

# MY JOURNAL TO PYTHON

4110E210

MY DEAR GREAT TEACHER



- Python
- Input and Output : input() and print
- Data Types : numeric, string, list, dict.
- Operation ON data type:
- CONTROLS : IF- | IF –ELSE IF | -F-ELSE
- LOOP : FOR | WHILE | RANGE () | BREAK |  
CONTINUE
- FUNCTION
  - ① PARAMETERS (ARGUMENTS)
  - ② RECURSIVE FUNCTION
  - ③ LAMBA FUNCTION

# AGENDA



# 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.

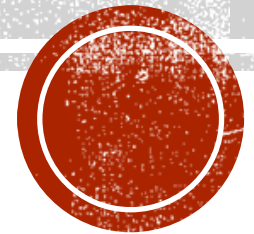
## Example



The screenshot shows a code editor interface. At the top, there are two tabs: '+ 程式碼' (Code) and '+ 文字' (Text). The 'Code' tab is active. Below the tabs, there is a code editor area. On the left side of the editor, there is a green checkmark and the text '0 秒' (0 seconds). The code editor contains a single line of Python code: `print("Hello, Fighter!")`. Below the code, the output of the code is displayed: 'Hello, Fighter!'.



# PYTHON SYNTAX



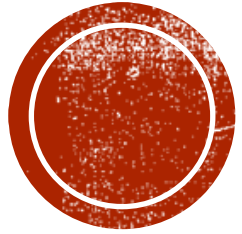
- Python syntax can be executed by writing directly in the Command Line:

```
>>> print("Hello, World!")  
Hello, World!
```

Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

```
C:\Users\Your Name>python myfile.py
```





# PYTHON INDENTATION

Indentation refers to the spaces at the beginning of a code line.

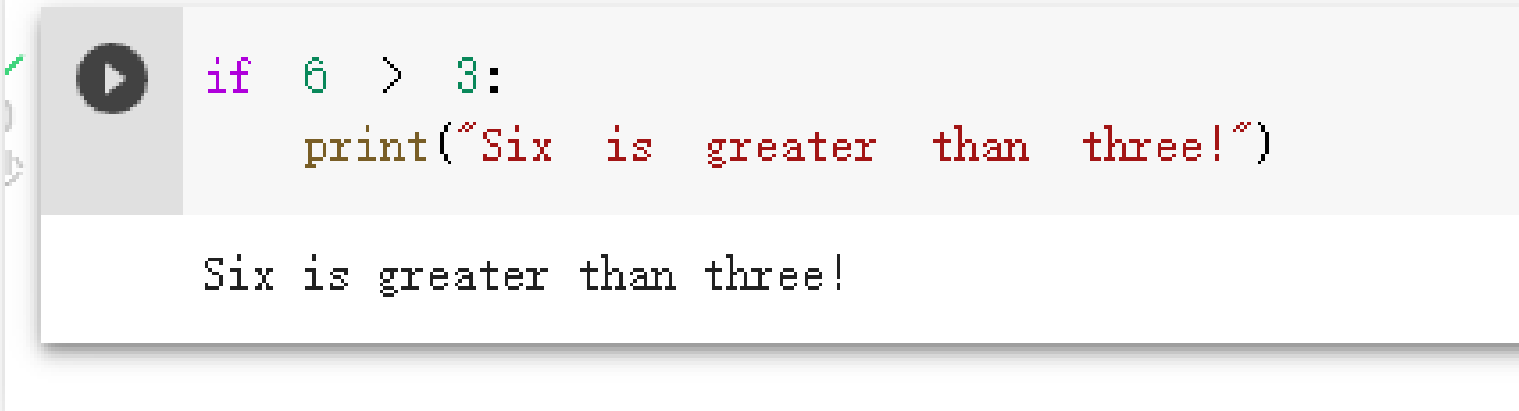
Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.



Python uses indentation to indicate a block of code.

## Example

A screenshot of a Python code editor. On the left, there is a vertical toolbar with icons for running, saving, and undo/redo. The code is as follows:

```
if 6 > 3:  
    print("Six is greater than three!")
```

The code is color-coded: 'if' is purple, '6' is green, '>' is blue, '3' is green, 'print' is brown, and the string is red. The code is indented to show a block.

