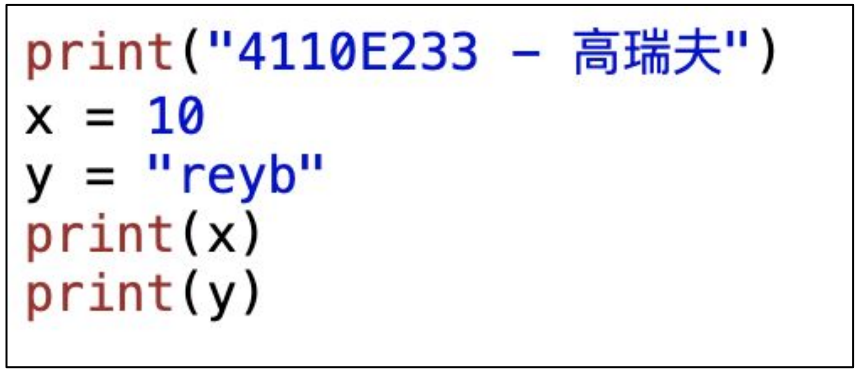
Variables

Variables are containers for storing data values.

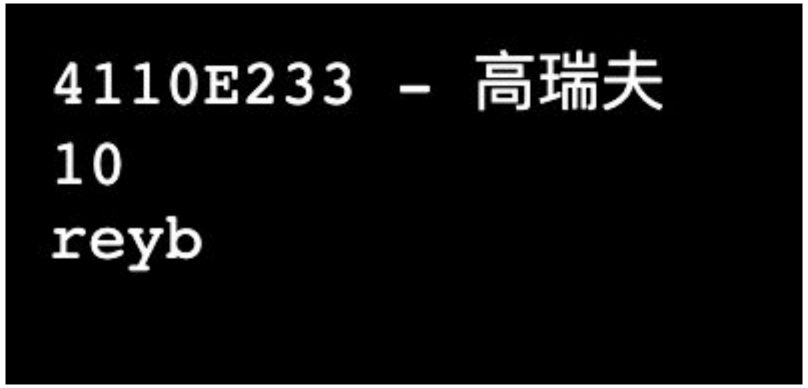
● Creating Variables

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.



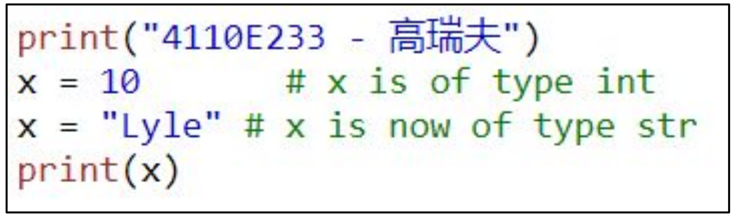
```
print("4110E233 - 高瑞夫")
x = 10
y = "reyb"
print(x)
print(y)
```



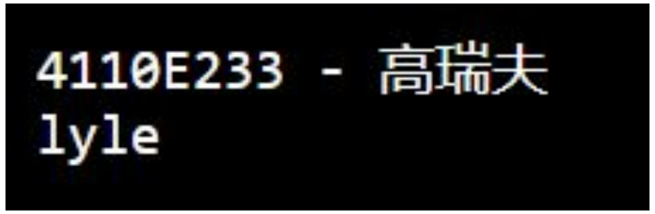
```
4110E233 - 高瑞夫
10
reyb
```

● Creating Variables

Variables do not need to be declared with any particular *type*, and can even change type after they have been set.



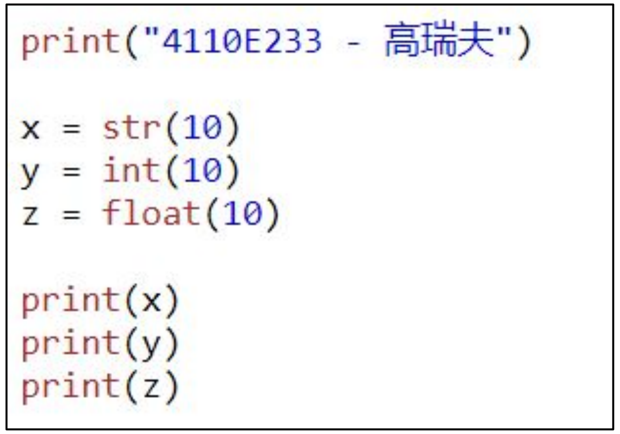
```
print("4110E233 - 高瑞夫")
x = 10       # x is of type int
x = "Lyle" # x is now of type str
print(x)
```



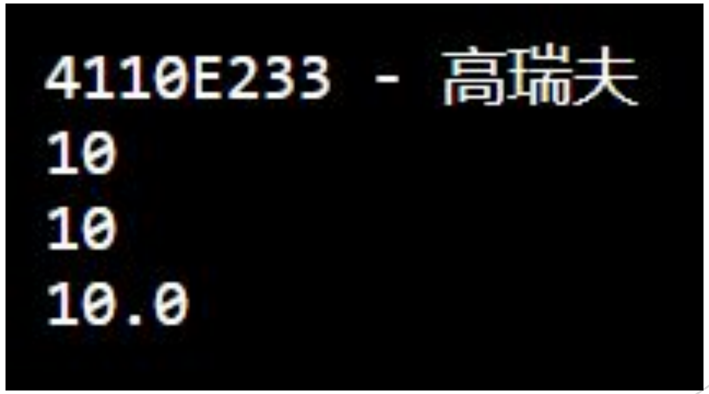
```
4110E233 - 高瑞夫
lyle
```

● Casting

If you want to specify the data type of a variable, this can be done with casting.



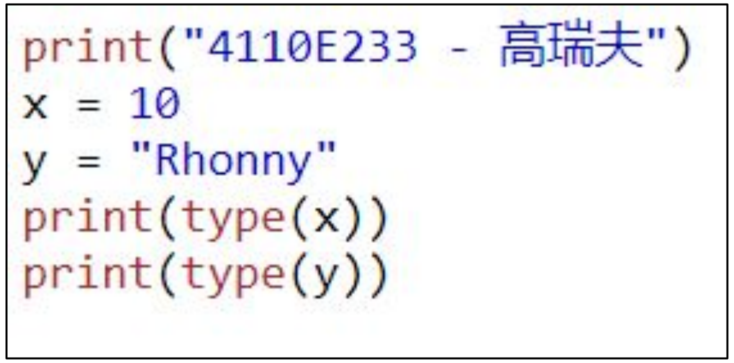
```
print("4110E233 - 高瑞夫")

x = str(10)
y = int(10)
z = float(10)

print(x)
print(y)
print(z)
```



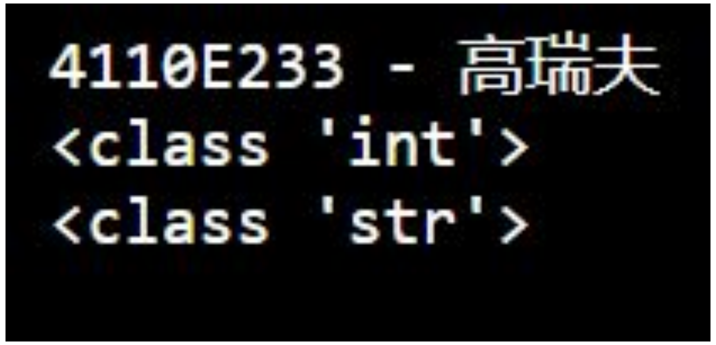
```
4110E233 - 高瑞夫
10
10
10.0
```

● Get the Type

You can get the data type of a variable with the type() function.



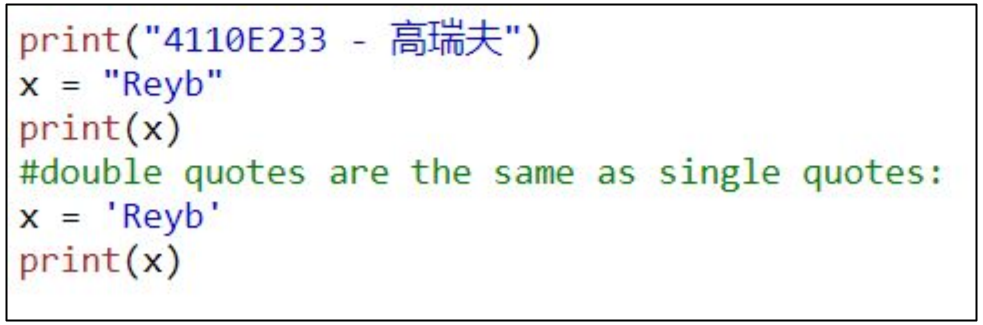
```
print("4110E233 - 高瑞夫")
x = 10
y = "Rhonny"
print(type(x))
print(type(y))
```



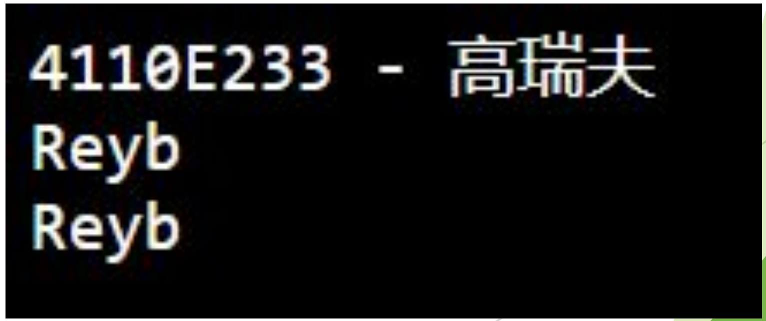
```
4110E233 - 高瑞夫
<class 'int'>
<class 'str'>
```

● Single or Double Quotes?

String variables can be declared either by using single or double quotes:



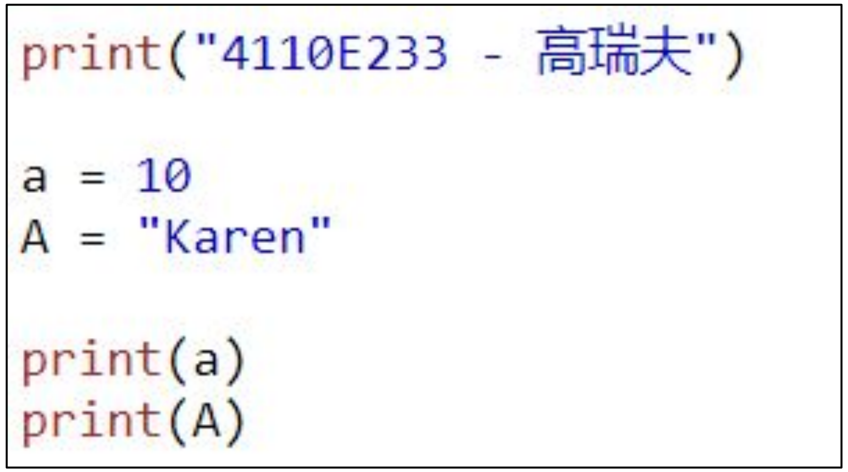
```
print("4110E233 - 高瑞夫")
x = "Reyb"
print(x)
#double quotes are the same as single quotes:
x = 'Reyb'
print(x)
```



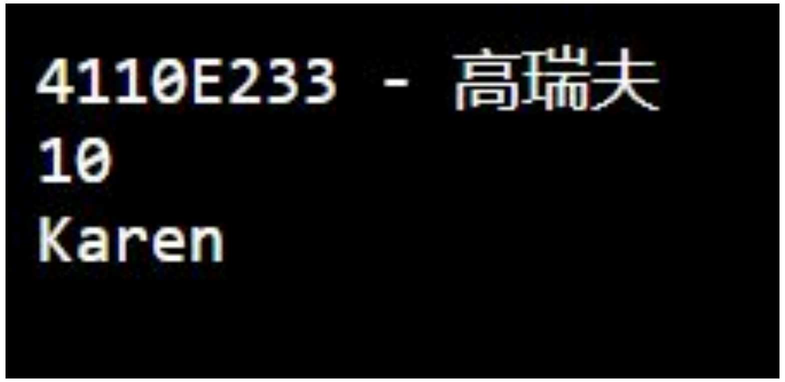
```
4110E233 - 高瑞夫
Reyb
Reyb
```

● Case-Sensitive

Variable names are case-sensitive.



```
print("4110E233 - 高瑞夫")

a = 10
A = "Karen"

print(a)
print(A)
```



```
4110E233 - 高瑞夫
10
Karen
```

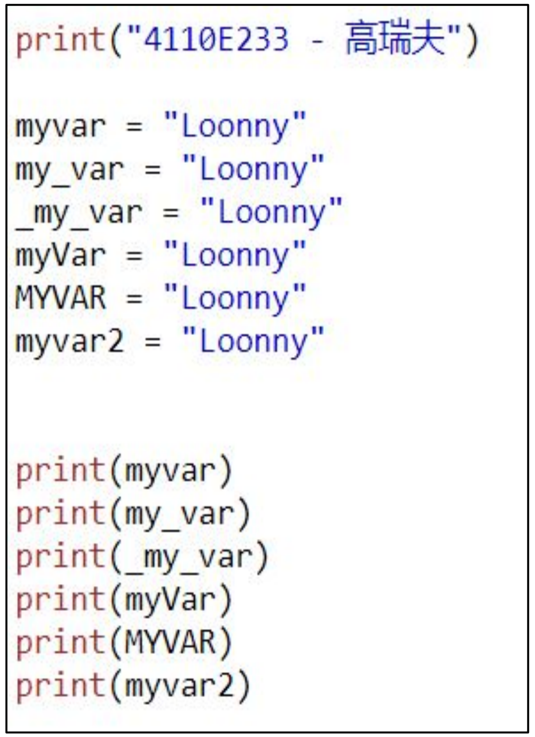
Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

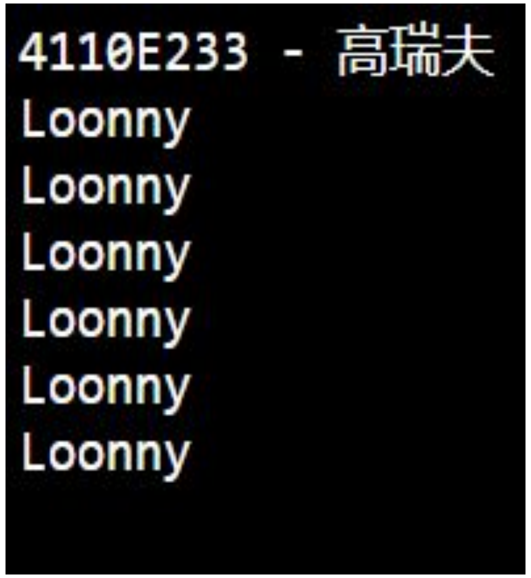
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Remember that variable names are case-sensitive

Legal variable names

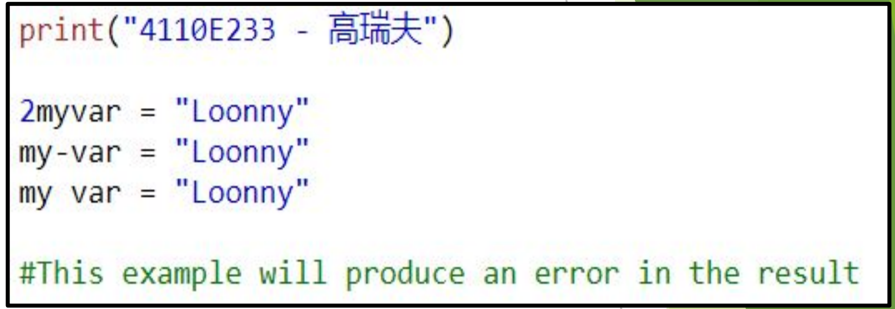


```python
print("4110E233 - 高瑞夫")

myvar = "Loonny"
my_var = "Loonny"
_my_var = "Loonny"
myVar = "Loonny"
MYVAR = "Loonny"
myvar2 = "Loonny"


print(myvar)
print(my_var)
print(_my_var)
print(myVar)
print(MYVAR)
print(myvar2)
```

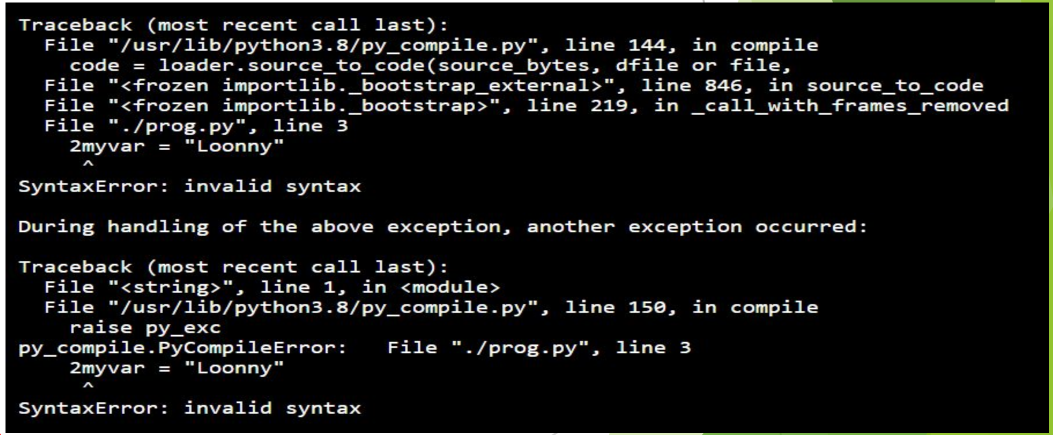


```
4110E233 - 高瑞夫
Loonny
Loonny
Loonny
Loonny
Loonny
Loonny
```

Illegal variable names



```python
print("4110E233 - 高瑞夫")

2myvar = "Loonny"
my-var = "Loonny"
my var = "Loonny"

#This example will produce an error in the result
```



```
Traceback (most recent call last):
  File "/usr/lib/python3.8/py_compile.py", line 144, in compile
    code = loader.source_to_code(source_bytes, dfile or file,
  File "<frozen importlib._bootstrap_external>", line 846, in source_to_code
  File "<frozen importlib._bootstrap>", line 219, in _call_with_frames_removed
  File "./prog.py", line 3
    2myvar = "Loonny"
     ^
SyntaxError: invalid syntax

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib/python3.8/py_compile.py", line 150, in compile
    raise py_exc
py_compile.PyCompileError:   File "./prog.py", line 3
    2myvar = "Loonny"
     ^
SyntaxError: invalid syntax
```

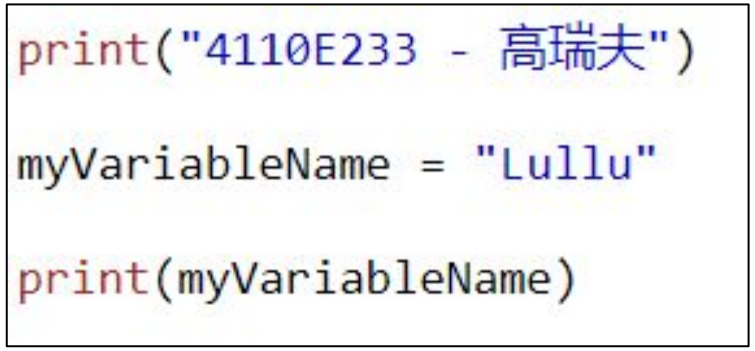
Multi Words Variable Names

Variable names with more than one word can be difficult to read.

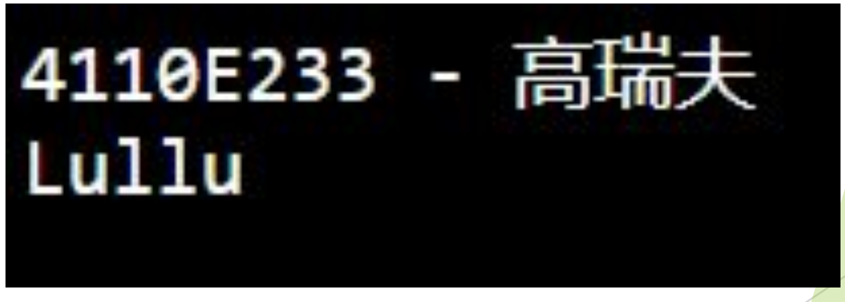
There are several techniques you can use to make them more readable:

- Camel Case

Each word, except the first, starts with a capital letter:



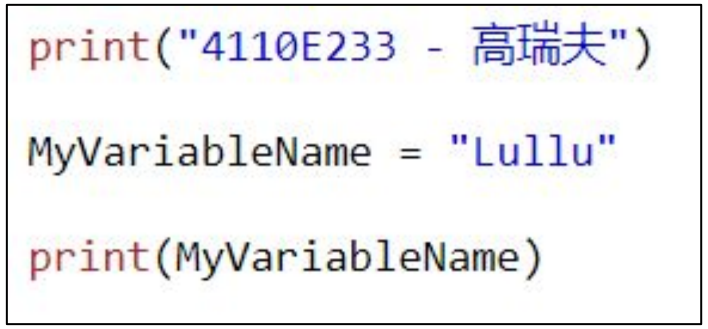
```
print("4110E233 - 高瑞夫")

myVariableName = "Lullu"

print(myVariableName)
```



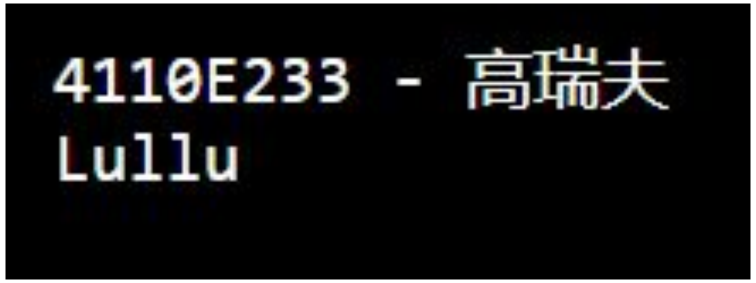
```
4110E233 - 高瑞夫
Lullu
```

Pascal Case

Each word starts with a capital letter:



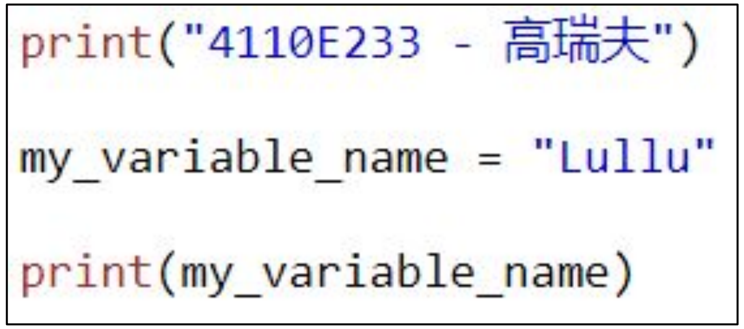
```
print("4110E233 - 高瑞夫")

MyVariableName = "Lullu"

print(MyVariableName)
```



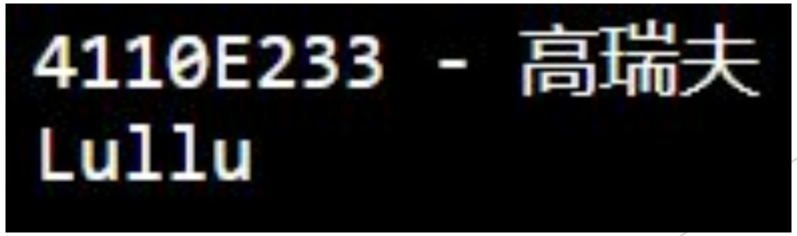
```
4110E233 - 高瑞夫
Lullu
```

Snake Case

Each word is separated by an underscore character:



```
print("4110E233 - 高瑞夫")

my_variable_name = "Lullu"

print(my_variable_name)
```

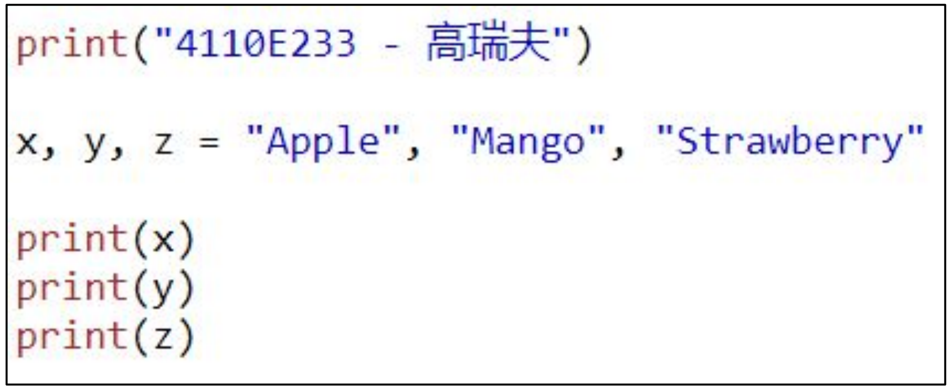


```
4110E233 - 高瑞夫
Lullu
```

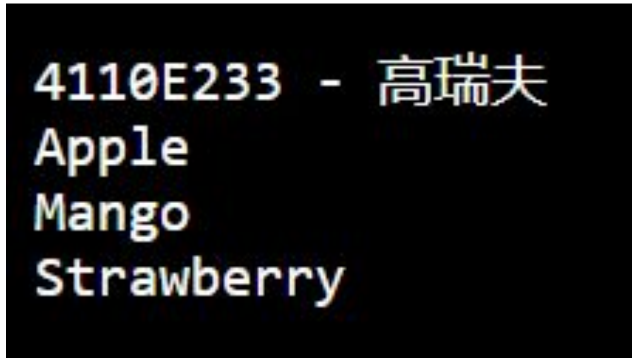
Assign Multiple Values

- Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:



```
print("4110E233 - 高瑞夫")

x, y, z = "Apple", "Mango", "Strawberry"

print(x)
print(y)
print(z)
```

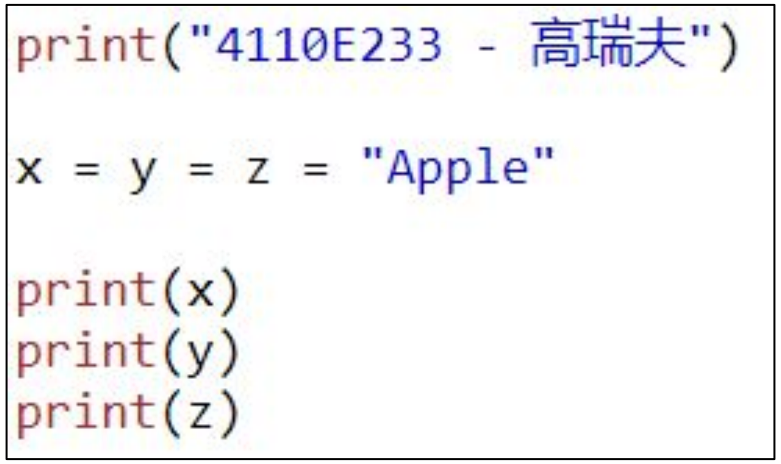


```
4110E233 - 高瑞夫
Apple
Mango
Strawberry
```

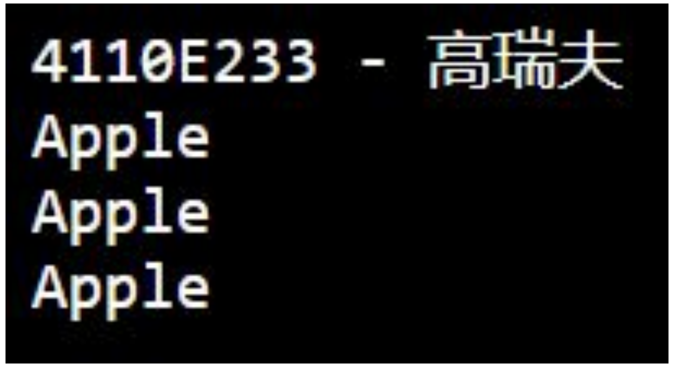
Note: Make sure the number of variables matches the number of values, or else you will get an error.

● One Value to Multiple Variables

And you can assign the *same* value to multiple variables in one line:



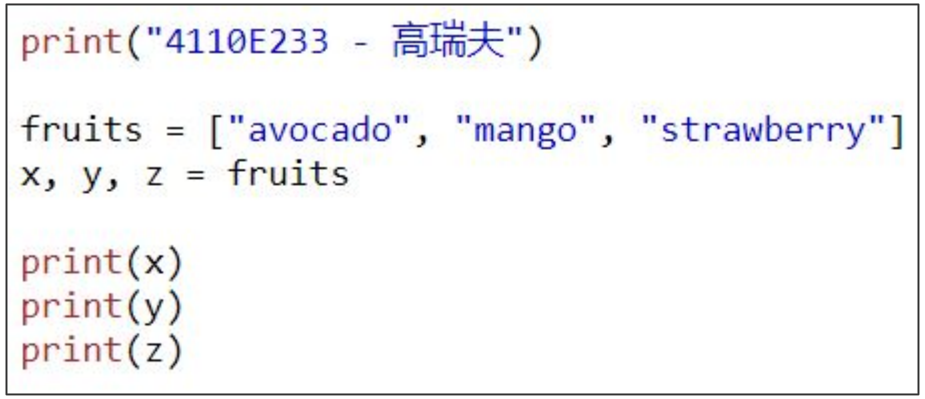
```
print("4110E233 - 高瑞夫")

x = y = z = "Apple"

print(x)
print(y)
print(z)
```



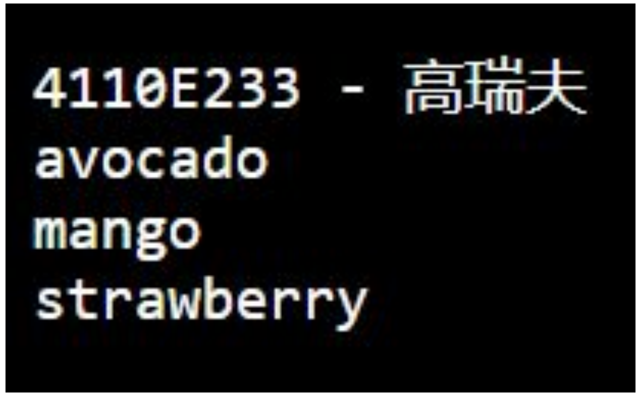
```
4110E233 - 高瑞夫
Apple
Apple
Apple
```

● Unpack a Collection

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.



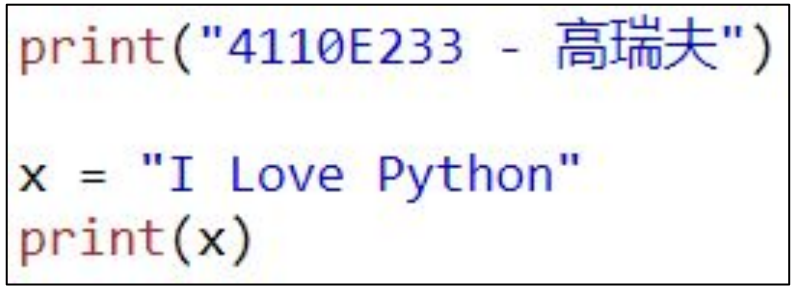
```python
print("4110E233 - 高瑞夫")

fruits = ["avocado", "mango", "strawberry"]
x, y, z = fruits

print(x)
print(y)
print(z)
```



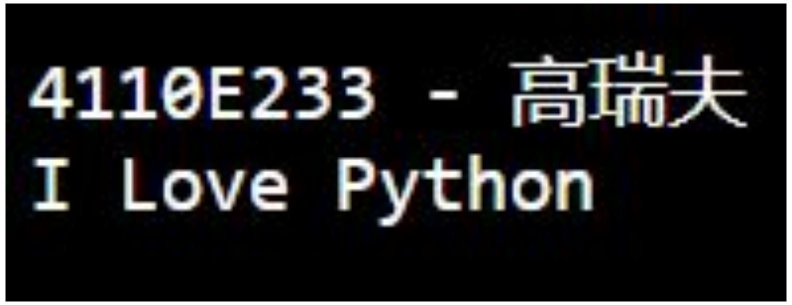
```
4110E233 - 高瑞夫
avocado
mango
strawberry
```

Output Variables

The Python print() function is often used to output variables.

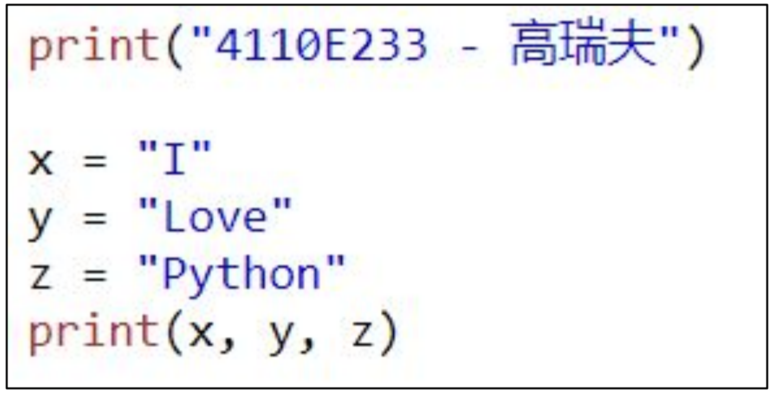


```
print("4110E233 - 高瑞夫")

x = "I Love Python"
print(x)
```

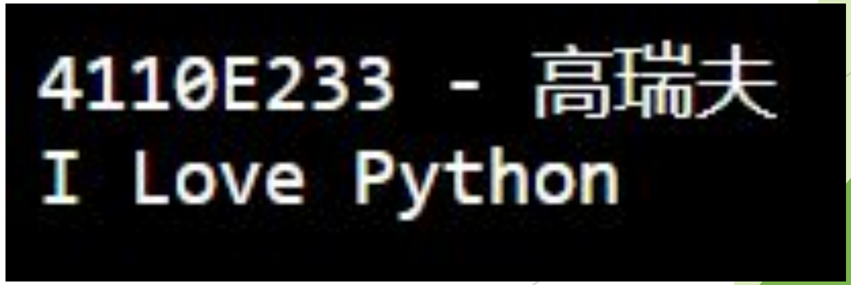


```
4110E233 - 高瑞夫
I Love Python
```

In the print() function, you output multiple variables, separated by a comma:

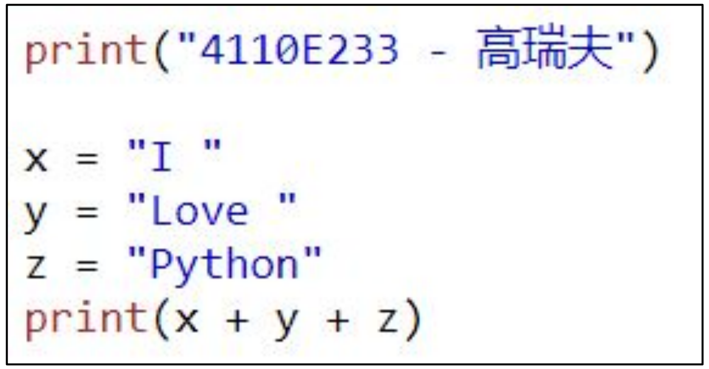


```
print("4110E233 - 高瑞夫")

x = "I"
y = "Love"
z = "Python"
print(x, y, z)
```

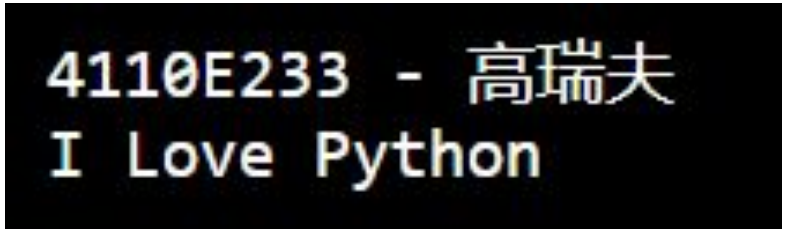


```
4110E233 - 高瑞夫
I Love Python
```

You can also use the + operator to output multiple variables:



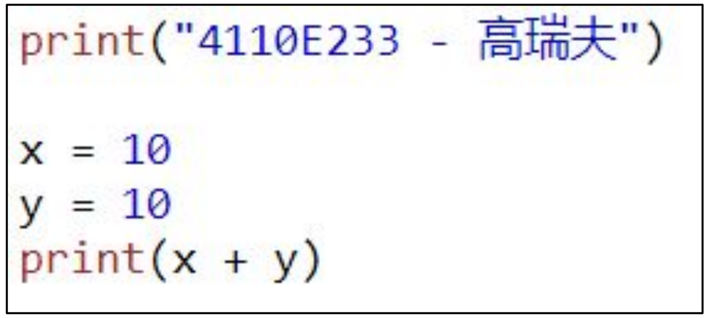
```
print("4110E233 - 高瑞夫")

x = "I "
y = "Love "
z = "Python"
print(x + y + z)
```



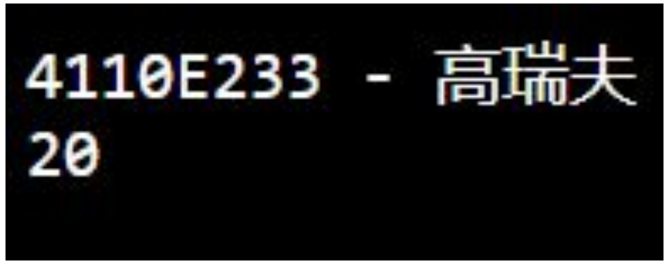
```
4110E233 - 高瑞夫
I Love Python
```

Notice the space character after "I " and "Love ", without them the result would be "ILovePython".

For numbers, the + character works as a mathematical operator:

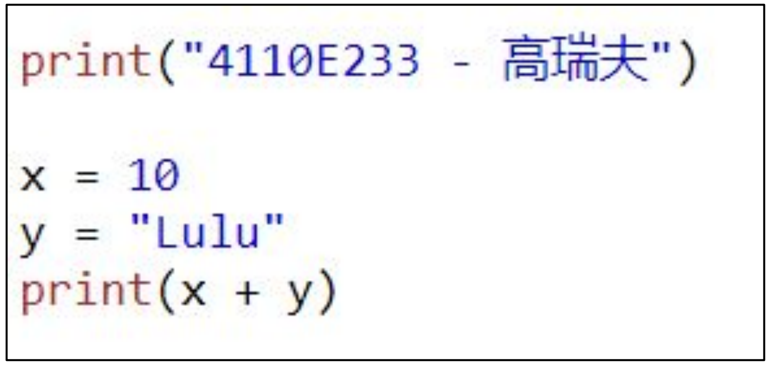


```
print("4110E233 - 高瑞夫")

x = 10
y = 10
print(x + y)
```

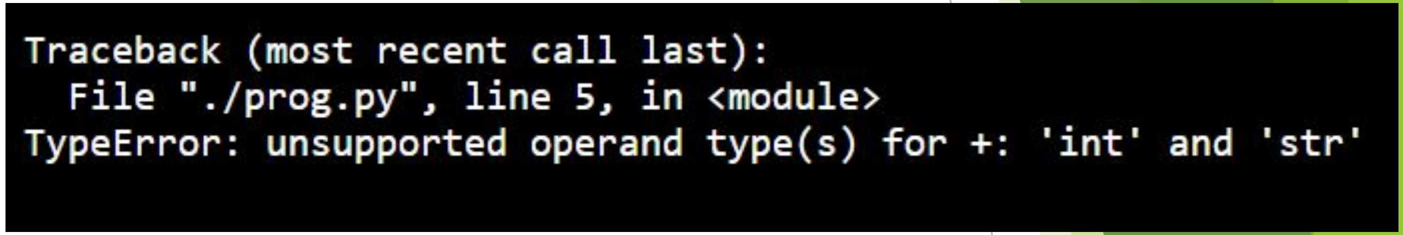


```
4110E233 - 高瑞夫
20
```

In the print() function, when you try to combine a string and a number with the + operator, Python will give you an error:

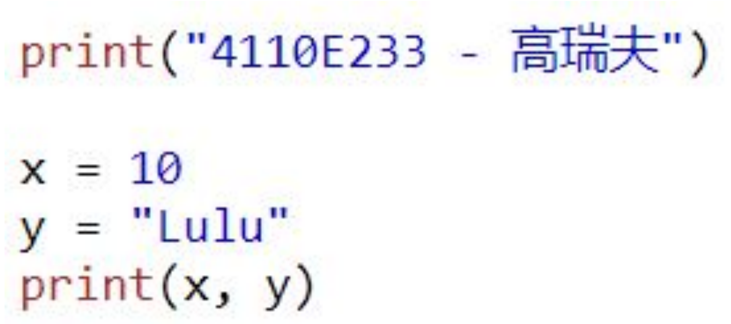


```
print("4110E233 - 高瑞夫")

x = 10
y = "Lulu"
print(x + y)
```

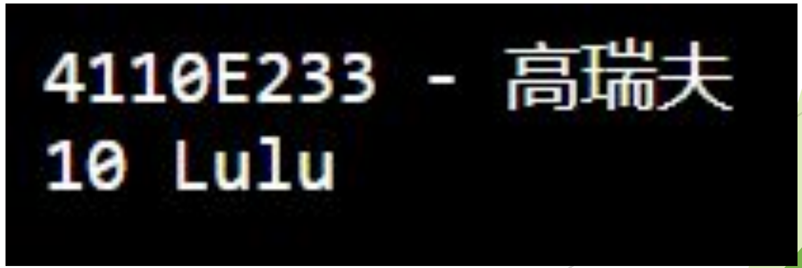


```
Traceback (most recent call last):
  File "./prog.py", line 5, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:



```
print("4110E233 - 高瑞夫")

x = 10
y = "Lulu"
print(x, y)
```

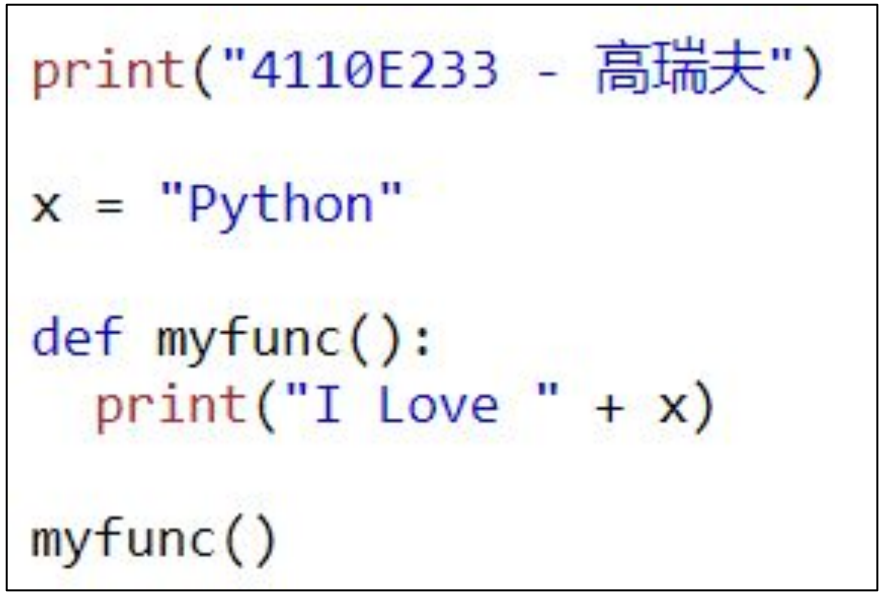


```
4110E233 - 高瑞夫
10 Lulu
```

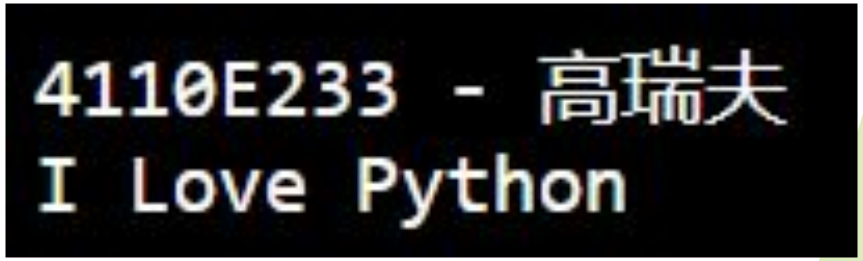
Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

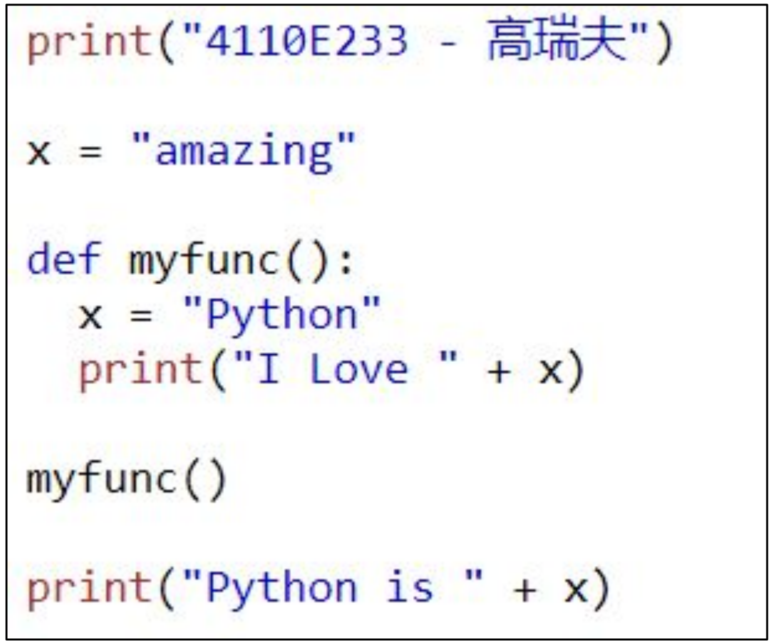


```
print("4110E233 - 高瑞夫")

x = "Python"

def myfunc():
  print("I Love " + x)

myfunc()
```

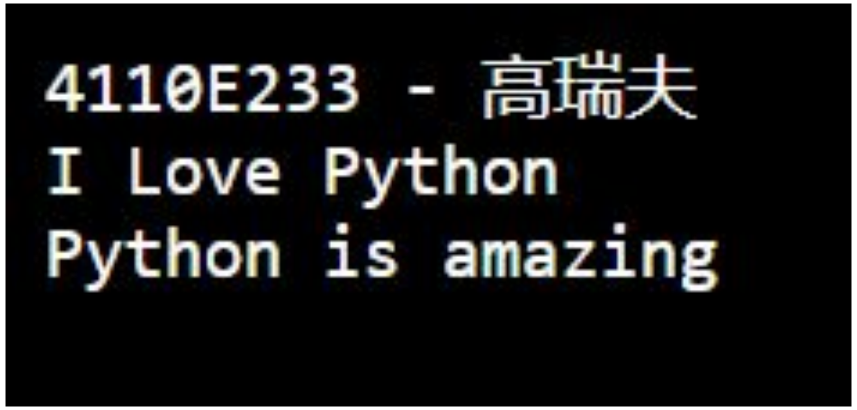


```
4110E233 - 高瑞夫
I Love Python
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.



```
print("4110E233 - 高瑞夫")

x = "amazing"

def myfunc():
  x = "Python"
  print("I Love " + x)

myfunc()

print("Python is " + x)
```

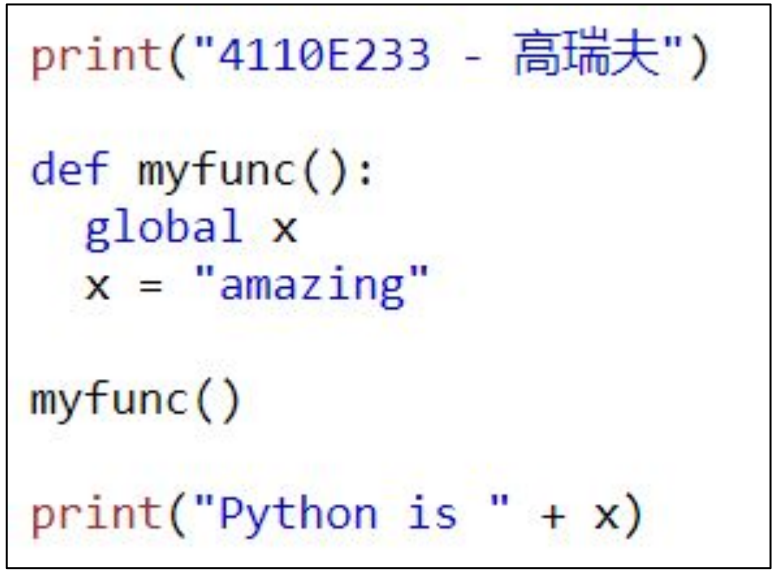


```
4110E233 - 高瑞夫
I Love Python
Python is amazing
```

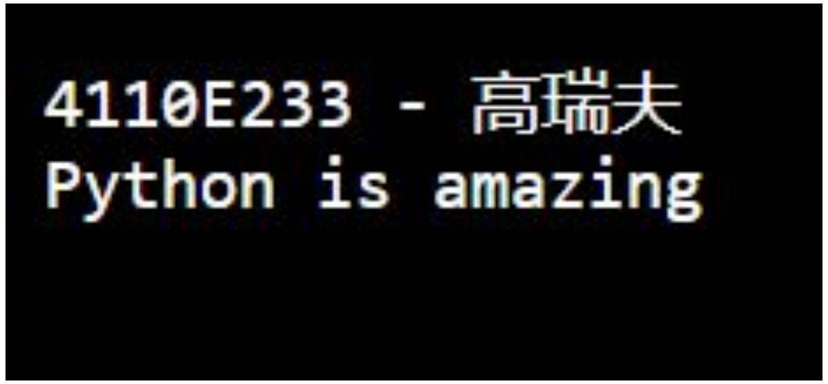
● The global Keyword

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the global keyword.