`if 6 > 3:`  
 `print("Six is greater than three!")`

Six is greater than three!





Python will give you an error if you skip the indentation:

## Example

0  
沙



```
if 6 > 3:  
print("Six is greater than three!")
```

File "<ipython-input-10-34e4e31cc3cb>", line 2  
print("Six is greater than three!")  
^

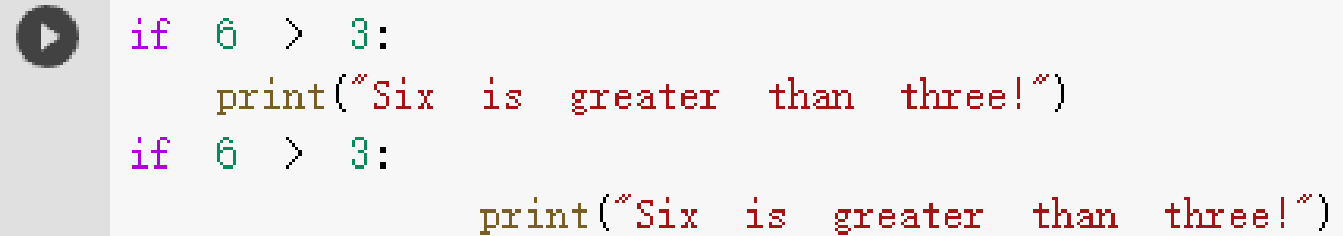
IndentationError: expected an indented block

SEARCH STACK OVERFLOW



The number of spaces is up to you as a programmer, the most common use is four, but it has to be at least one.

## Example

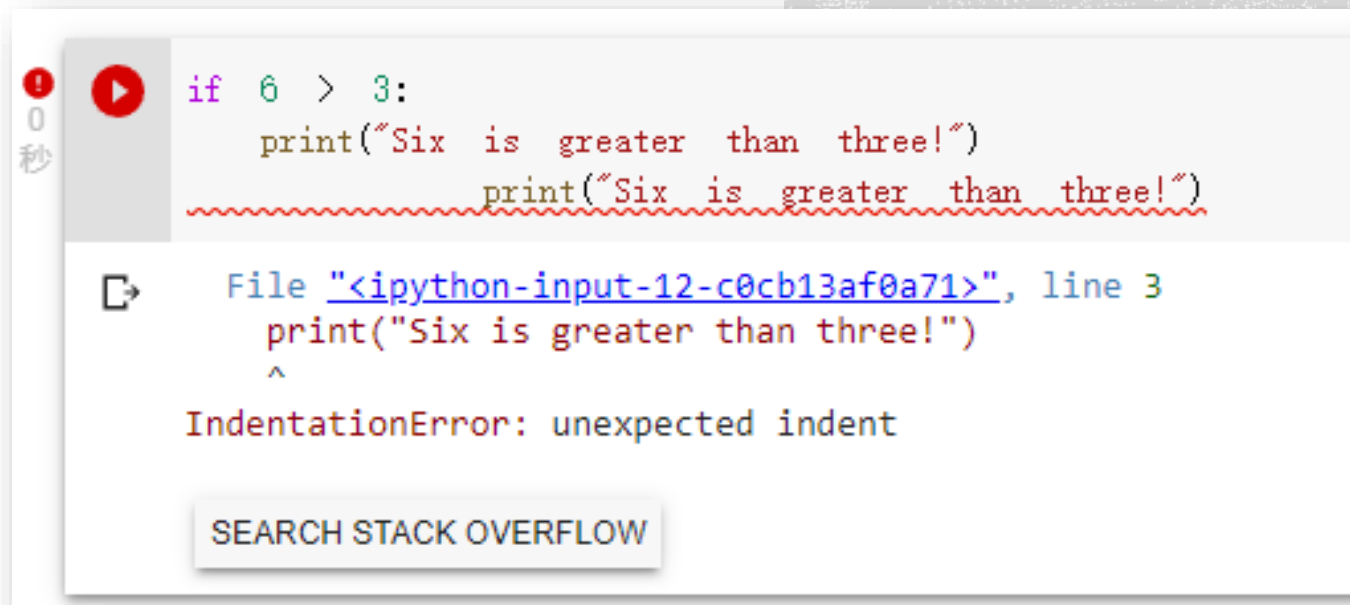


```
if 6 > 3:  
    print("Six is greater than three!")  
if 6 > 3:  
        print("Six is greater than three!")
```

```
Six is greater than three!  
Six is greater than three!
```



You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

A screenshot of a Jupyter Notebook interface showing a Python code block with an indentation error. The code block has a red play button icon and a timer showing '0 秒'. The code is:

```
if 6 > 3:  
    print("Six is greater than three!")  
    print("Six is greater than three!")
```

The second line of the code block is underlined with a red wavy line, indicating an error. Below the code block, the error message is displayed: 'File "<ipython-input-12-c0cb13af0a71>", line 3' followed by 'print("Six is greater than three!")' with a caret under the first space, and 'IndentationError: unexpected indent'. At the bottom of the error message box is a button that says 'SEARCH STACK OVERFLOW'.

```
if 6 > 3:  
    print("Six is greater than three!")  
    print("Six is greater than three!")
```

File "<ipython-input-12-c0cb13af0a71>", line 3  
 print("Six is greater than three!")  
 ^  
IndentationError: unexpected indent

SEARCH STACK OVERFLOW

In Python, variables are created when you assign a value to it:

## Example

:



```
x = 5
```

```
y = "Hello, Fighter!"
```



Python has no command for declaring a variable.

## Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a `#`, and Python will render the rest of the line as a comment:

### Example



0s

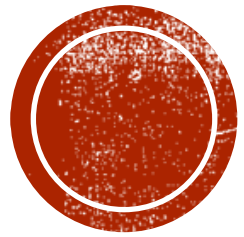


```
#This is a comment.  
print("Hello, Fighter!")
```



```
Hello, Fighter!
```



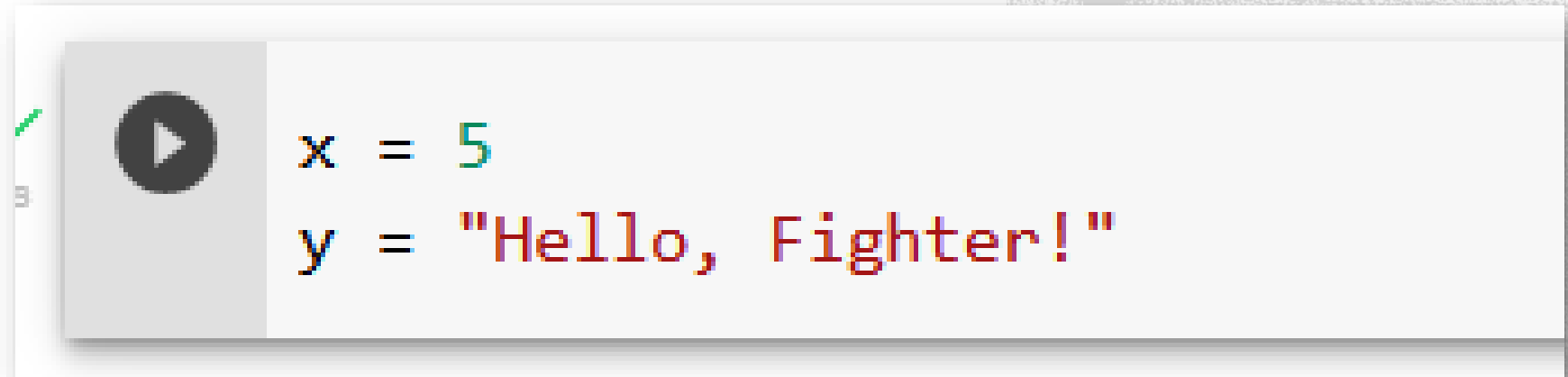


# PYTHON VARIABLES

In Python, variables are created when you assign a value to it:

## Example

Variables in Python:

A code editor snippet with a light gray background. On the left, there is a dark gray play button icon. To its right, two lines of Python code are displayed: `x = 5` and `y = "Hello, Fighter!"`. The code is color-coded: `x` is blue, `=` is black, `5` is green, `y` is blue, `=` is black, and the string `"Hello, Fighter!"` is red. A green checkmark is visible on the far left edge of the code block.

```
x = 5  
y = "Hello, Fighter!"
```

Python has no command for declaring a variable.