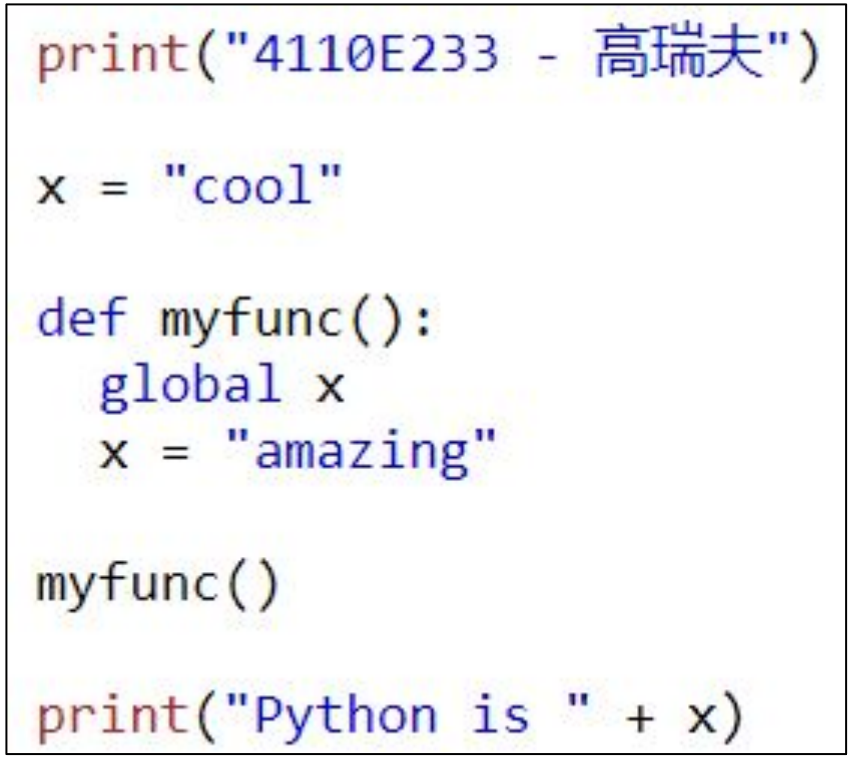


```
print("4110E233 - 高瑞夫")

def myfunc():
  global x
  x = "amazing"

myfunc()

print("Python is " + x)
```

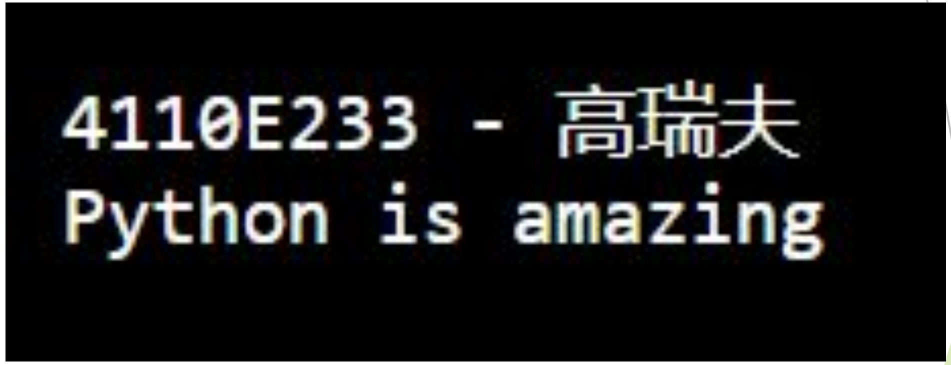


```
4110E233 - 高瑞夫
Python is amazing
```

Also, use the global keyword if you want to change a global variable inside a function.



```
print("4110E233 - 高瑞夫")

x = "cool"

def myfunc():
  global x
  x = "amazing"

myfunc()

print("Python is " + x)
```



```
4110E233 - 高瑞夫
Python is amazing
```

PYTHON DATA TYPES

CONTENT

PYTHON DATA TYPES

Data Types

Data Types

- Built-in Data Types

In programming, data type is an important concept.

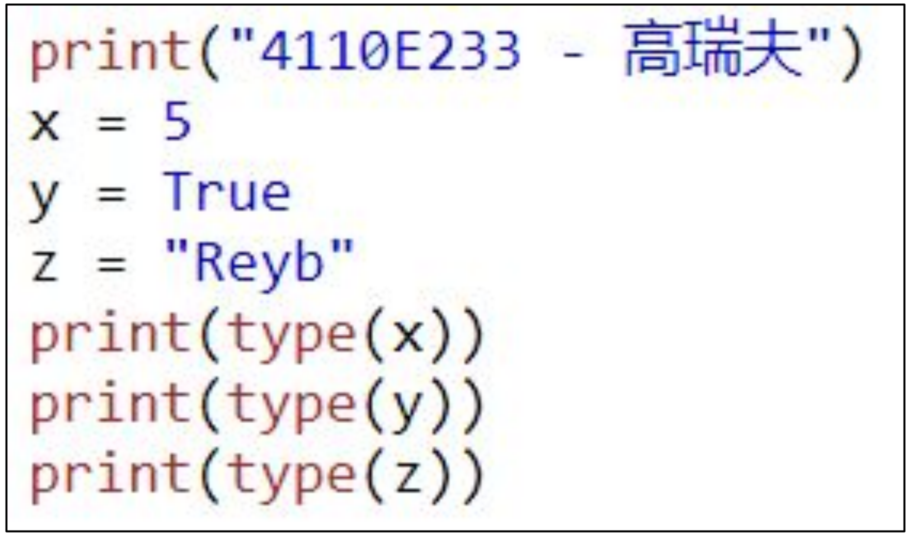
Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

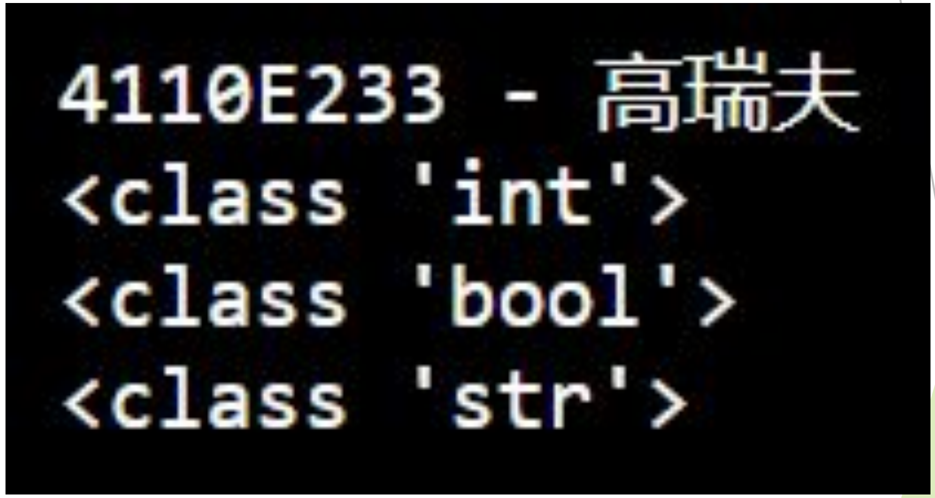
Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`
None Type:	`NoneType`

● Getting the Data Types

You can get the data type of any object by using the type() function:



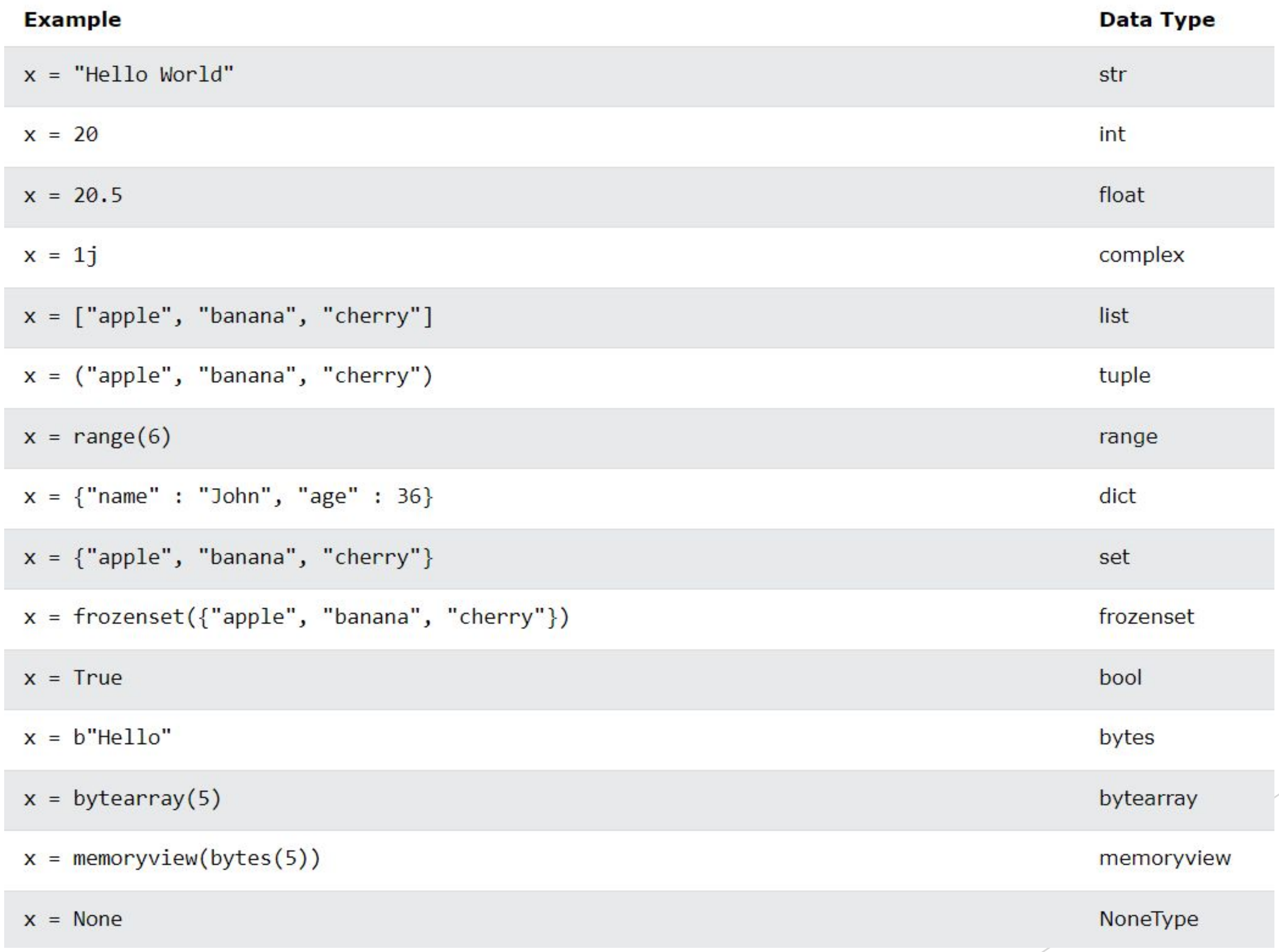
```
print("4110E233 - 高瑞夫")
x = 5
y = True
z = "Reyb"
print(type(x))
print(type(y))
print(type(z))
```



```
4110E233 - 高瑞夫
<class 'int'>
<class 'bool'>
<class 'str'>
```

● Setting the Data Types

In Python, the data type is set when you assign a value to a variable:



Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview
`x = None`	NoneType

PYTHON DATA TYPES

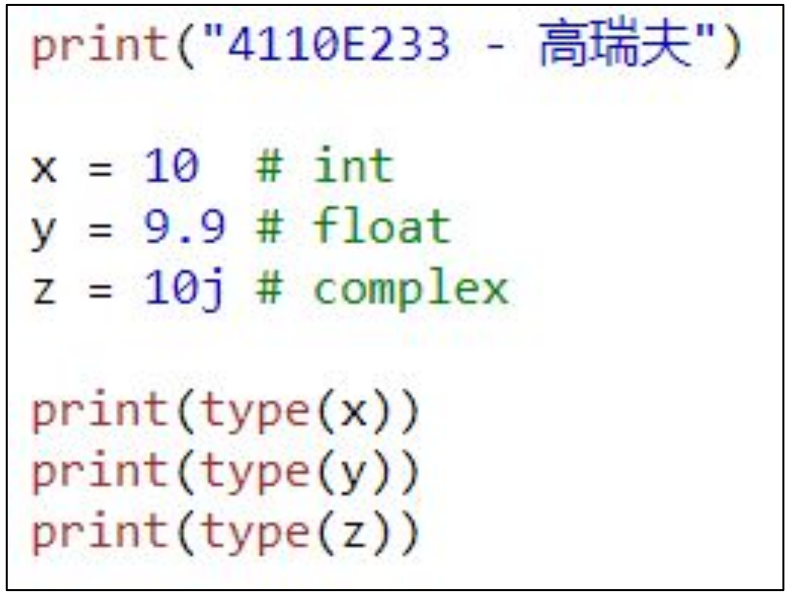
Python Numbers

Python Numbers

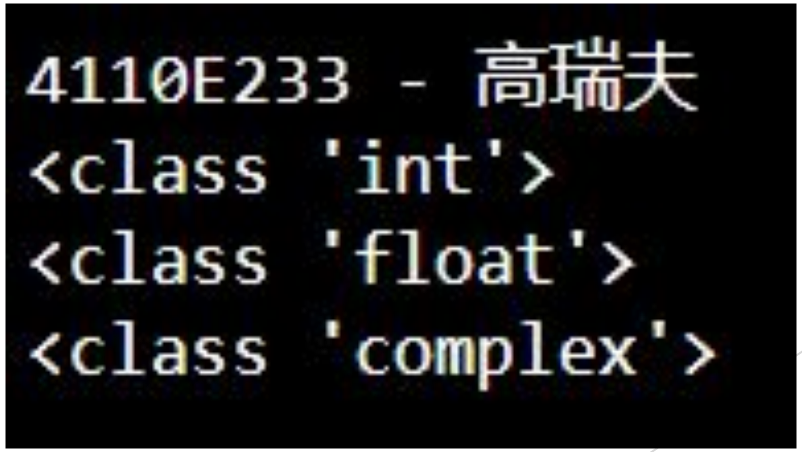
There are three numeric types in Python:

- int
- float
- complex

Variables of numeric types are created when you assign a value to them. To verify the type of any object in Python, use the type() function:



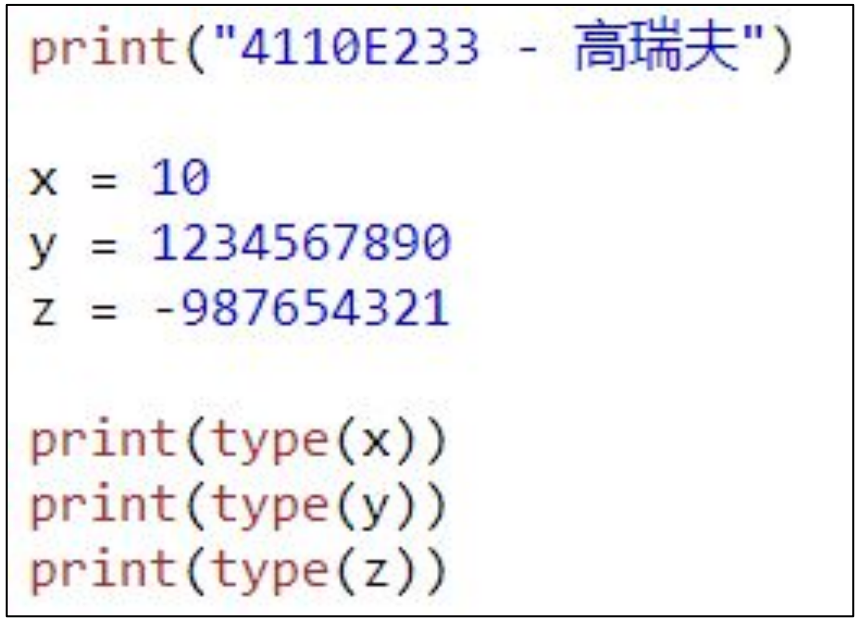
```
print("4110E233 - 高瑞夫")

x = 10  # int
y = 9.9 # float
z = 10j # complex

print(type(x))
print(type(y))
print(type(z))
```



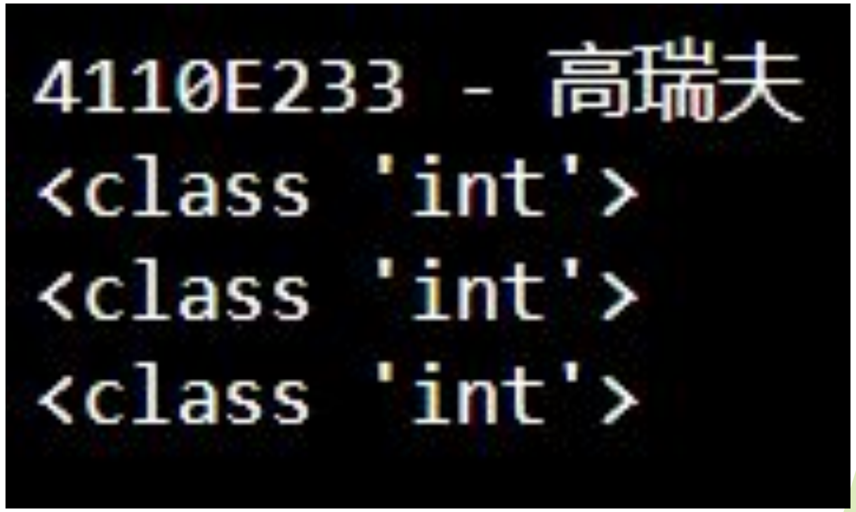
```
4110E233 - 高瑞夫
<class 'int'>
<class 'float'>
<class 'complex'>
```

● Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.



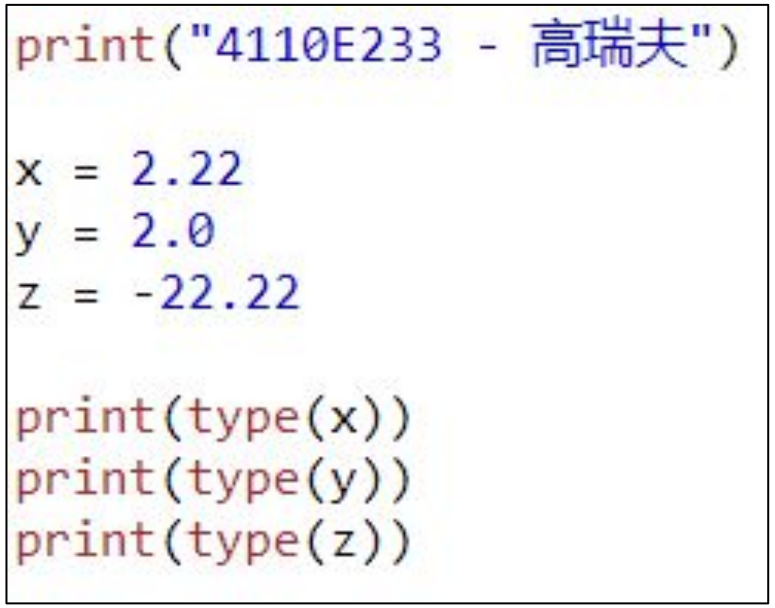
```
print("4110E233 - 高瑞夫")

x = 10
y = 1234567890
z = -987654321

print(type(x))
print(type(y))
print(type(z))
```



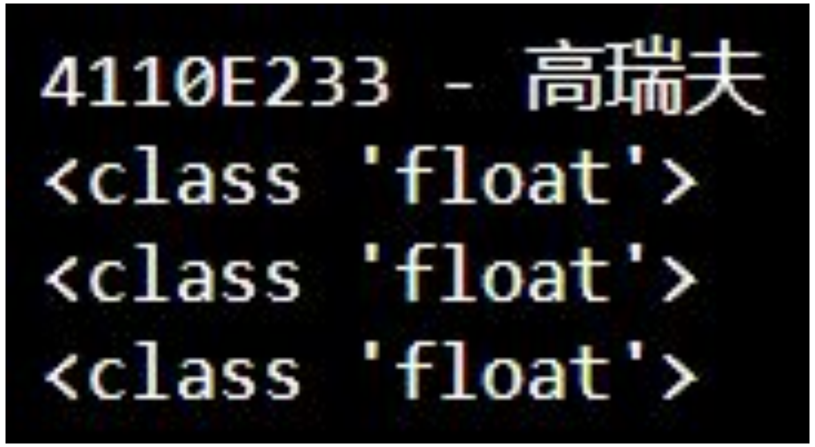
```
4110E233 - 高瑞夫
<class 'int'>
<class 'int'>
<class 'int'>
```

● Float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

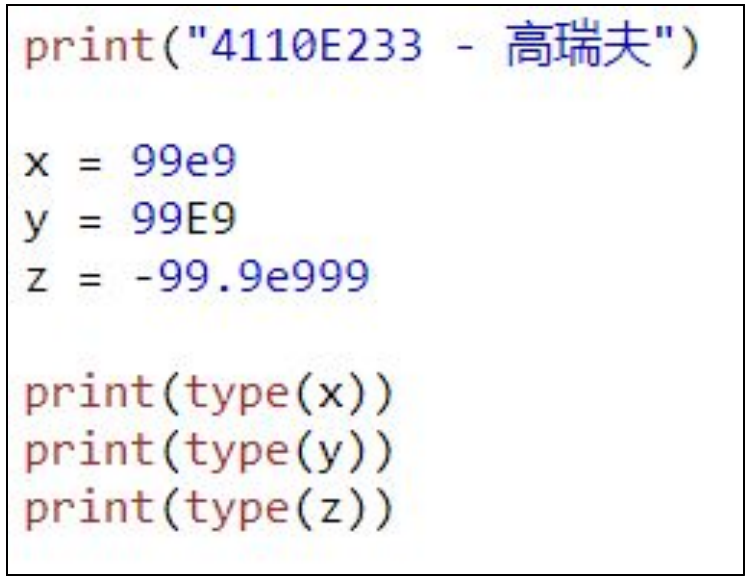


```
print("4110E233 - 高瑞夫")

x = 2.22
y = 2.0
z = -22.22

print(type(x))
print(type(y))
print(type(z))
```

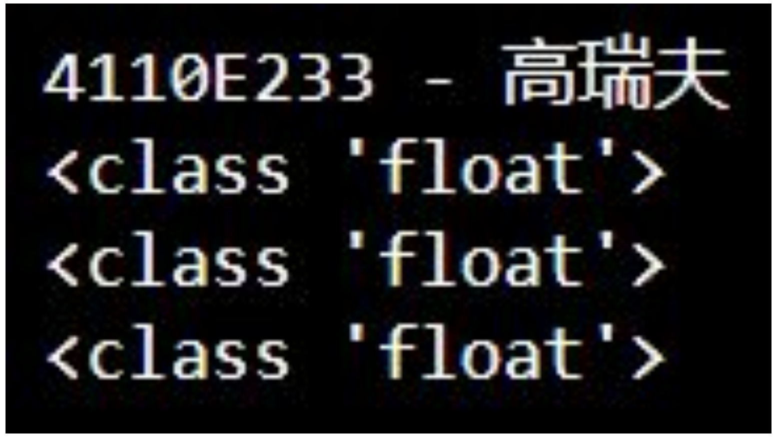


```
4110E233 - 高瑞夫
<class 'float'>
<class 'float'>
<class 'float'>
```

Float can also be scientific numbers with an "e" to indicate the power of 10.