You will learn more about variables in the Python Variables chapter.





## Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment:

## Example



```
#This is a comment.  
print("Hello, Fighter!")
```

Hello, Fighter!



Comments can be used to explain Python code.  
Comments can be used to make the code more readable.  
Comments can be used to prevent execution when testing code.

## Creating a Comment

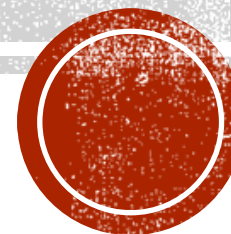
Comments starts with a #, and Python will ignore them:

A code editor snippet with a light gray background. On the left, there is a small gray square containing a black play button icon. To the right of this icon, the text is as follows: the first line is a comment '#This is a comment' in green, and the second line is a print statement 'print("Hello, Fihgter!")' in brown. Below this code, on a white background, the output 'Hello, Fihgter!' is displayed in a black monospaced font.

```
#This is a comment  
print("Hello, Fihgter!")  
  
Hello, Fihgter!
```



# PYTHON COMMENTS



Comments can be used to explain Python code.

Comments can be used to make the code more readable.

Comments can be used to prevent execution when testing code.

## Creating a Comment

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




```
#This is a comment  
print("Hello, Fighter!")
```

```
Hello, Fighter!
```



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

## Example



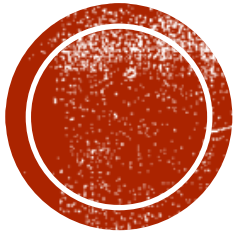
```
print("Hello, Fighter!") #This is a comment
```



```
Hello, Fighter!
```



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



```
#print("Hello, Fighter!")  
print("Cheers, LOve!")
```

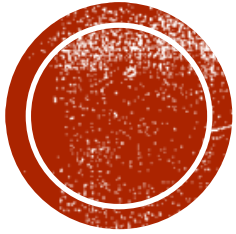


```
Cheers, LOve!
```



# Multi Line Comments

Python does not really have a syntax for multi line comments.  
To add a multiline comment you could insert a `#` for each line:



## Example



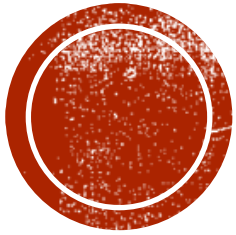
```
#This is a comment  
#written in  
#more than just one line  
print("Hello, Fighter!")
```

Hello, Fighter!



Or, not quite as intended, you can use a multiline string.

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:



### Example

As long as the string is not assigned to a variable, Python will read the code, but then ignore it, and you have made a multiline comment.



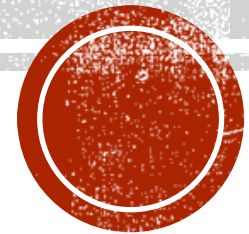
```
"""  
This is a comment  
written in  
more than just one line  
"""  
  
print("Hello, Fighter!")
```

☞ Hello, Fighter!

# PYTHON VARIABLES

## **Variables**

Variables are containers for storing data values.