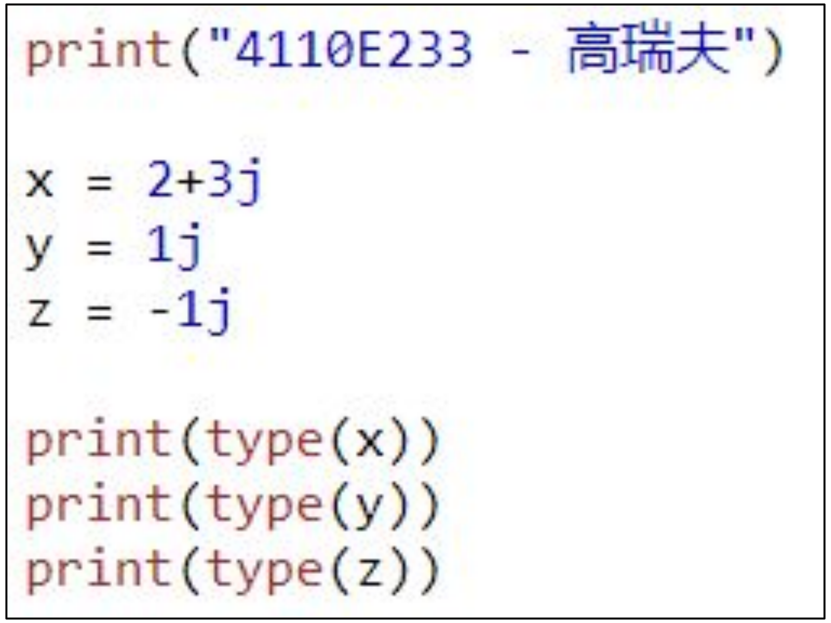
```
print("4110E233 - 高瑞夫")

x = 99e9
y = 99E9
z = -99.9e999

print(type(x))
print(type(y))
print(type(z))
```



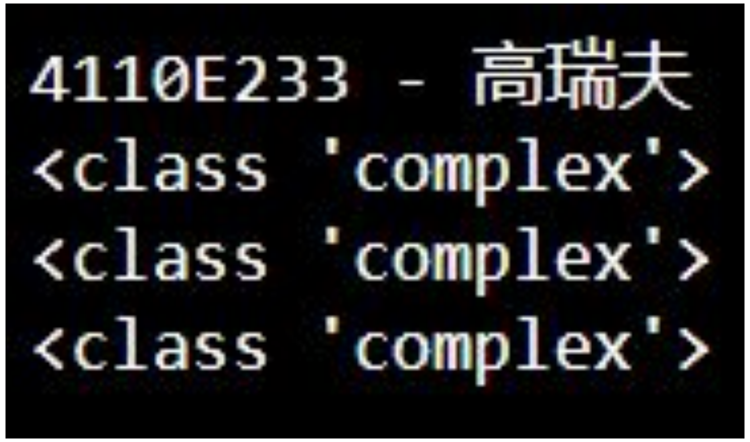
```
4110E233 - 高瑞夫
<class 'float'>
<class 'float'>
<class 'float'>
```

● Complex

Complex numbers are written with a "j" as the imaginary part:



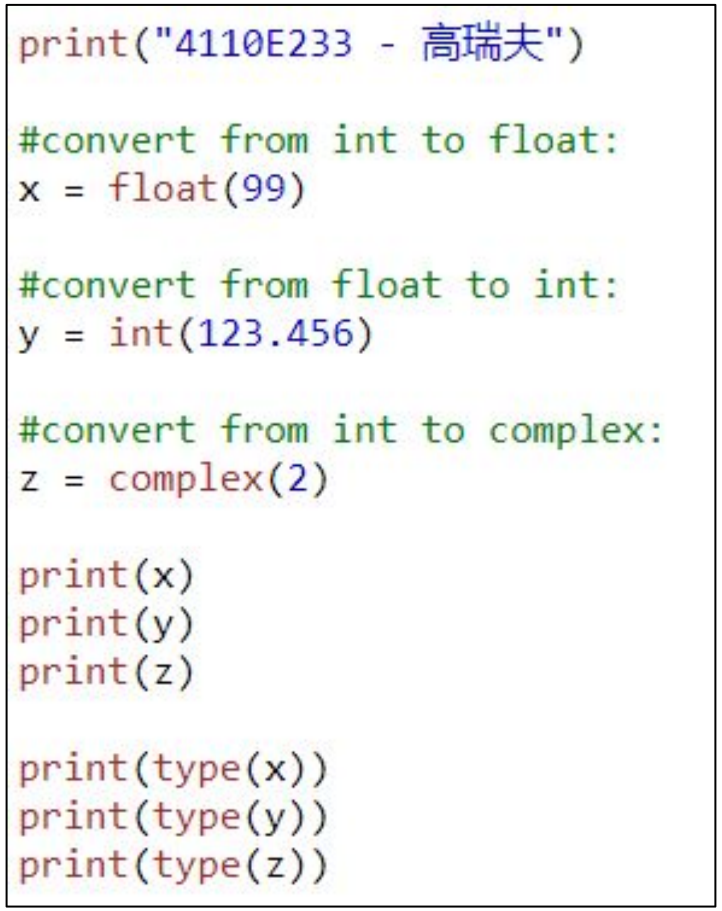
```
print("4110E233 - 高瑞夫")

x = 2+3j
y = 1j
z = -1j

print(type(x))
print(type(y))
print(type(z))
```



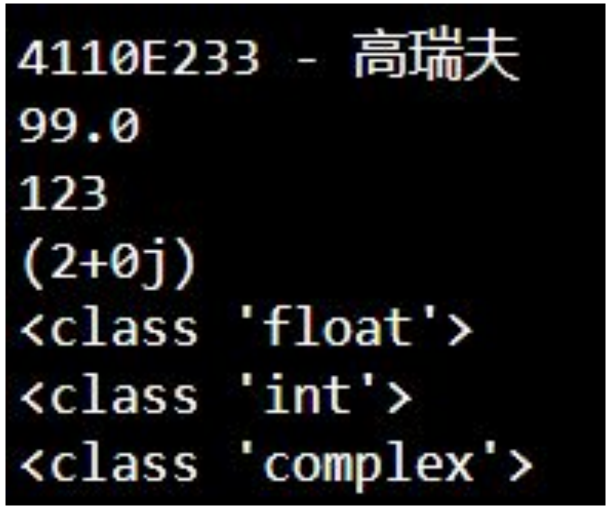
```
4110E233 - 高瑞夫
<class 'complex'>
<class 'complex'>
<class 'complex'>
```

● Type Conversion

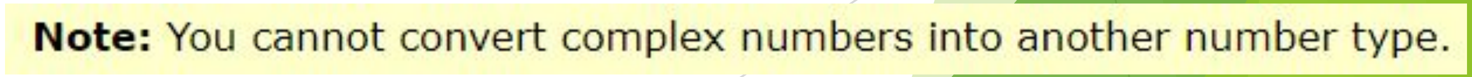
You can convert from one type to another with the int(), float(), and complex() methods:



```
print("4110E233 - 高瑞夫")

#convert from int to float:
x = float(99)

#convert from float to int:
y = int(123.456)

#convert from int to complex:
z = complex(2)

print(x)
print(y)
print(z)

print(type(x))
print(type(y))
print(type(z))
```



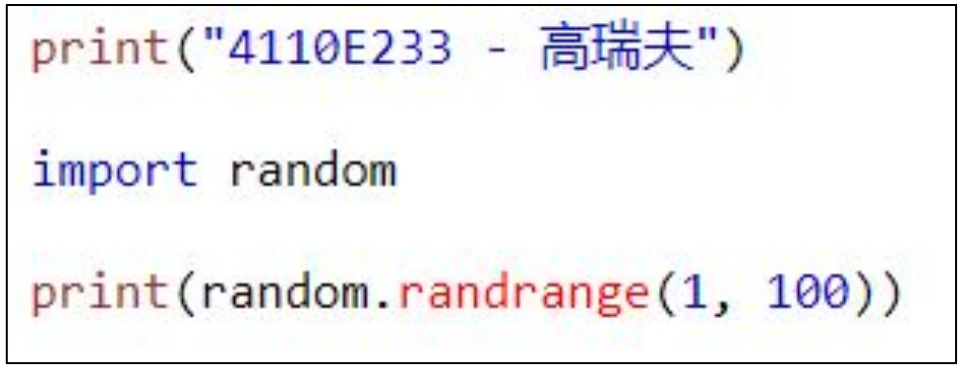
```
4110E233 - 高瑞夫
99.0
123
(2+0j)
<class 'float'>
<class 'int'>
<class 'complex'>
```



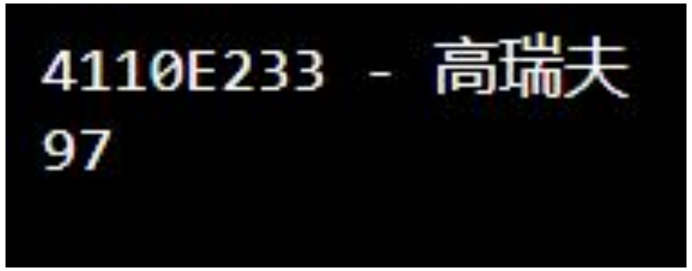
Note: You cannot convert complex numbers into another number type.

● Random Numbers

Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:



```
print("4110E233 - 高瑞夫")

import random

print(random.randrange(1, 100))
```



```
4110E233 - 高瑞夫
97
```

PYTHON DATA TYPES

Python Casting

Python Casting

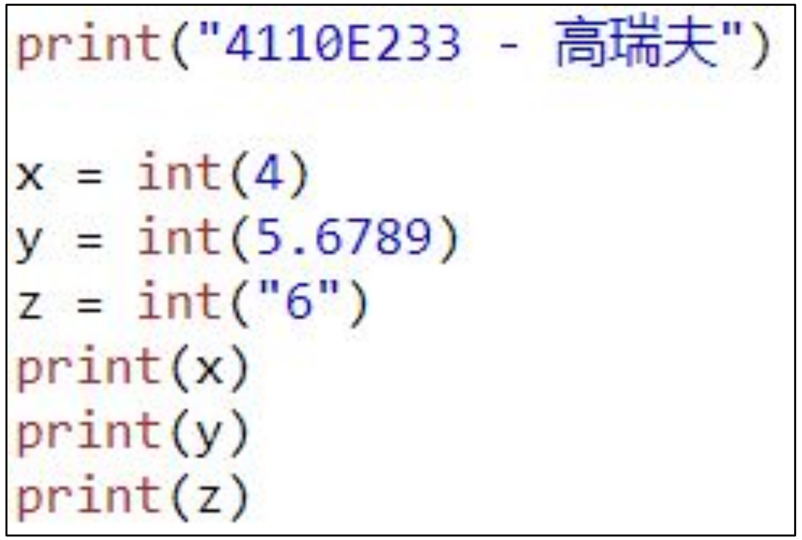
- Specify a Variable Type

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

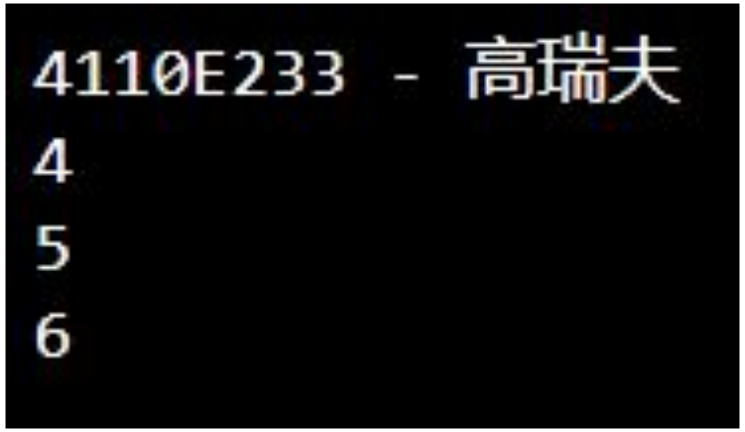
Casting in python is therefore done using constructor functions:

- int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Example: Integers

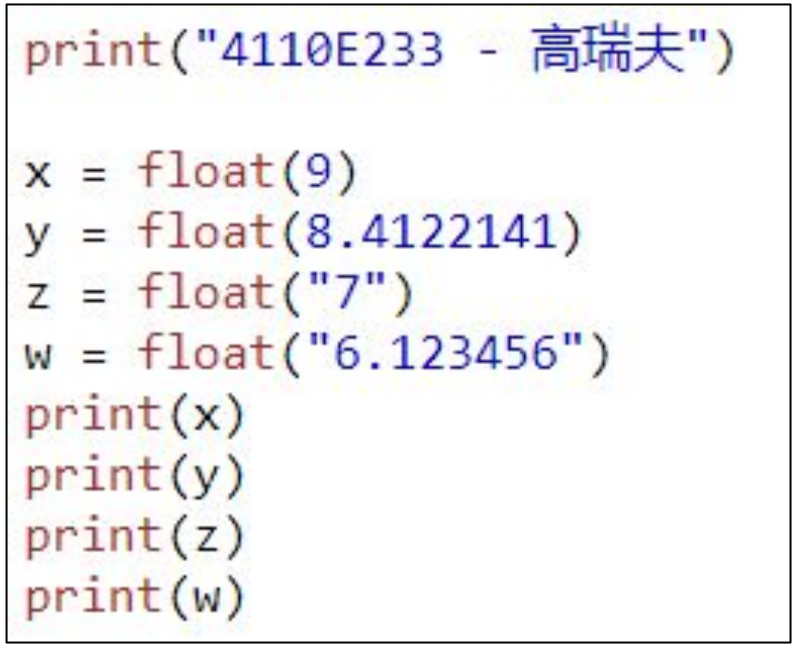


```
print("4110E233 - 高瑞夫")

x = int(4)
y = int(5.6789)
z = int("6")
print(x)
print(y)
print(z)
```

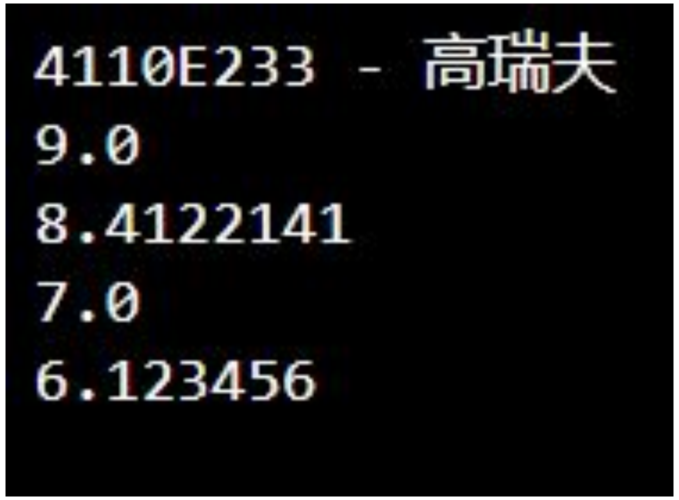


```
4110E233 - 高瑞夫
4
5
6
```

Example: Floats

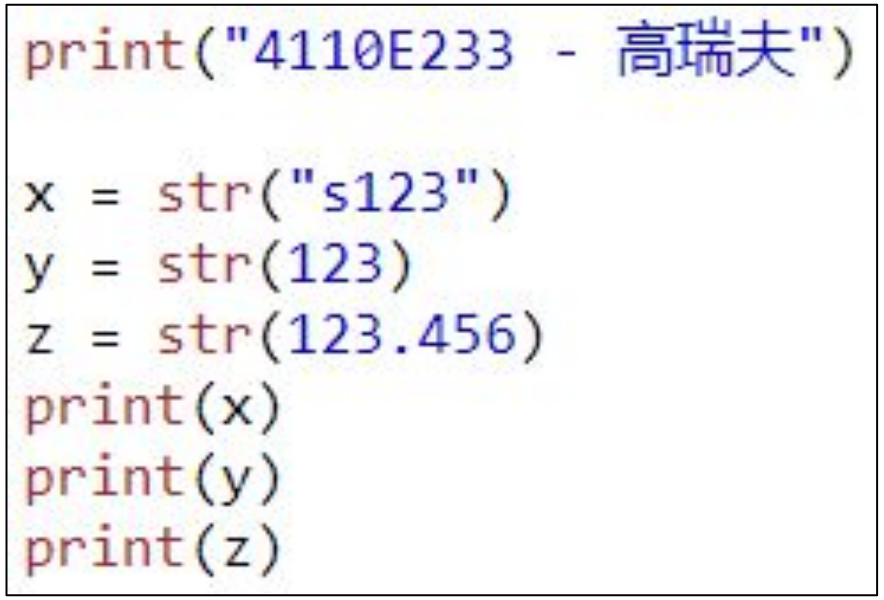


```
print("4110E233 - 高瑞夫")

x = float(9)
y = float(8.4122141)
z = float("7")
w = float("6.123456")
print(x)
print(y)
print(z)
print(w)
```

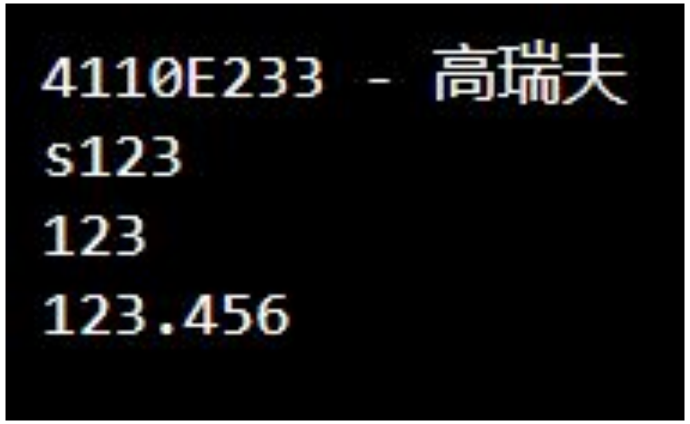


```
4110E233 - 高瑞夫
9.0
8.4122141
7.0
6.123456
```

Example: Strings



```
print("4110E233 - 高瑞夫")

x = str("s123")
y = str(123)
z = str(123.456)
print(x)
print(y)
print(z)
```



```
4110E233 - 高瑞夫
s123
123
123.456
```

PYTHON DATA TYPES

Python Strings

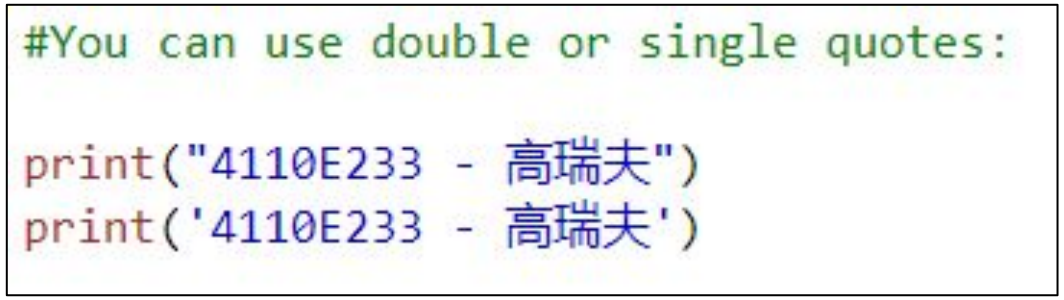
Python Strings

- Strings

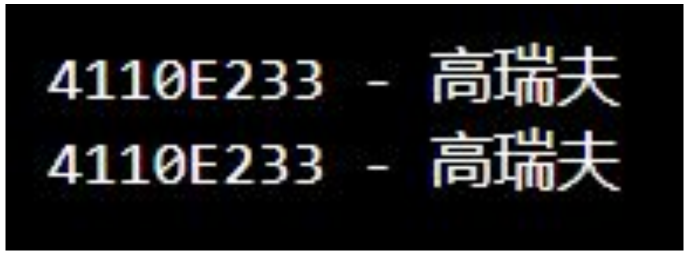
Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:



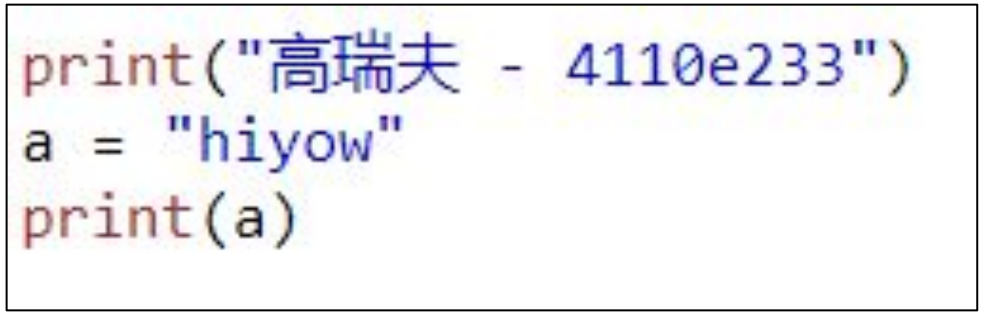
```
#You can use double or single quotes:

print("4110E233 - 高瑞夫")
print('4110E233 - 高瑞夫')
```



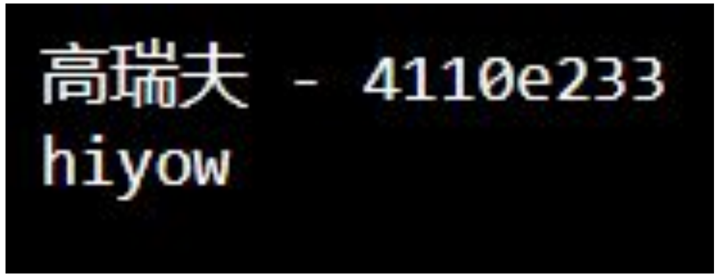
```
4110E233 - 高瑞夫
4110E233 - 高瑞夫
```

● Assign Strings to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:



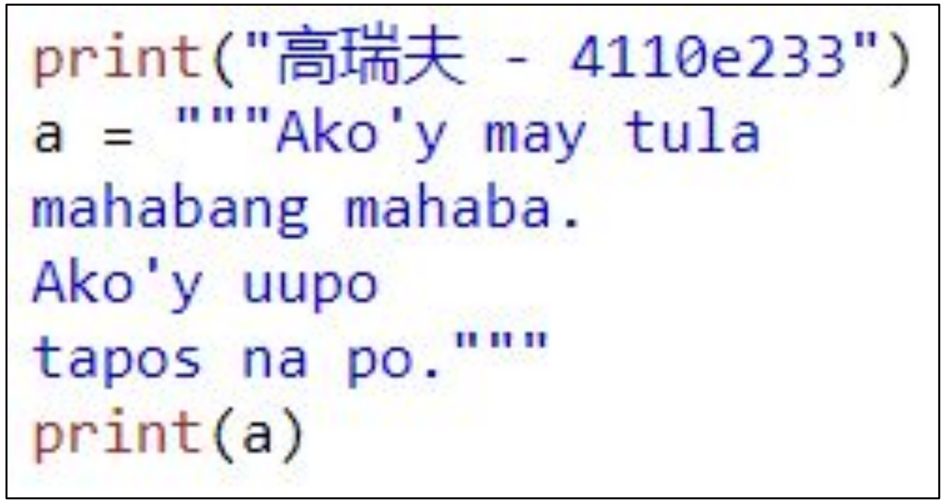
```
print("高瑞夫 - 4110e233")
a = "hiyow"
print(a)
```



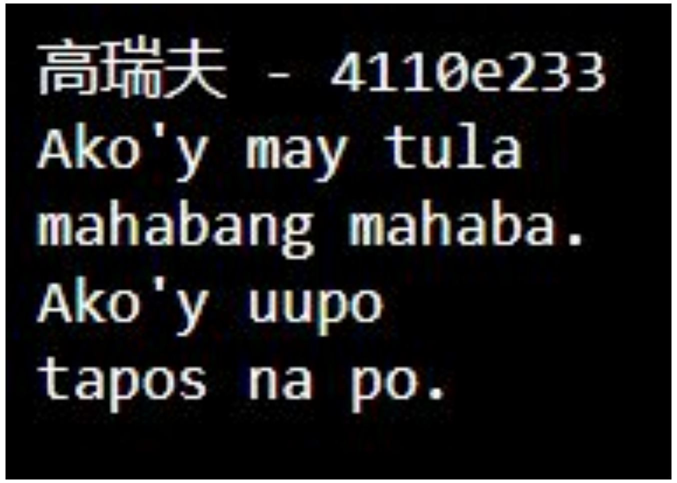
```
高瑞夫 - 4110e233
hiyow
```

● Multiline Strings

You can assign a multiline string to a variable by using three quotes:



```
print("高瑞夫 - 4110e233")
a = """Ako'y may tula
mahabang mahaba.
Ako'y uupo
tapos na po."""
print(a)
```



```
高瑞夫 - 4110e233
Ako'y may tula
mahabang mahaba.
Ako'y uupo
tapos na po.
```

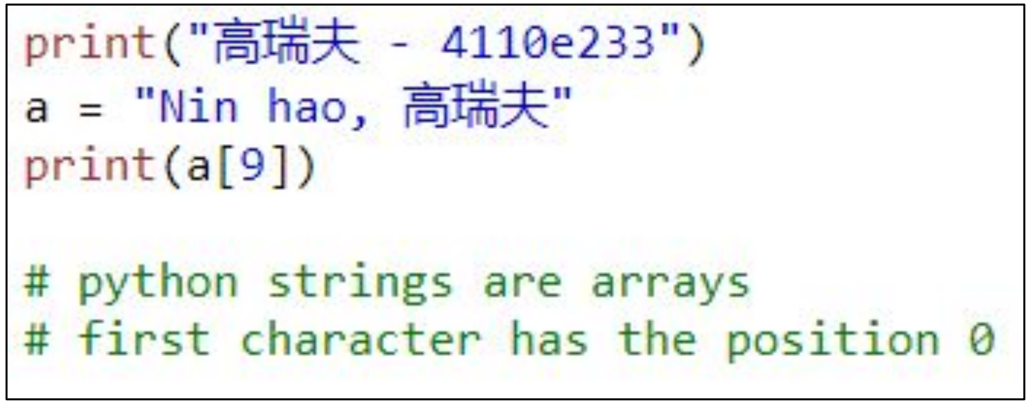
Note: in the result, the line breaks are inserted at the same position as in the code.