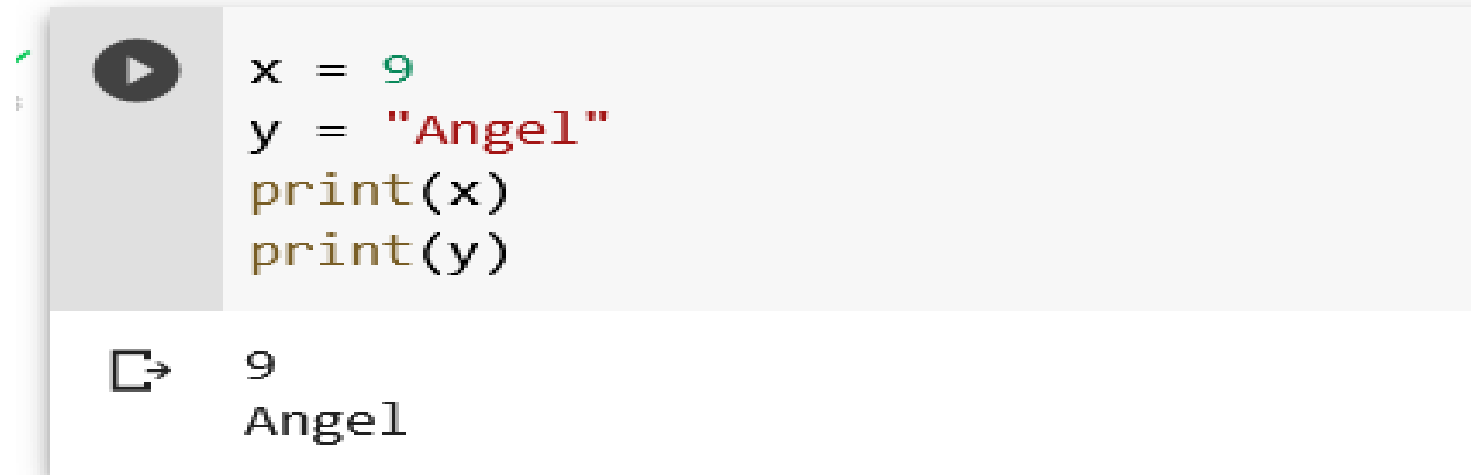


# Creating Variables

Python has no command for declaring a variable.

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

## Example

A screenshot of a code editor or terminal window showing a Python script being executed. The script consists of four lines: 'x = 9' (with '9' in green), 'y = "Angel"' (with 'Angel' in red), 'print(x)' (with 'print' in brown), and 'print(y)' (with 'print' in brown). To the left of the code is a play button icon. Below the code, the output is displayed: '9' on the first line and 'Angel' on the second line, preceded by a copy icon (a square with a right-pointing arrow).


```
x = 9
y = "Angel"
print(x)
print(y)
```

```
9
Angel
```



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

## Example

```
 x = 9          # x is of type int  
x = "Angel"    # x is now of type str  
print(x)
```

Angel



# Casting

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

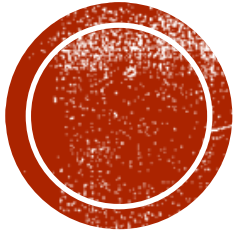
## Example



```
x = str(9)      # x will be '9'  
y = int(9)      # y will be 9  
z = float(9)    # z will be 9.0
```



# Python - Variable Names



## 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)



## Example

```
▶ myvar = "Angel"  
  my_var = "Angel"  
  _my_var = "Angel"  
  myVar = "Angel"  
  MYVAR = "Angel"  
  myvar2 = "Angel"
```

## Example

Illegal variable names:

```
! 0s ▶ 2myvar = "Angel"  
      my-var = "Angel"  
      my var = "Angel"
```

File "<ipython-input-21-4f0bd1ea342e>", line 1  
 2myvar = "Angel"  
 ^  
SyntaxError: invalid syntax

SEARCH STACK OVERFLOW





# 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:

Each word starts with a capital letter:

A code editor snippet showing a variable assignment. On the left, there is a green checkmark and a play button icon. The code is `myVariableName = "Angel"`. The variable name `myVariableName` is in black, and the string value `"Angel"` is in red.

```
myVariableName = "Angel"
```



## Pascal Case

Each word starts with a capital letter:



```
MyVariableName = "Angel"
```

## Snake Case

Each word is separated by an underscore character:



0s

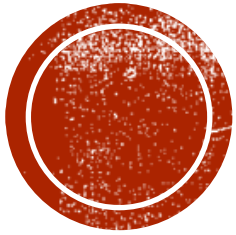


```
my_variable_name = "John"
```



# Python Variables - Assign Multiple Values

## Many Values to Multiple Variables



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

### Example

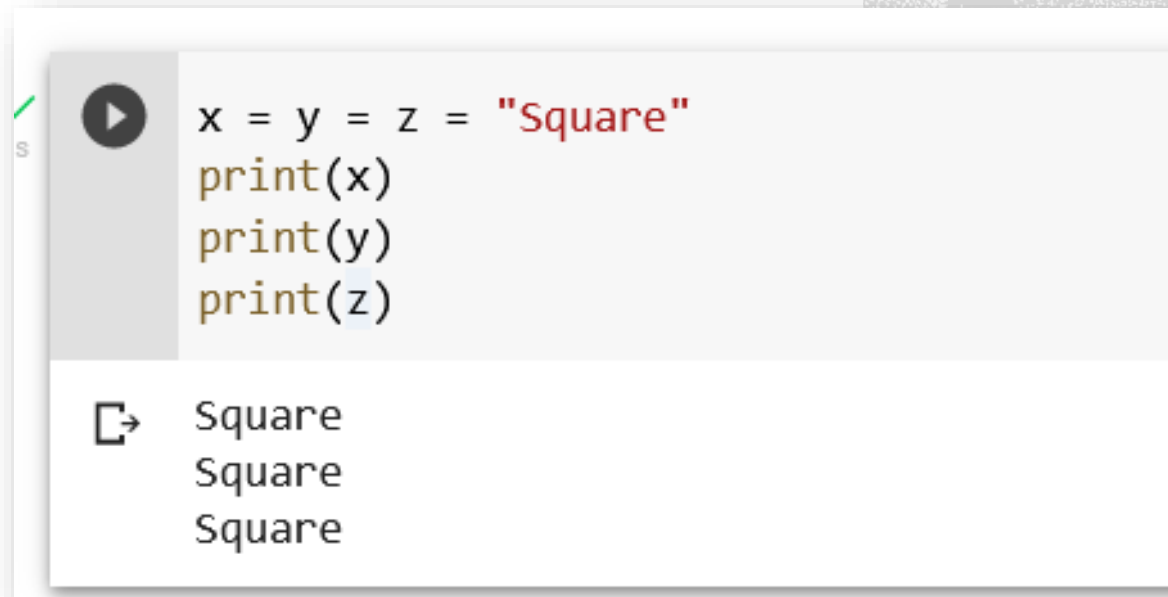


```
x, y, z = "Square", "Circle", "Triangle"  
print(x)  
print(y)  
print(z)
```

# One Value to Multiple Variables

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

## Example

A screenshot of a Python code editor showing a code block with a play button icon. The code assigns the string "Square" to three variables x, y, and z, and then prints each variable. Below the code, the output shows the word "Square" printed three times, one on each line.

```
x = y = z = "Square"  
print(x)  
print(y)  
print(z)
```

Square  
Square  
Square



# 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*.

## Example



```
shapes = ["square", "circle", "triangle"]  
x, y, z = shapes  
print(x)  
print(y)  
print(z)
```

```
square  
circle  
triangle
```



# PYTHON - OUTPUT VARIABLES

## Output Variables

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

### Example



```
x = "Python is challenging"  
print(x)
```

Python is challenging



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

## Example

```
✓ 2s ▶ x = "Python"  
      y = "is"  
      z = "challenging"  
      print(x, y, z)  
  
Python is challenging
```

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

## Example

```
✓ 1s ▶ x = "Python "  
      y = "is "  
      z = "challenging"  
      print(x + y + z)  
  
📄 Python is challenging
```