● Strings are Arrays

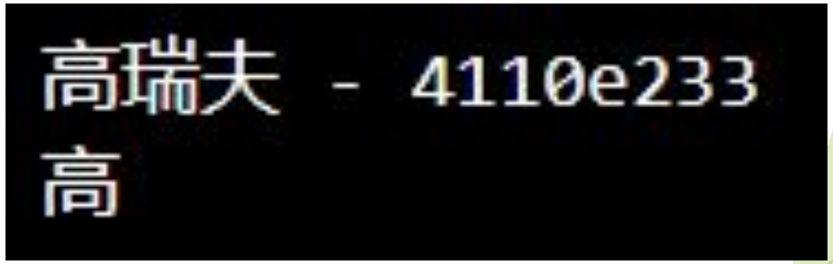
Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.



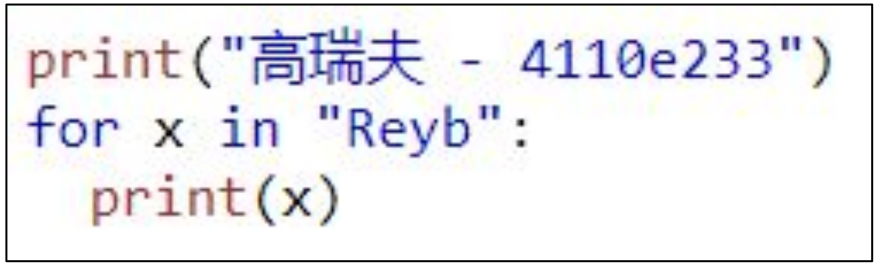
```
print("高瑞夫 - 4110e233")
a = "Nin hao, 高瑞夫"
print(a[9])

# python strings are arrays
# first character has the position 0
```



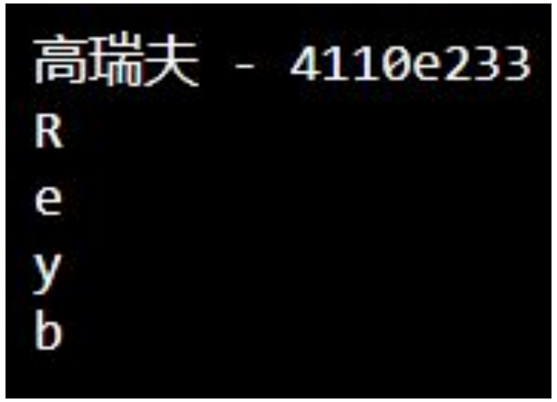
```
高瑞夫 - 4110e233
高
```

● Looping Through a String

Since strings are arrays, we can loop through the characters in a string, with a for loop.



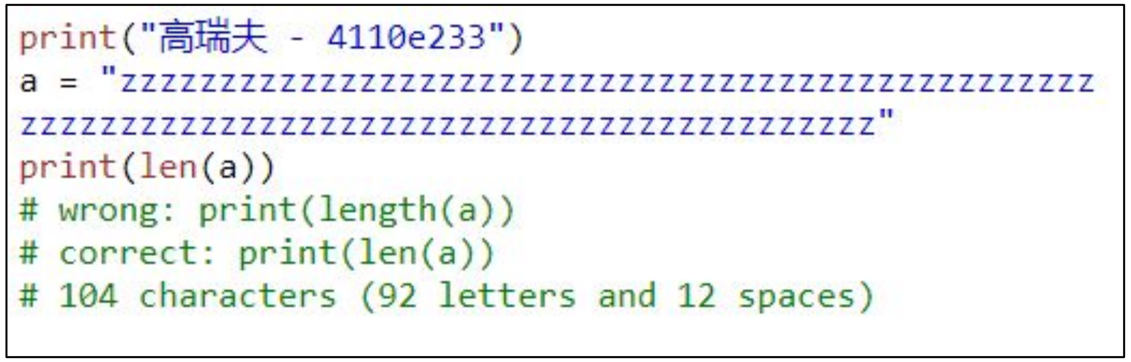
```
print("高瑞夫 - 4110e233")
for x in "Reyb":
  print(x)
```



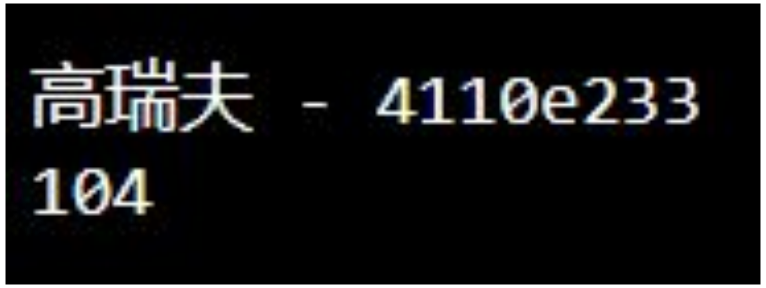
```
高瑞夫 - 4110e233
R
e
y
b
```

● String Length

To get the length of a string, use the len() function.



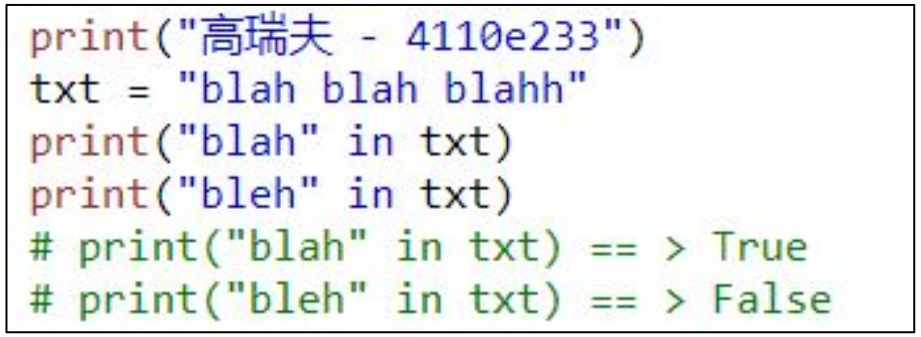
```
print("高瑞夫 - 4110e233")
a = "zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz"
print(len(a))
# wrong: print(length(a))
# correct: print(len(a))
# 104 characters (92 letters and 12 spaces)
```



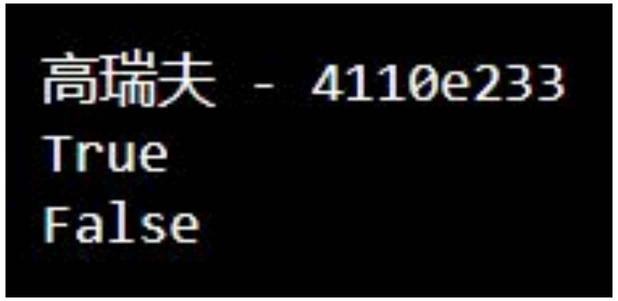
```
高瑞夫 - 4110e233
104
```

Check String

- To check if a certain phrase or character is present in a string, we can use the keyword in.
- Keyword in vs if
- True or False

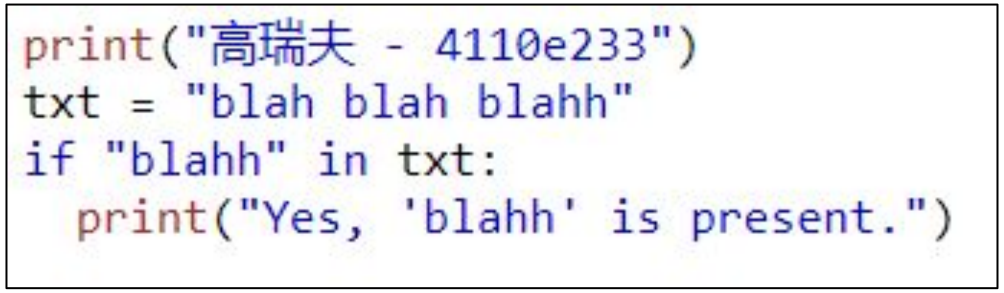


```
print("高瑞夫 - 4110e233")
txt = "blah blah blahh"
print("blah" in txt)
print("bleh" in txt)
# print("blah" in txt) == > True
# print("bleh" in txt) == > False
```

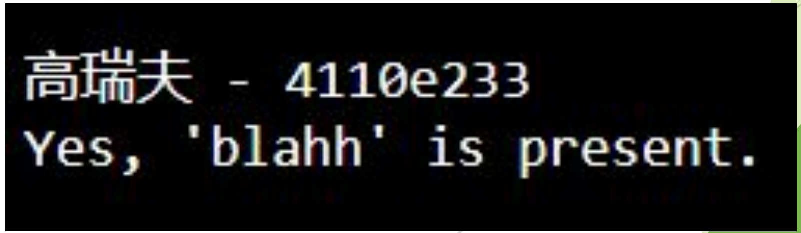


```
高瑞夫 - 4110e233
True
False
```

Use it in an if statement:



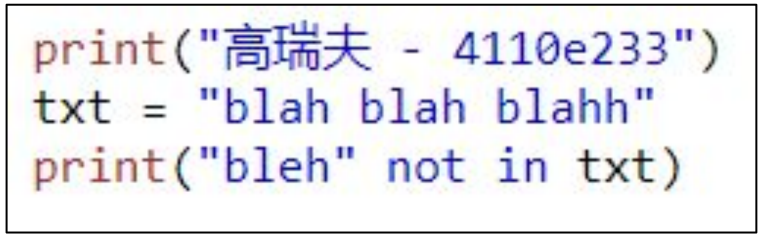
```
print("高瑞夫 - 4110e233")
txt = "blah blah blahh"
if "blahh" in txt:
  print("Yes, 'blahh' is present.")
```



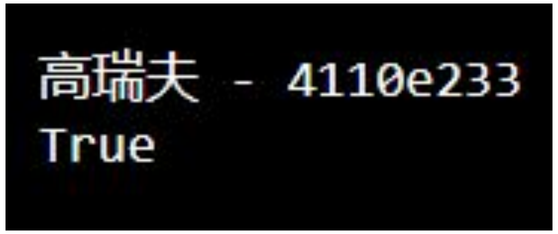
```
高瑞夫 - 4110e233
Yes, 'blahh' is present.
```

● Check if NOT

To check if a certain phrase or character is NOT present in a string, we can use the keyword not in.

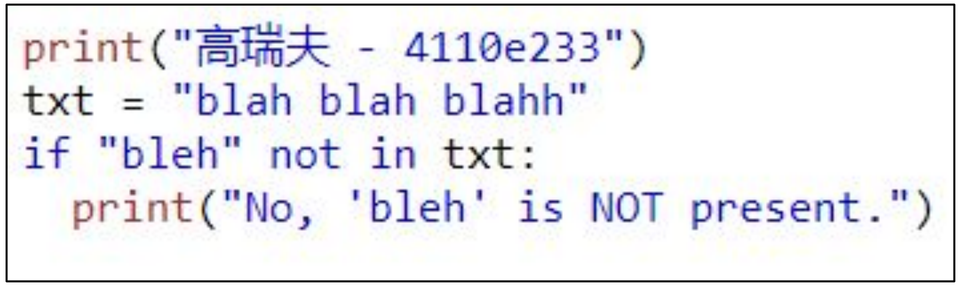


```
print("高瑞夫 - 4110e233")
txt = "blah blah blahh"
print("bleh" not in txt)
```

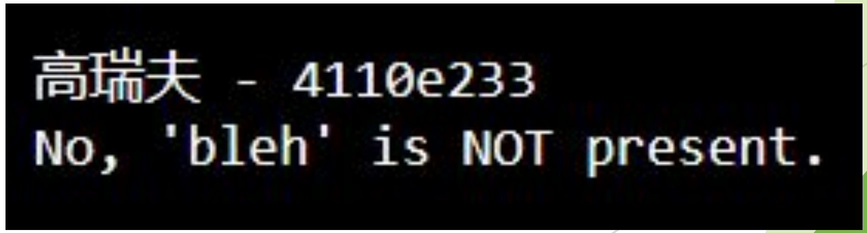


```
高瑞夫 - 4110e233
True
```

Use it in an if statement:



```
print("高瑞夫 - 4110e233")
txt = "blah blah blahh"
if "bleh" not in txt:
  print("No, 'bleh' is NOT present.")
```



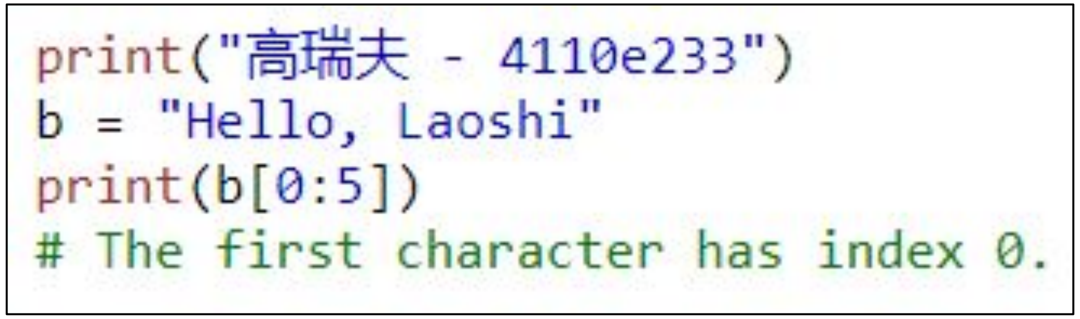
```
高瑞夫 - 4110e233
No, 'bleh' is NOT present.
```

Slicing Strings

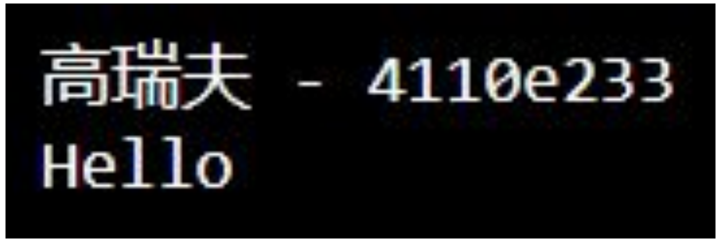
● Slicing

You can return a range of characters by using the slice syntax.

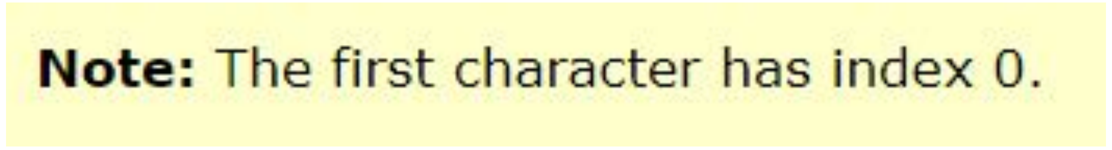
Specify the start index and the end index, separated by a colon, to return a part of the string.



```
print("高瑞夫 - 4110e233")
b = "Hello, Laoshi"
print(b[0:5])
# The first character has index 0.
```



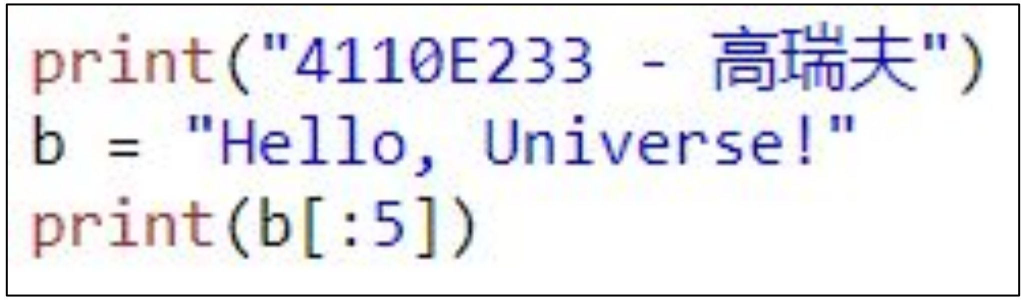
```
高瑞夫 - 4110e233
Hello
```



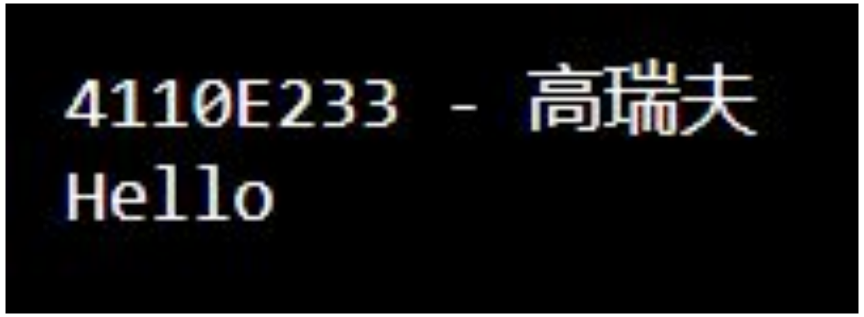
Note: The first character has index 0.

● Slice From the Start

By leaving out the start index, the range will start at the first character:



```
print("4110E233 - 高瑞夫")
b = "Hello, Universe!"
print(b[:5])
```

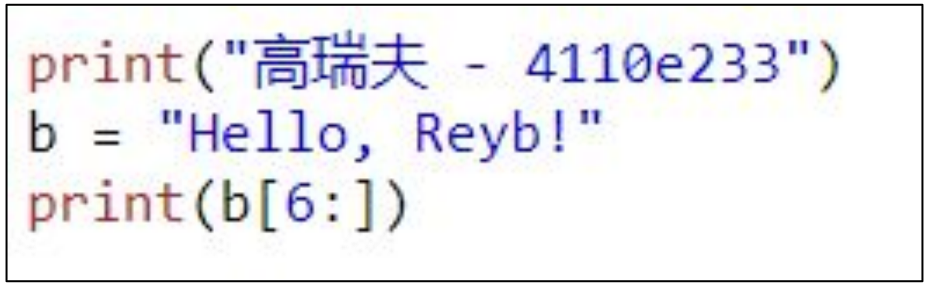


```
4110E233 - 高瑞夫
Hello
```

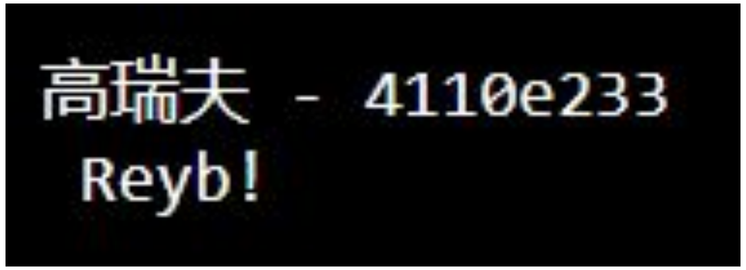
Note: Get the characters from the start to position 5 (not included)

● Slice To the End

By leaving out the *end* index, the range will go to the end:



```
print("高瑞夫 - 4110e233")
b = "Hello, Reyb!"
print(b[6:])
```

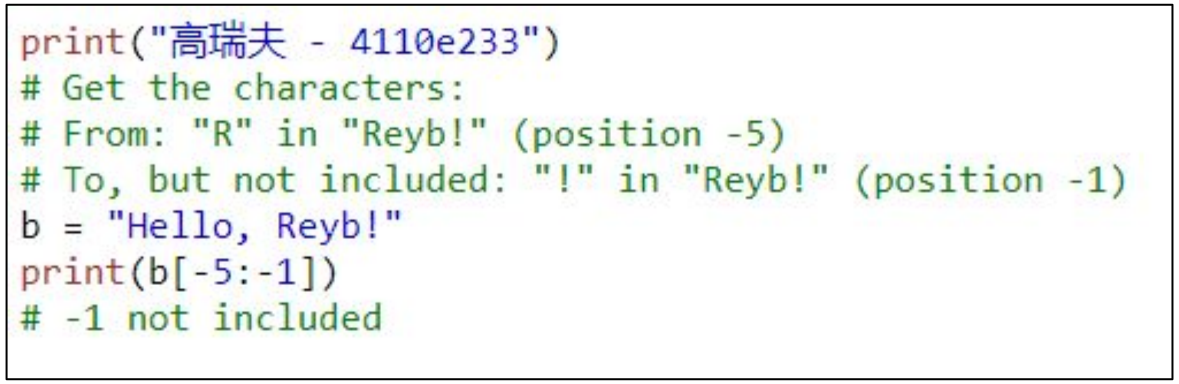


```
高瑞夫 - 4110e233
 Reyb!
```

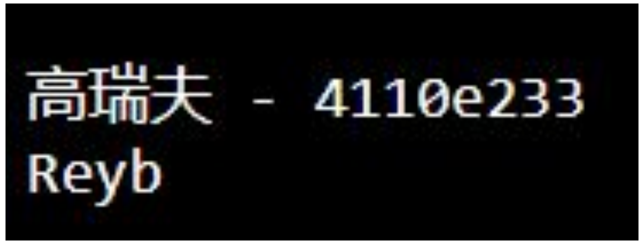
Note: Get the characters from position 2, and all the way to the end

● Negative Indexing

Use negative indexes to start the slice from the end of the string:



```
print("高瑞夫 - 4110e233")
# Get the characters:
# From: "R" in "Reyb!" (position -5)
# To, but not included: "!" in "Reyb!" (position -1)
b = "Hello, Reyb!"
print(b[-5:-1])
# -1 not included
```

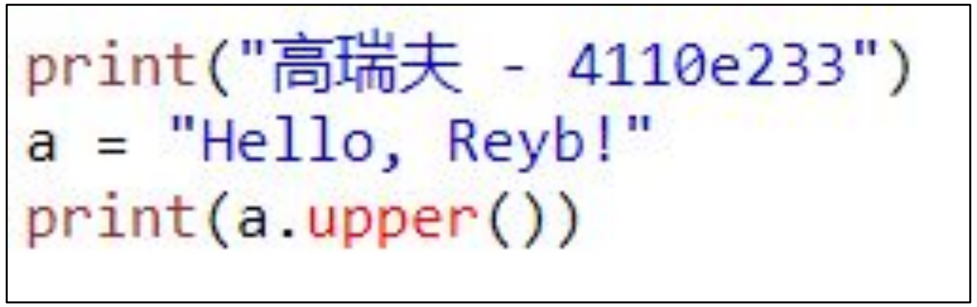


```
高瑞夫 - 4110e233
Reyb
```

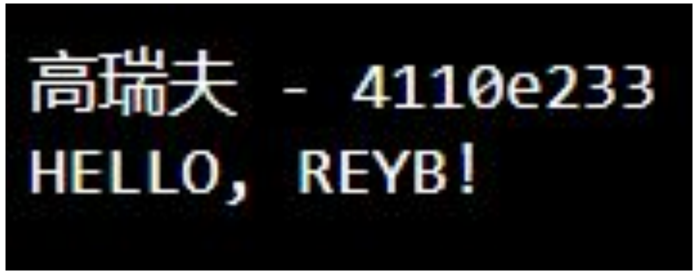
Modify Strings

- Upper Case

The upper() method returns the string in upper case:



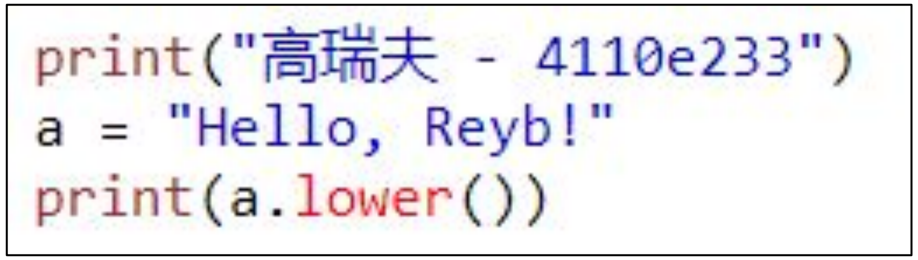
```
print("高瑞夫 - 4110e233")
a = "Hello, Reyb!"
print(a.upper())
```



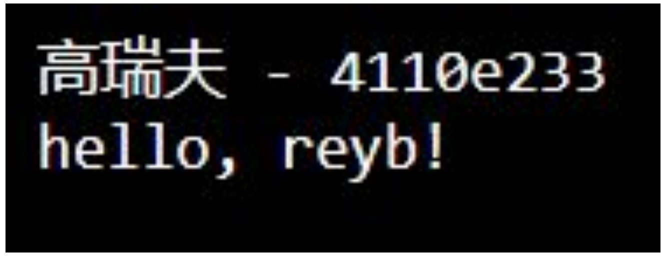
```
高瑞夫 - 4110e233
HELLO, REYB!
```

● Upper Case

The lower() method returns the string in lower case:



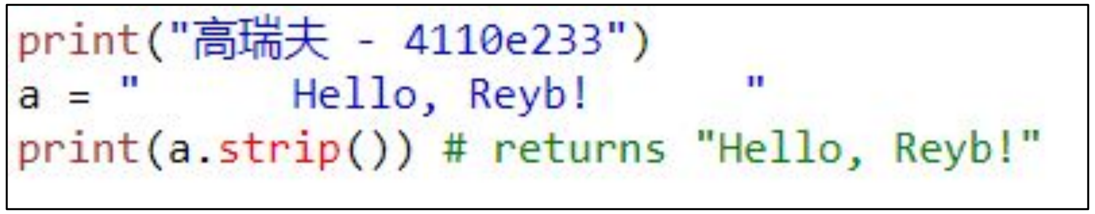
```
print("高瑞夫 - 4110e233")
a = "Hello, Reyb!"
print(a.lower())
```



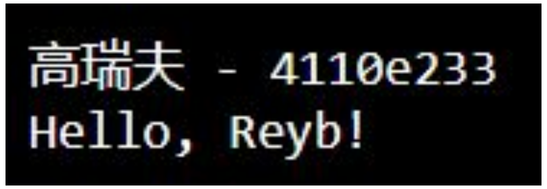
```
高瑞夫 - 4110e233
hello, reyb!
```

● Remove Whitespace

- Whitespace is the space before and/or after the actual text, and very often you want to remove this space.
- The strip() method removes any whitespace from the beginning or the end



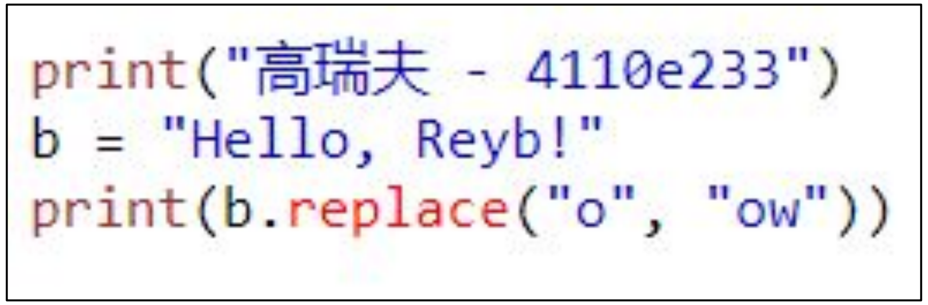
```
print("高瑞夫 - 4110e233")
a = "       Hello, Reyb!       "
print(a.strip()) # returns "Hello, Reyb!"
```



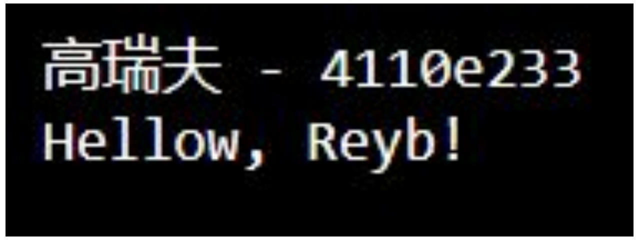
```
高瑞夫 - 4110e233
Hello, Reyb!
```

● Replace String

The replace() method replaces a string with another string



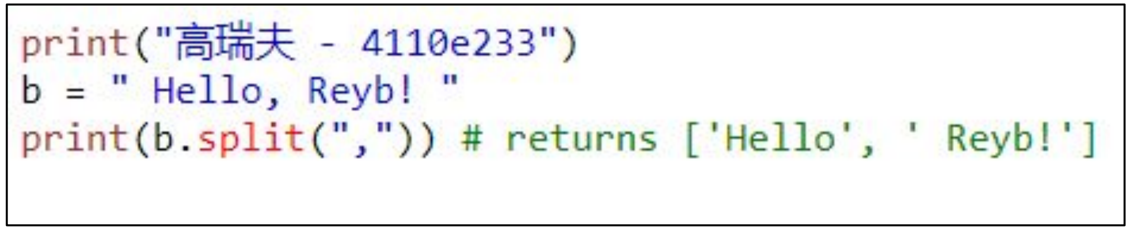
```
print("高瑞夫 - 4110e233")
b = "Hello, Reyb!"
print(b.replace("o", "ow"))
```



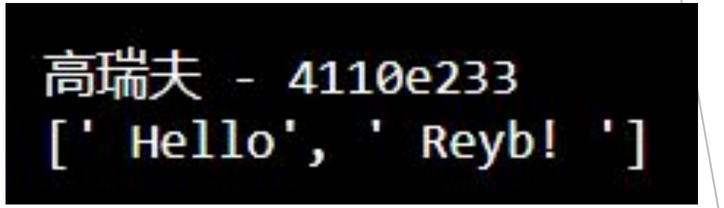
```
高瑞夫 - 4110e233
Hellow, Reyb!
```

● Split String

The split() method returns a list where the text between the specified separator becomes the list items.



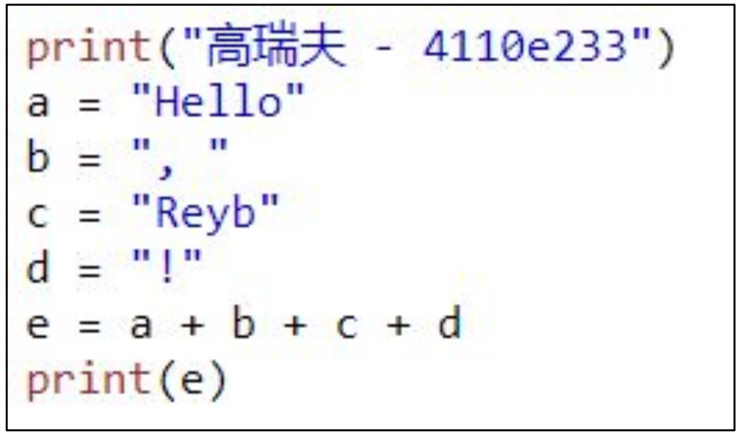
```
print("高瑞夫 - 4110e233")
b = " Hello, Reyb! "
print(b.split(",")) # returns ['Hello', ' Reyb!']
```



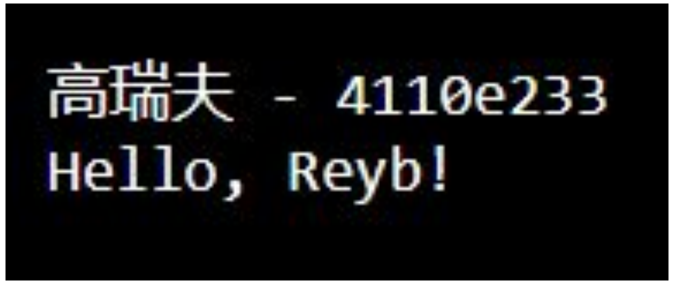
```
高瑞夫 - 4110e233
[' Hello', ' Reyb! ']
```

String Concatenation

To concatenate, or combine, two strings you can use the + operator.

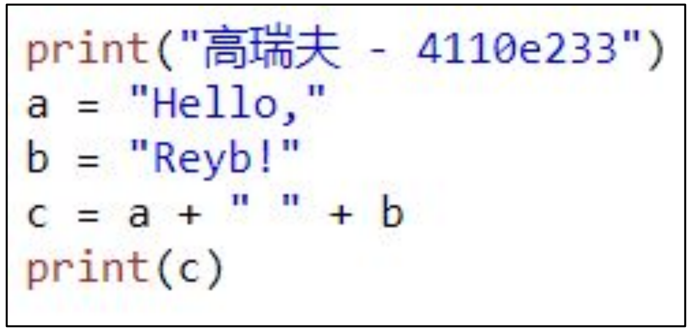


```
print("高瑞夫 - 4110e233")
a = "Hello"
b = ", "
c = "Reyb"
d = "!"
e = a + b + c + d
print(e)
```

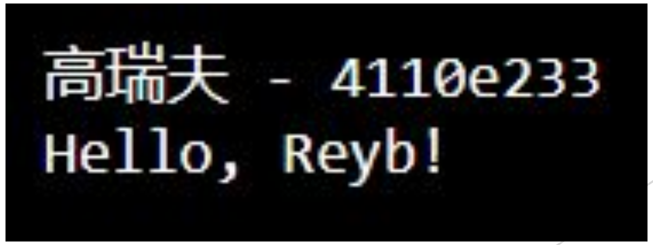


```
高瑞夫 - 4110e233
Hello, Reyb!
```

To add a space between them, add a " ":



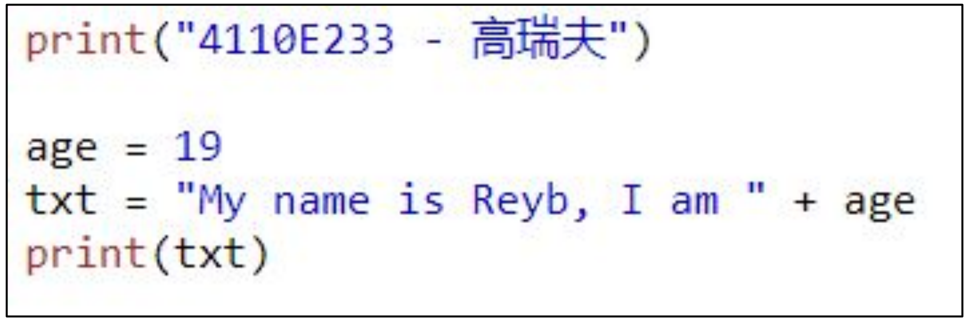
```
print("高瑞夫 - 4110e233")
a = "Hello,"
b = "Reyb!"
c = a + " " + b
print(c)
```



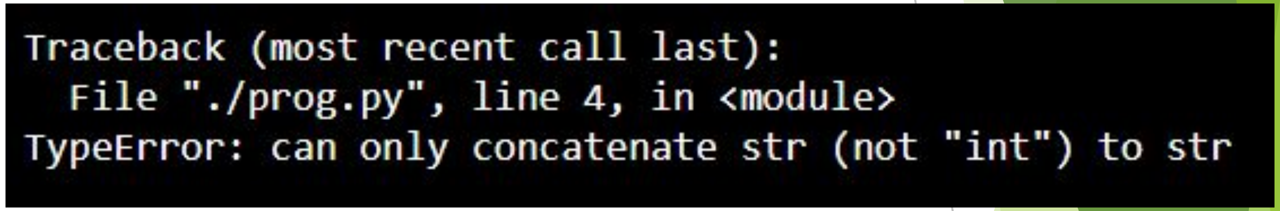
```
高瑞夫 - 4110e233
Hello, Reyb!
```

Format Strings

As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:



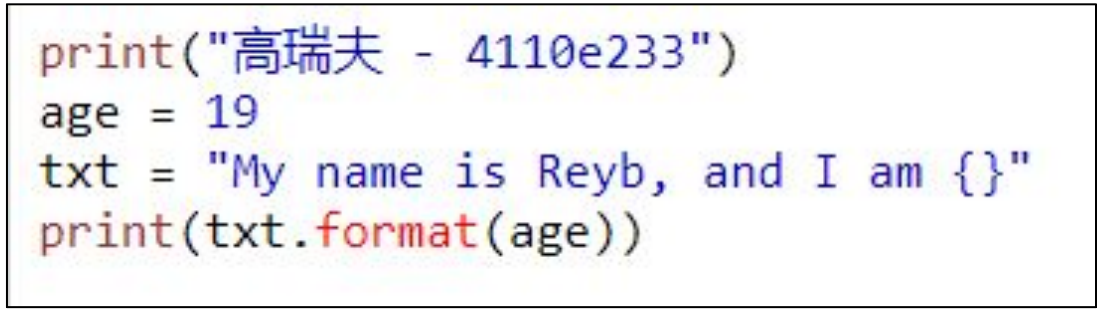
```
print("4110E233 - 高瑞夫")

age = 19
txt = "My name is Reyb, I am " + age
print(txt)
```



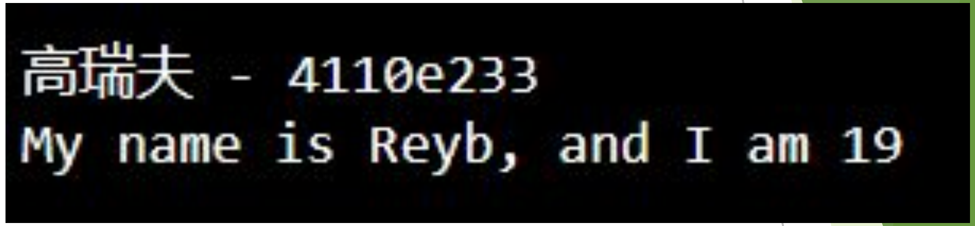
```
Traceback (most recent call last):
  File "./prog.py", line 4, in <module>
TypeError: can only concatenate str (not "int") to str
```

But we can combine strings and numbers by using the format() method!

The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

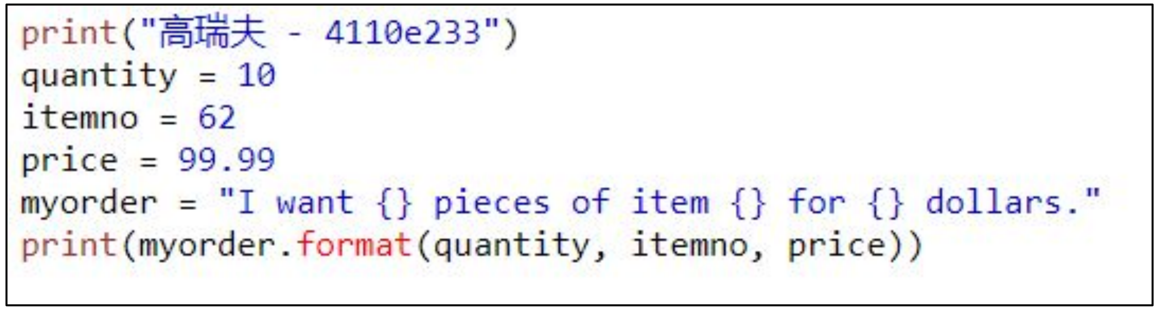


```
print("高瑞夫 - 4110e233")
age = 19
txt = "My name is Reyb, and I am {}"
print(txt.format(age))
```

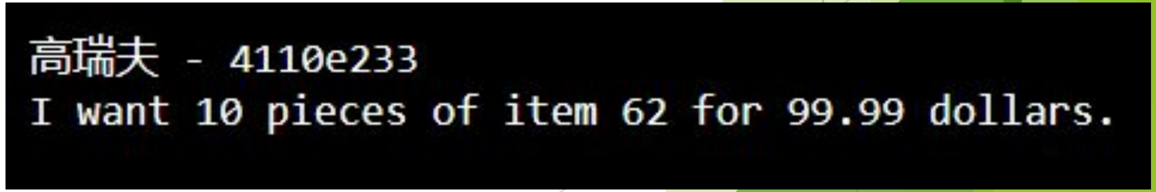


```
高瑞夫 - 4110e233
My name is Reyb, and I am 19
```

The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

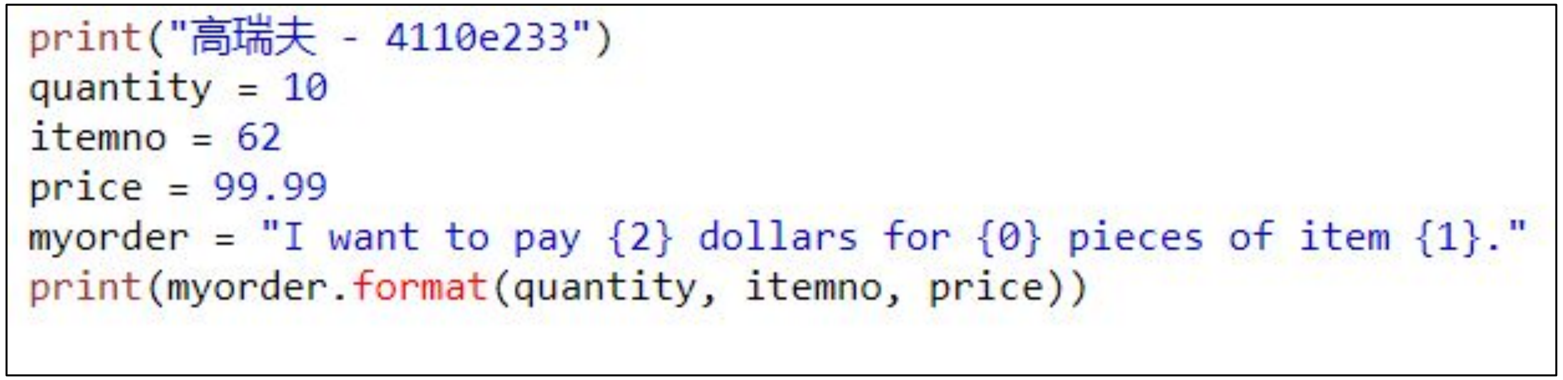


```
print("高瑞夫 - 4110e233")
quantity = 10
itemno = 62
price = 99.99
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

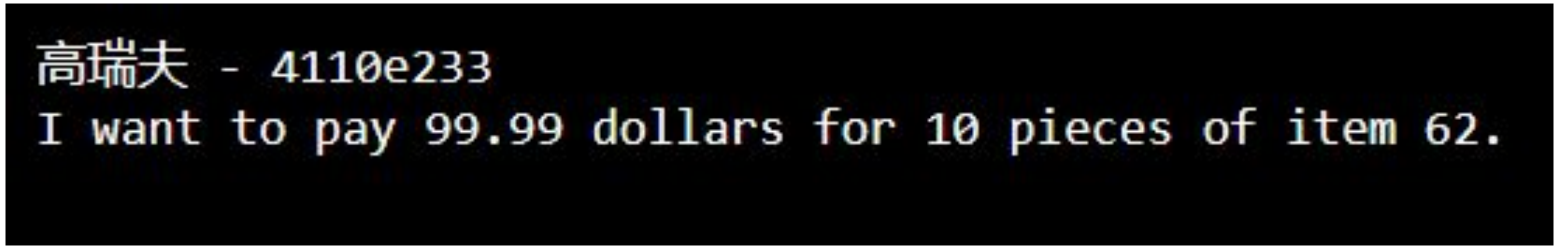


```
高瑞夫 - 4110e233
I want 10 pieces of item 62 for 99.99 dollars.
```

You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:



```
print("高瑞夫 - 4110e233")
quantity = 10
itemno = 62
price = 99.99
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```



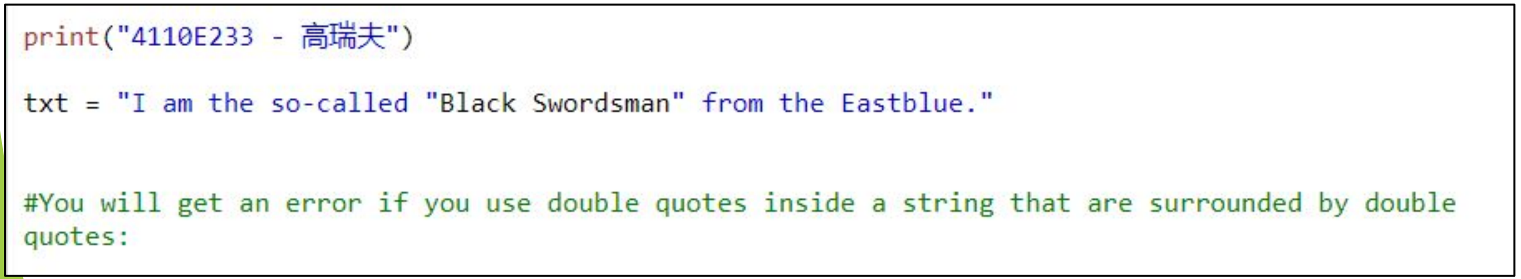
```
高瑞夫 - 4110e233
I want to pay 99.99 dollars for 10 pieces of item 62.
```

Escape Characters

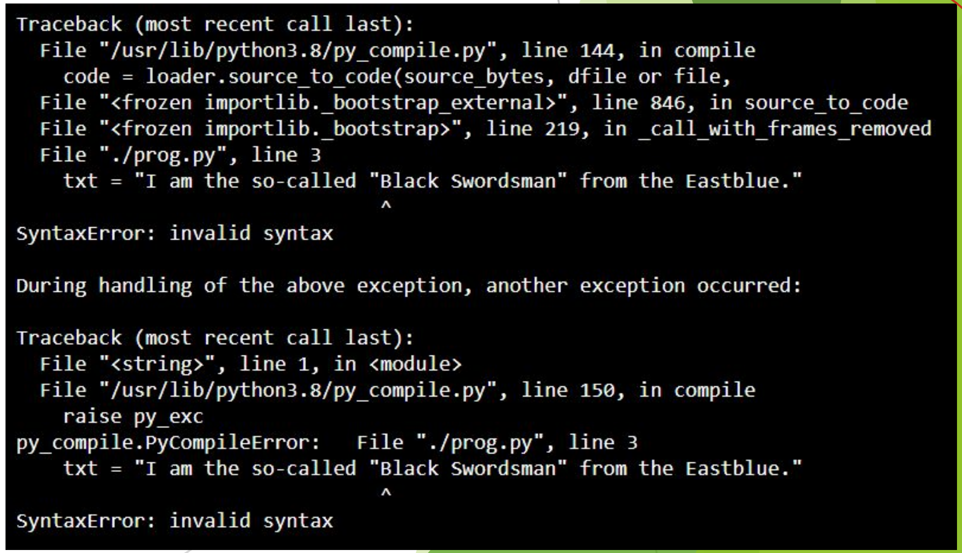
To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

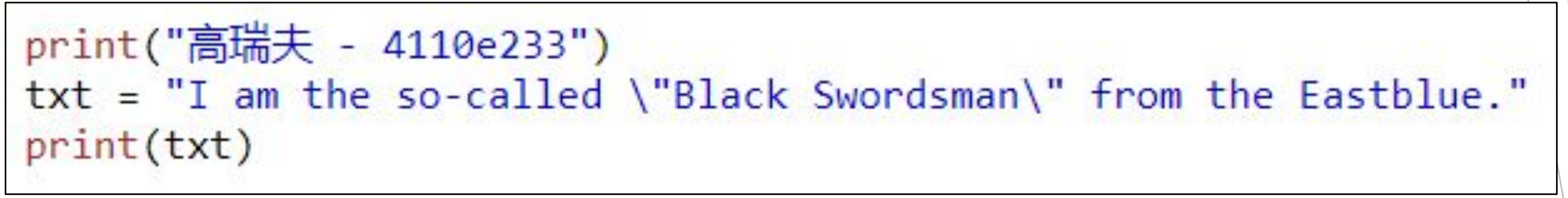


```
print("4110E233 - 高瑞夫")

txt = "I am the so-called "Black Swordsman" from the Eastblue."


#You will get an error if you use double quotes inside a string that are surrounded by double
quotes:
```

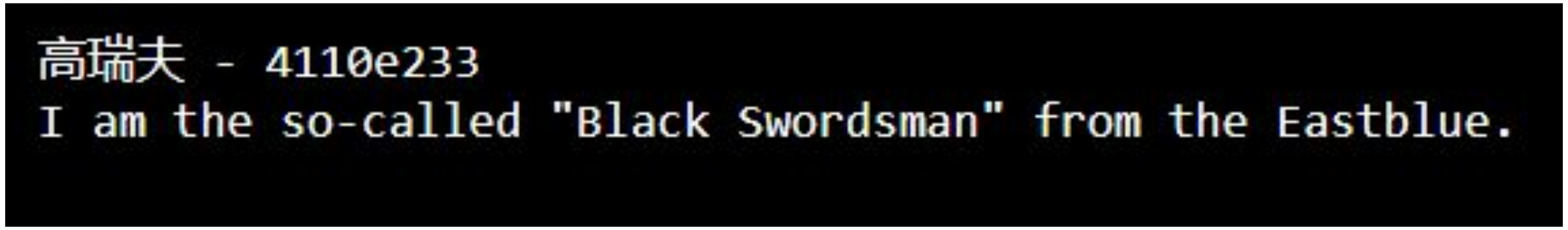


```
Traceback (most recent call last):
  File "/usr/lib/python3.8/py_compile.py", line 144, in compile
    code = loader.source_to_code(source_bytes, dfile or file,
  File "<frozen importlib._bootstrap_external>", line 846, in source_to_code
  File "<frozen importlib._bootstrap>", line 219, in _call_with_frames_removed
  File "./prog.py", line 3
    txt = "I am the so-called "Black Swordsman" from the Eastblue."
                               ^
SyntaxError: invalid syntax

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib/python3.8/py_compile.py", line 150, in compile
    raise py_exc
py_compile.PyCompileError:   File "./prog.py", line 3
    txt = "I am the so-called "Black Swordsman" from the Eastblue."
                               ^
SyntaxError: invalid syntax
```

To fix this problem, use the escape character \":

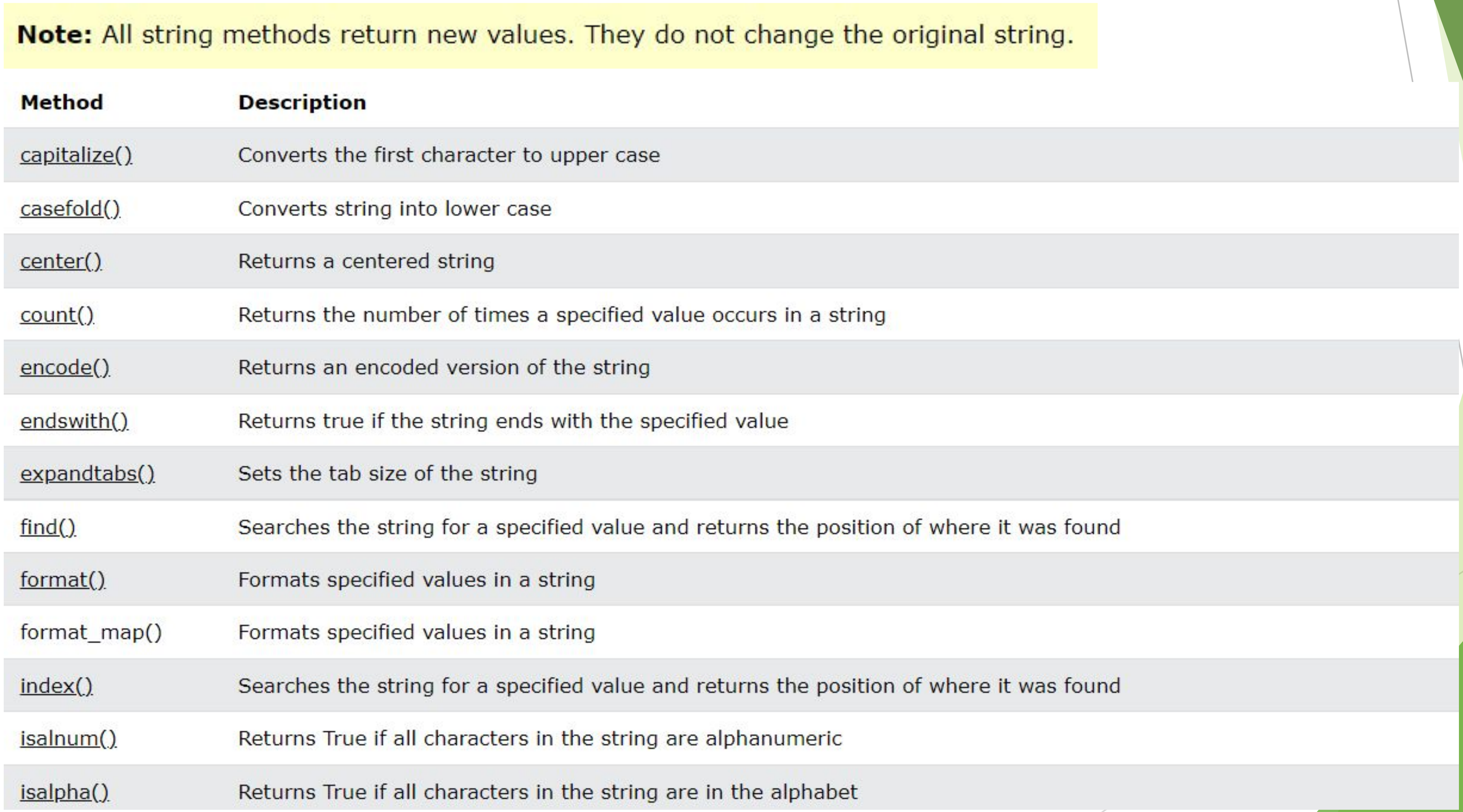


```
print("高瑞夫 - 4110e233")
txt = "I am the so-called \"Black Swordsman\" from the Eastblue."
print(txt)
```



```
高瑞夫 - 4110e233
I am the so-called "Black Swordsman" from the Eastblue.
```

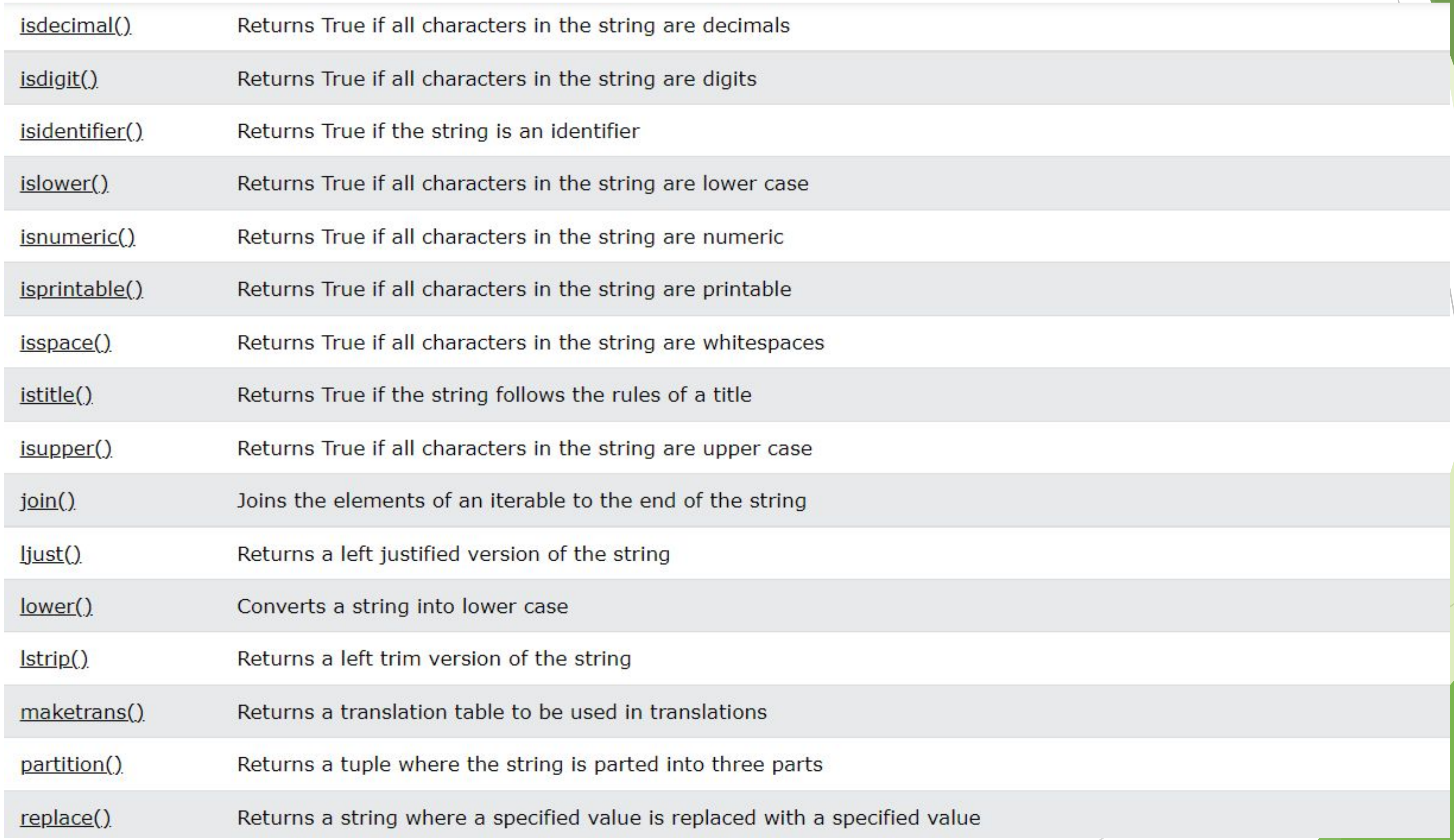
String Methods



Note: All string methods return new values. They do not change the original string.

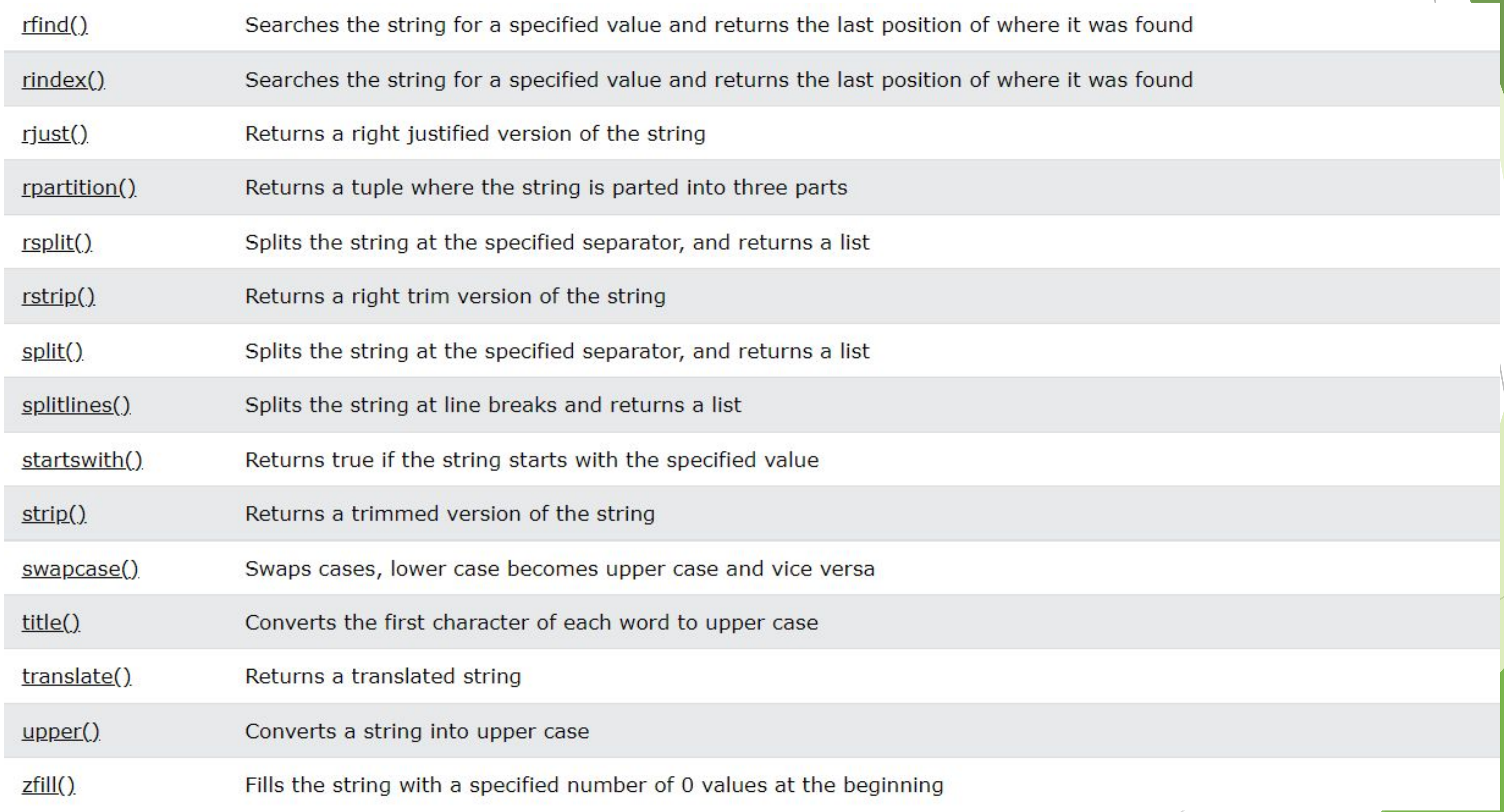
Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet

String Methods



isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value

String Methods



rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

PYTHON DATA TYPES

Python Booleans

Python Booleans

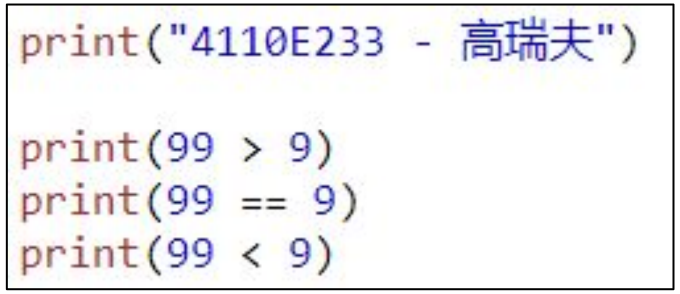
Booleans represent one of two values: True or False.

- ### Booleans Values

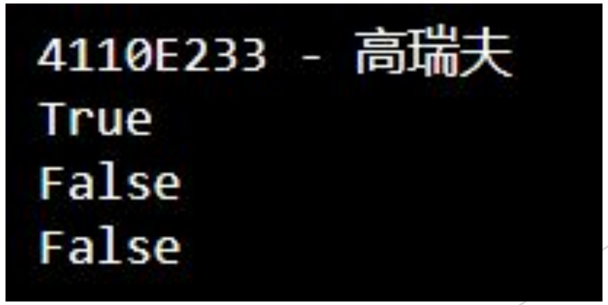
In programming you often need to know if an expression is True or False.

You can evaluate any expression in Python, and get one of two answers, True or False.

When you compare two values, the expression is evaluated and Python returns the Boolean answer:

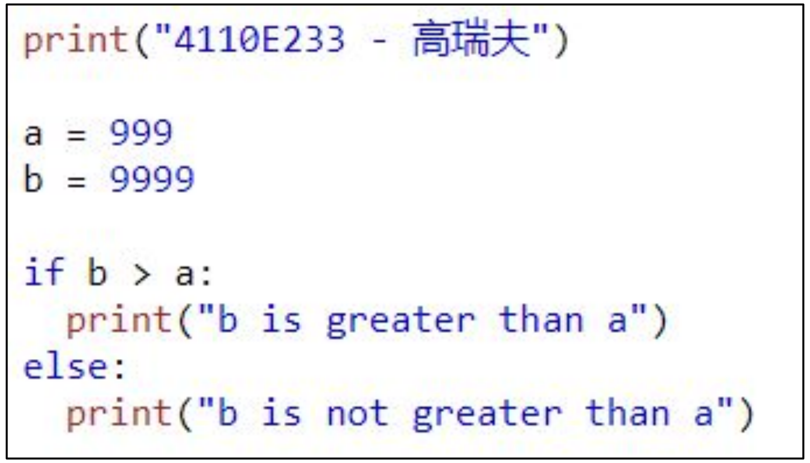


```python
print("4110E233 - 高瑞夫")

print(99 > 9)
print(99 == 9)
print(99 < 9)
```

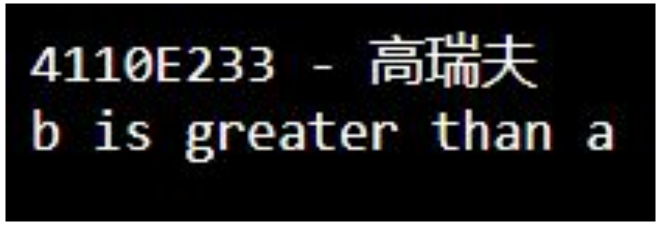


```
4110E233 - 高瑞夫
True
False
False
```

When you run a condition in an if statement, Python returns True or False:



```
print("4110E233 - 高瑞夫")

a = 999
b = 9999

if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

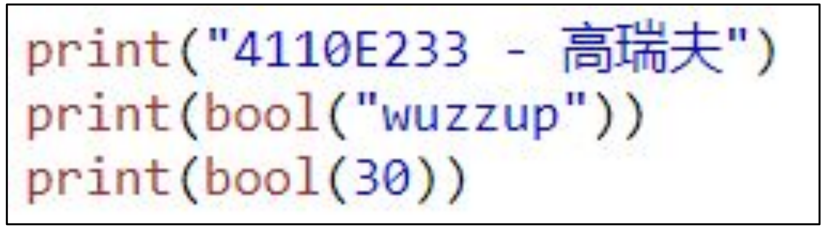


```
4110E233 - 高瑞夫
b is greater than a
```

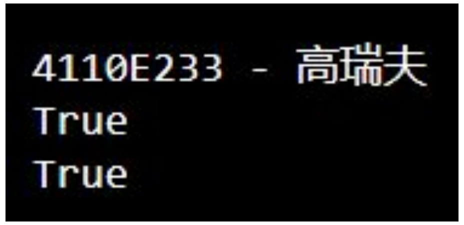
● Evaluate Values and Variables

The bool() function allows you to evaluate any value, and give you True or False in return,

Evaluate a string and a number:

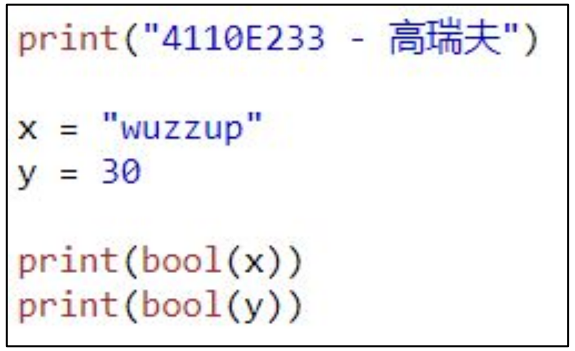


```
print("4110E233 - 高瑞夫")
print(bool("wuzzup"))
print(bool(30))
```

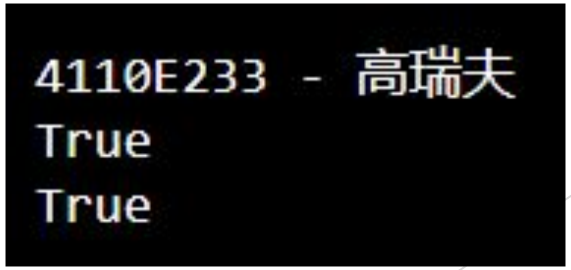


```
4110E233 - 高瑞夫
True
True
```

Evaluate two variables:



```
print("4110E233 - 高瑞夫")

x = "wuzzup"
y = 30

print(bool(x))
print(bool(y))
```



```
4110E233 - 高瑞夫
True
True
```

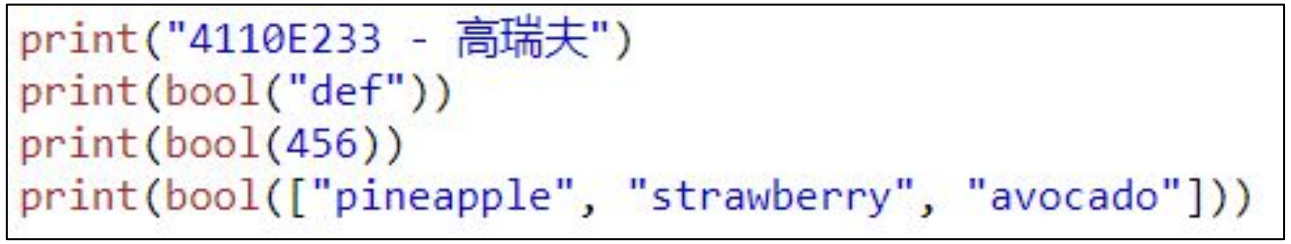
● Most Values are True

Almost any value is evaluated to True if it has some sort of content.

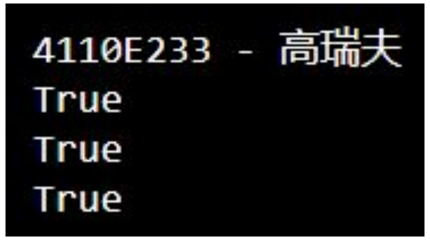
Any string is True, except empty strings.

Any number is True, except 0.

Any list, tuple, set, and dictionary are True, except empty ones.



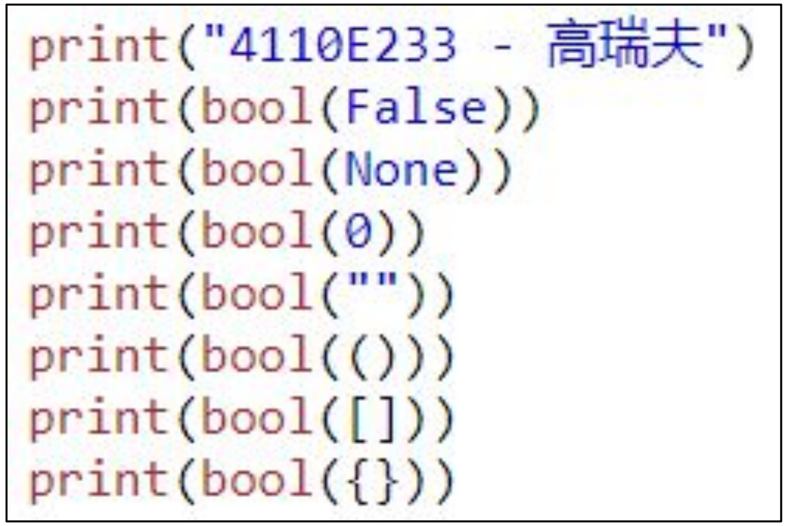
```
print("4110E233 - 高瑞夫")
print(bool("def"))
print(bool(456))
print(bool(["pineapple", "strawberry", "avocado"]))
```



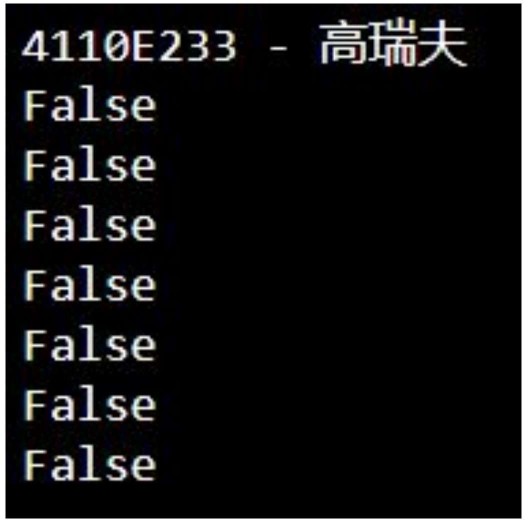
```
4110E233 - 高瑞夫
True
True
True
```

● Some Values are False

In fact, there are not many values that evaluate to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

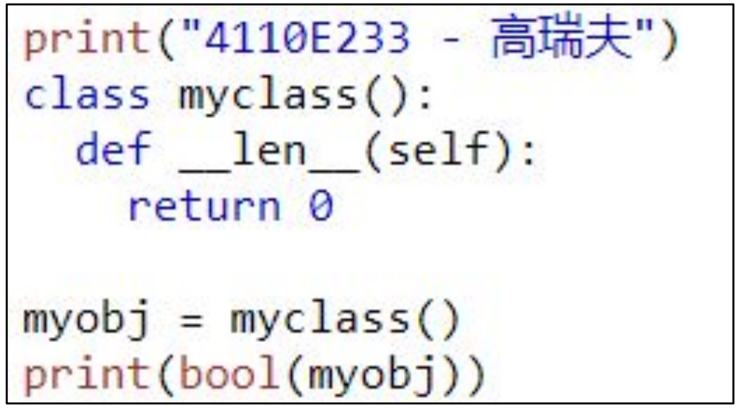


```
print("4110E233 - 高瑞夫")
print(bool(False))
print(bool(None))
print(bool(0))
print(bool(""))
print(bool(()))
print(bool([]))
print(bool({}))
```

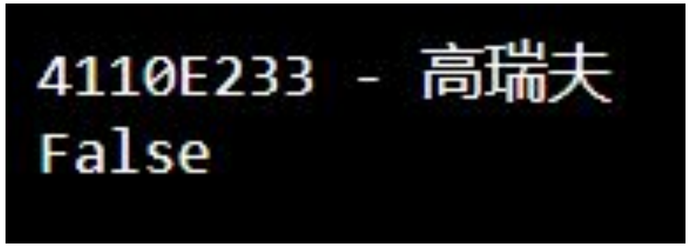


```
4110E233 - 高瑞夫
False
False
False
False
False
False
False
```

One more value, or object in this case, evaluates to False, and that is if you have an object that is made from a class with a __len__ function that returns 0 or False:



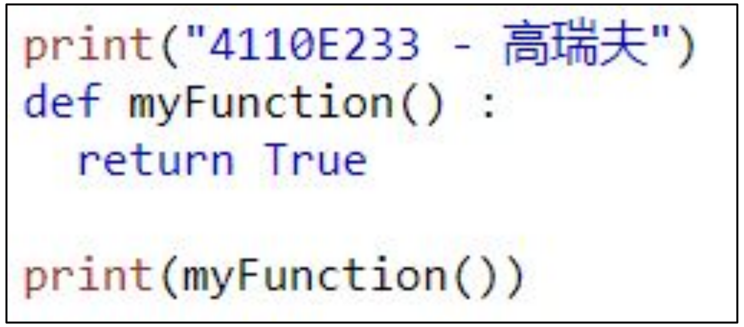
```
print("4110E233 - 高瑞夫")
class myclass():
  def __len__(self):
    return 0

myobj = myclass()
print(bool(myobj))
```



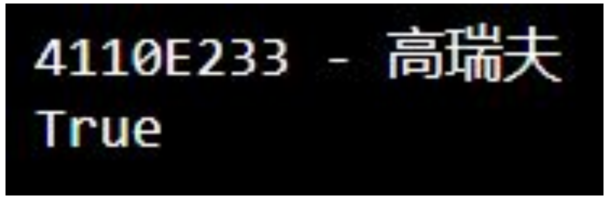
```
4110E233 - 高瑞夫
False
```

● Functions can Return a Boolean

You can create functions that returns a Boolean Value:

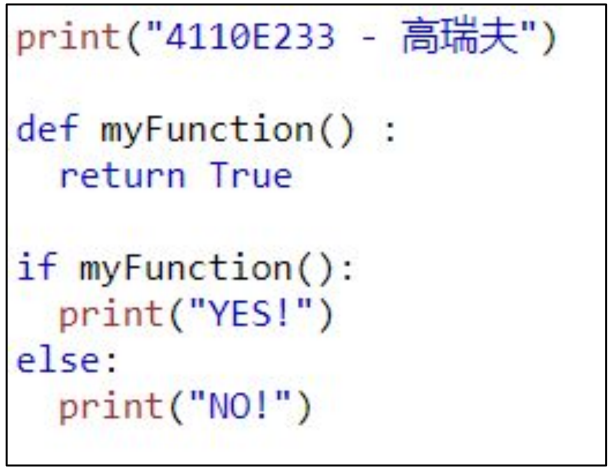


```
print("4110E233 - 高瑞夫")
def myFunction() :
  return True

print(myFunction())
```

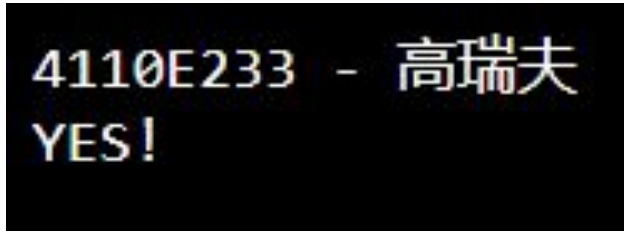


```
4110E233 - 高瑞夫
True
```

You can execute code based on the Boolean answer of a function:

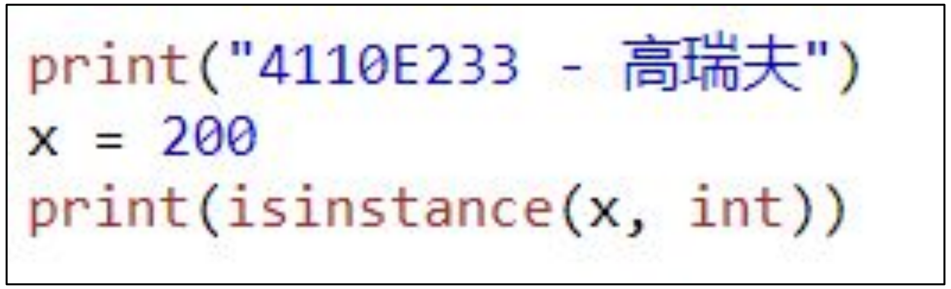


```
print("4110E233 - 高瑞夫")

def myFunction() :
  return True

if myFunction():
  print("YES!")
else:
  print("NO!")
```

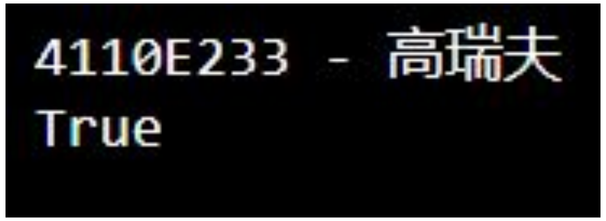


```
4110E233 - 高瑞夫
YES!
```

Python also has many built-in functions that return a boolean value, like the isinstance() function, which can be used to determine if an object is of a certain data type:



```
print("4110E233 - 高瑞夫")
x = 200
print(isinstance(x, int))
```



```
4110E233 - 高瑞夫
True
```

PYTHON OPERATION

CONTENT

CONTENT

CONTENT

- Nested Dictionaries
- Dictionary Methods

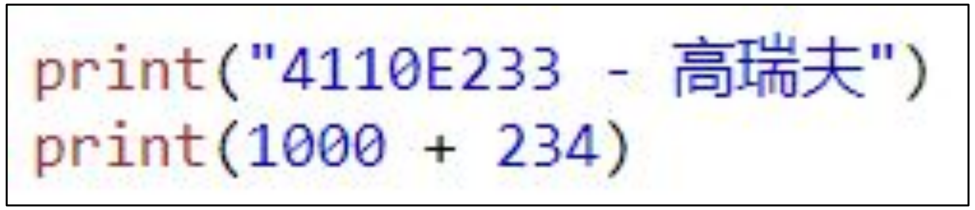
PYTHON OPERATORS

Operators

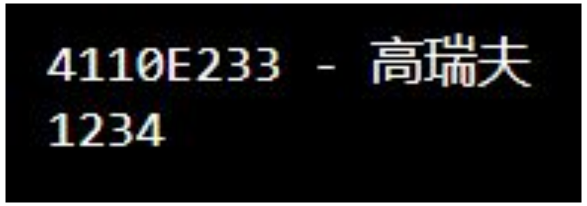
Python Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + operator to add together two values:



```
print("4110E233 - 高瑞夫")
print(1000 + 234)
```



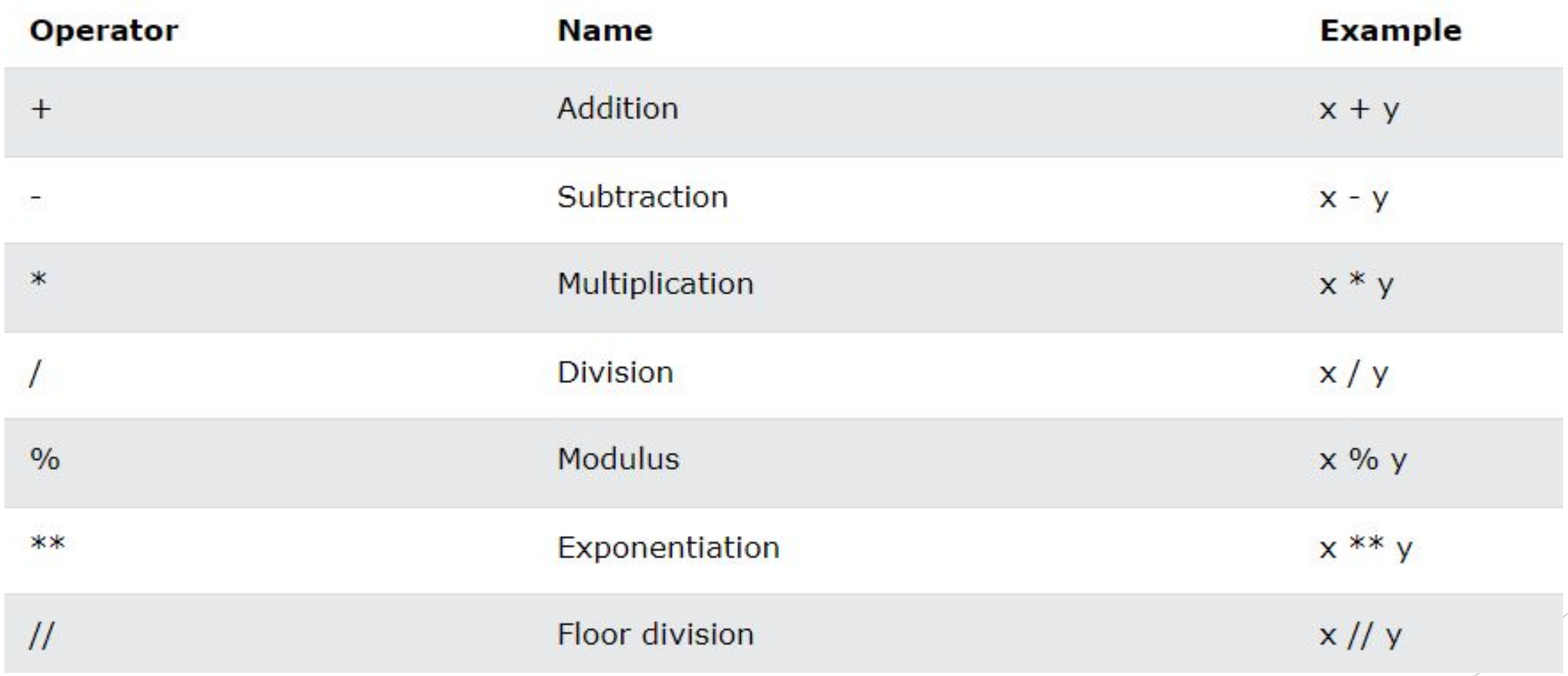
```
4110E233 - 高瑞夫
1234
```

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

● Python Arithmetic Operators

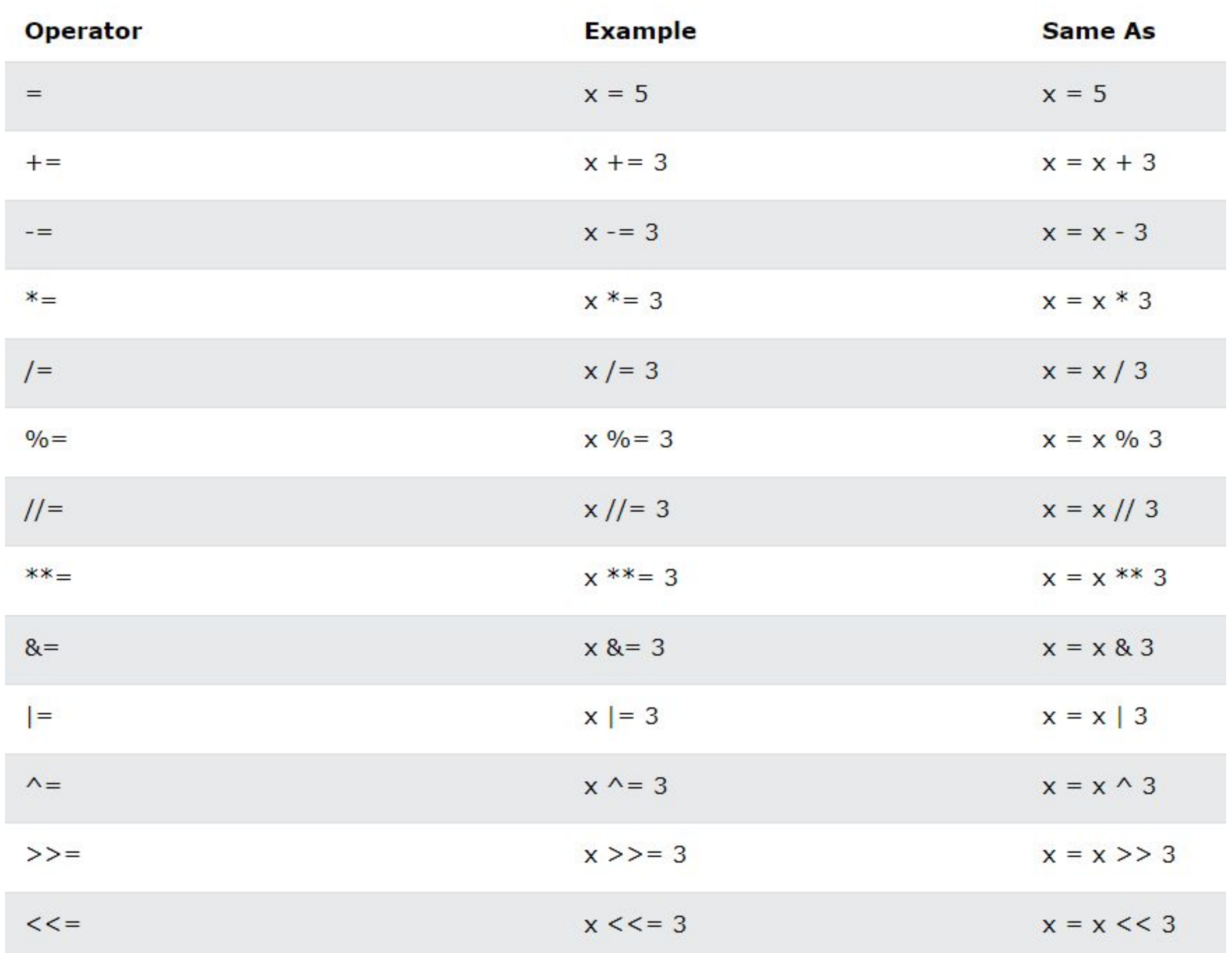
Arithmetic operators are used with numeric values to perform common mathematical operations:



Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Assignment Operators

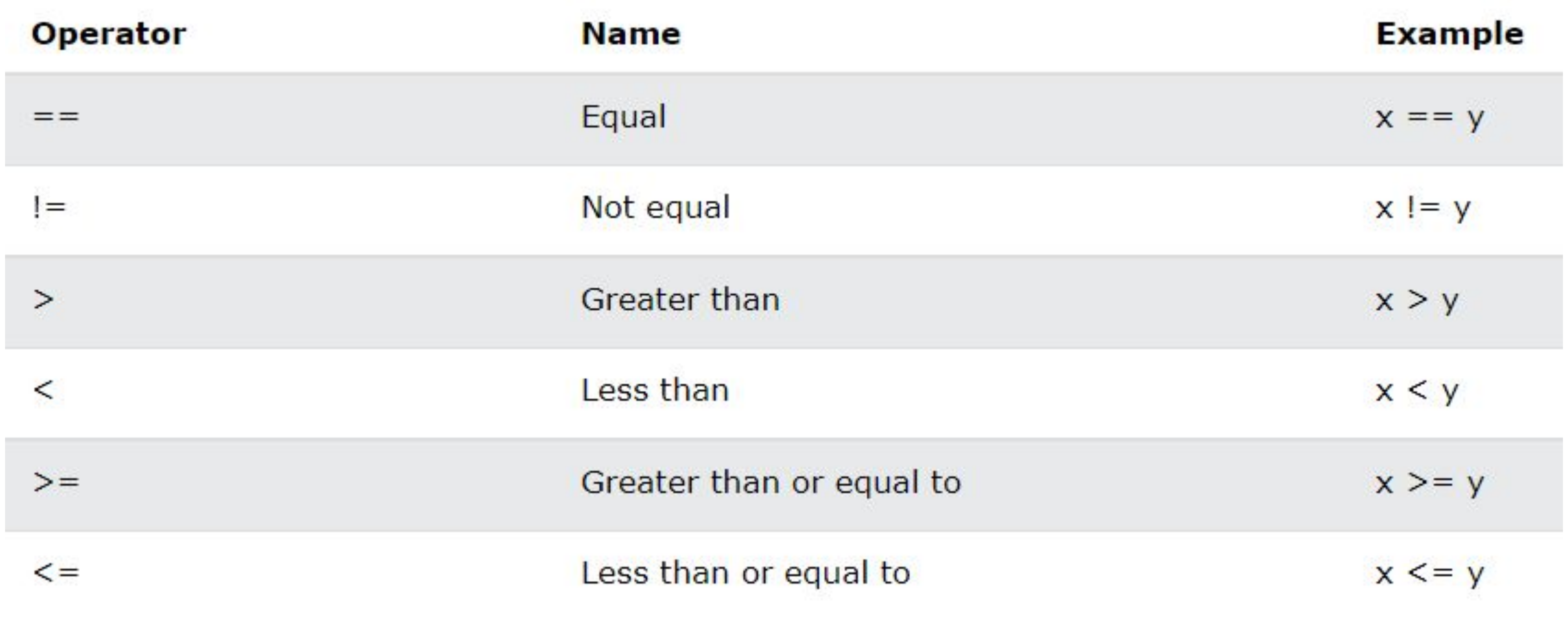
Assignment operators are used to assign values to variables:



Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

● Python Comparison Operators

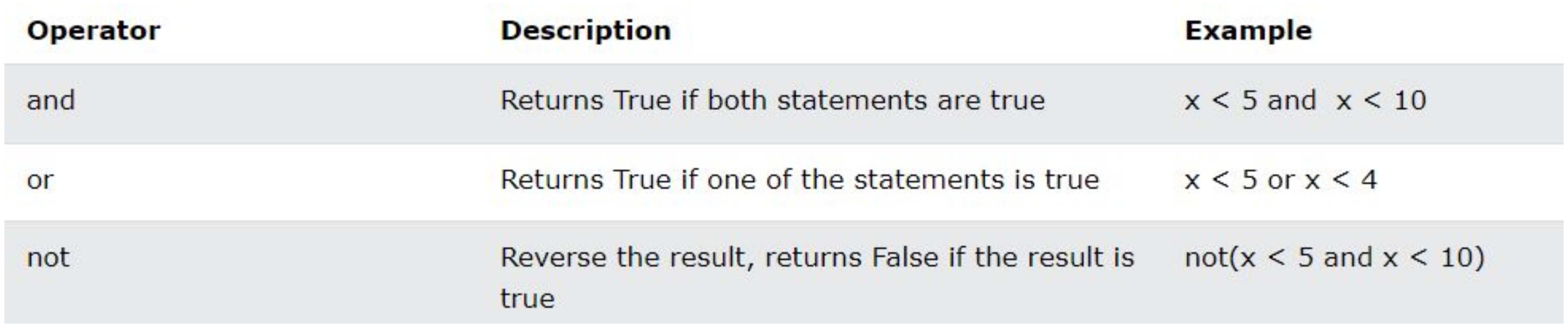
Comparison operators are used to compare two values:



Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

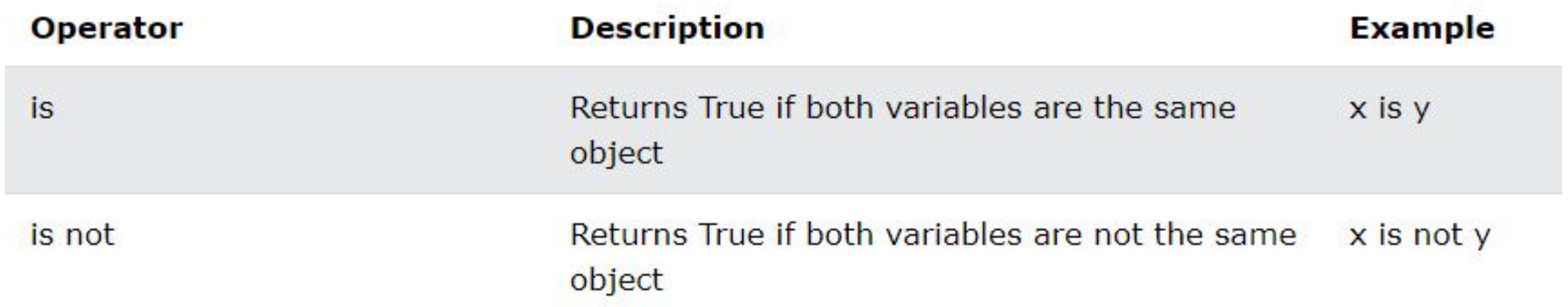
Logical operators are used to combine conditional statements:



Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

● Python Identity Operators

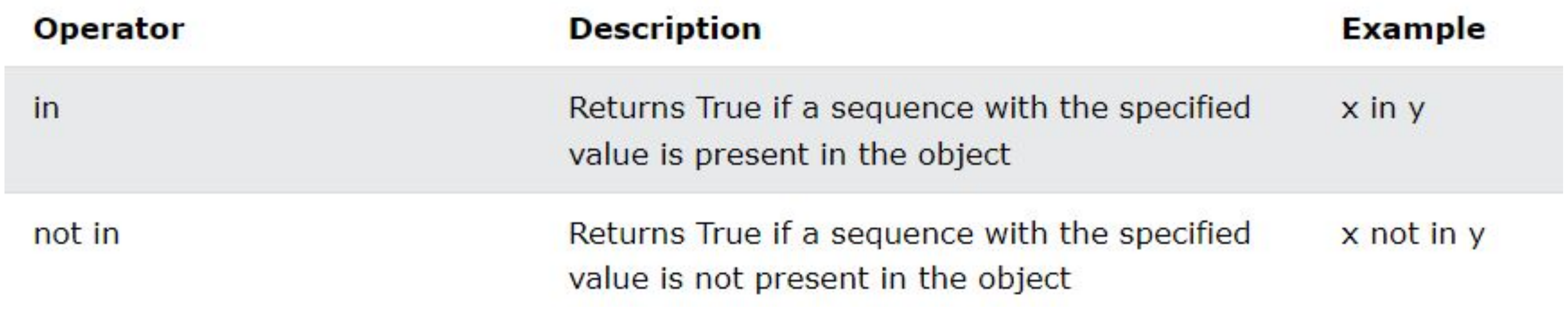
Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:



Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

● Python Membership Operators

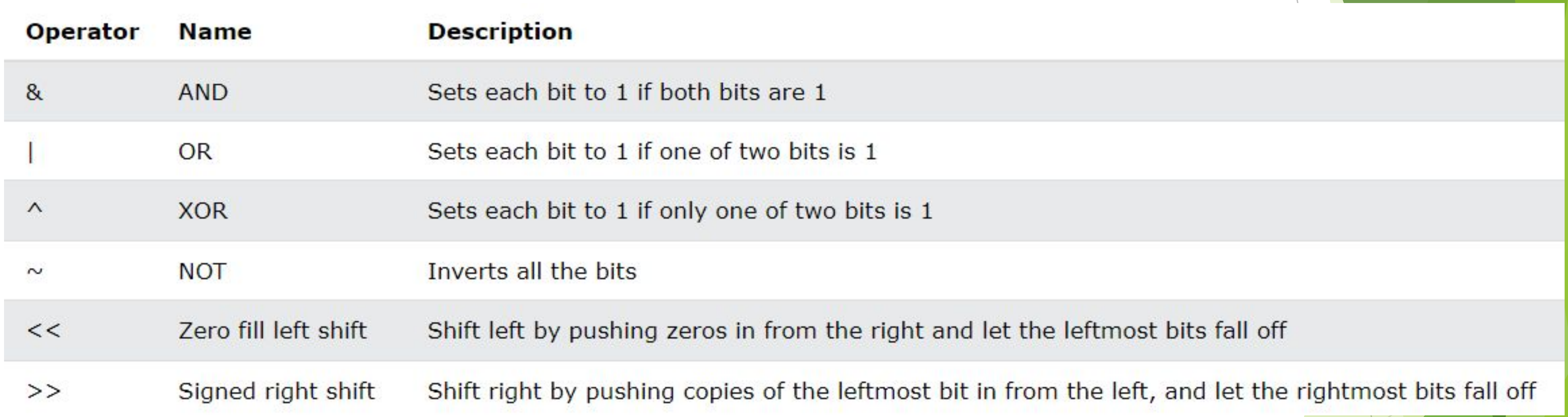
Membership operators are used to test if a sequence is presented in an object:



Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

● Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:



Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

PYTHON OPERATORS

Python Lists

Python Lists

https://github.com/reeeyyybb/cs2022/blob/main/python/1012.md

PYTHON OPERATORS

Python Dictionaries

Python Dictionaries

https://github.com/reeeyyybb/cs2022/blob/main/python/0914.md to be continued...