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

### Example



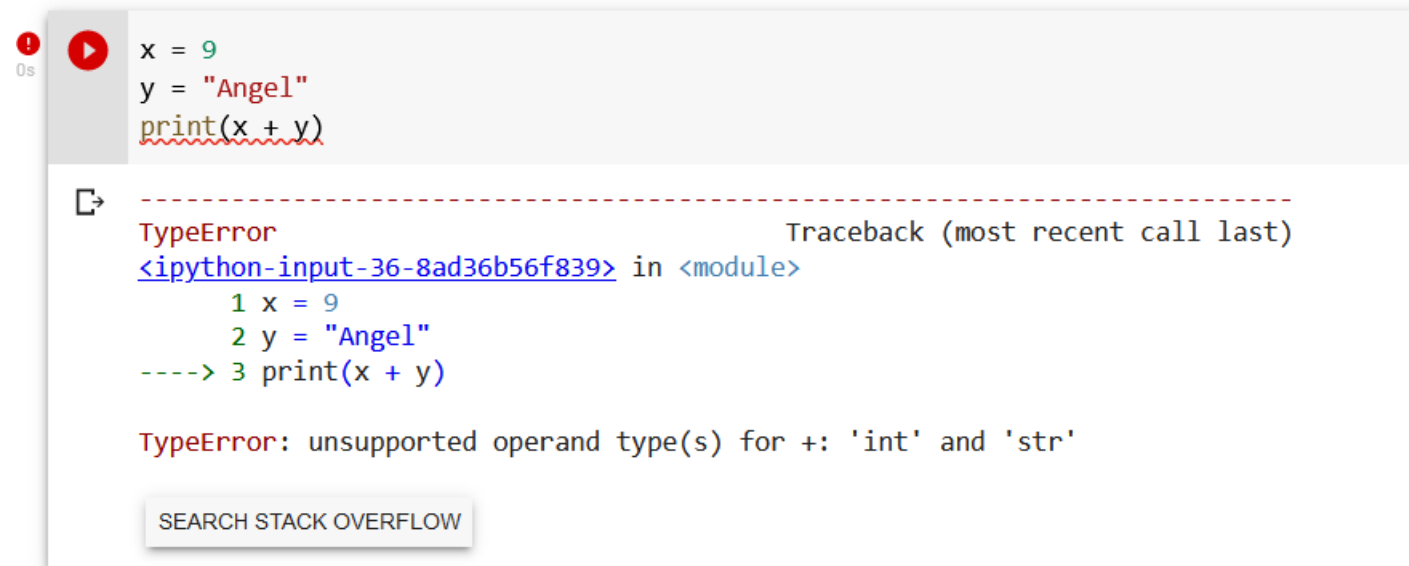
```
x = 9
y = 18
print(x + y)
```

27

The code is executed in a Jupyter-style interface. A green checkmark and a play button icon are visible on the left. The output of the print statement is 27.

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

### Example



```
x = 9
y = "Angel"
print(x + y)
```

-----  
**TypeError** Traceback (most recent call last)  
 <ipython-input-36-8ad36b56f839> in <module>  
 1 x = 9  
 2 y = "Angel"  
----> 3 print(x + y)  
**TypeError:** unsupported operand type(s) for +: 'int' and 'str'


SEARCH STACK OVERFLOW


The code is executed in a Jupyter-style interface. A red error icon and a play button icon are visible on the left. The output shows a stack trace for a TypeError, indicating that the '+' operator cannot be used between an integer and a string.



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

## Example

 0s



```
x = 9
y = "Angel"
print(x, y)
```

9 Angel



# Python - Global Variables

## 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.

## Example

A code editor snippet with a light gray background. On the left, there is a vertical bar with a green checkmark and the text 'is'. To the right of this bar is a play button icon. The code is as follows:

```
x = "challenging"

def myfunc():
    print("Python is " + x)

myfunc()
```

Below the code, there is a white box containing a copy icon and the output text: `Python is challenging`.

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.

## Example

Create a variable inside a function, with the same name as the global variable

```
✓  
15  x = "challenging"  
  
def myfunc():  
    x = "enthusiastic"  
    print("Python is " + x)  
  
myfunc()  
  
print("Python is " + x)  
  
📄 Python is enthusiastic  
Python is challenging
```



# 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.

## Example

If you use the global keyword, the variable belongs to the global scope:


```
✓  def myfunc():  
    global x  
    x = "enthusiastic"  
  
myfunc()  
  
print("Python is " + x)  
  
Python is enthusiastic
```



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

## Example

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

A code editor window with a light gray background. On the left, there is a vertical toolbar with a green checkmark at the top and a play button icon below it. The code is written in a syntax-highlighted style: keywords like 'def', 'global', and 'print' are in blue, and strings are in red. The code defines a function 'myfunc()' that uses the 'global' keyword to modify the variable 'x'. After calling the function, the value of 'x' is printed.

```
x = "challenging"

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

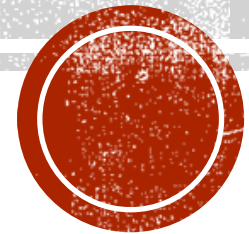
myfunc()

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

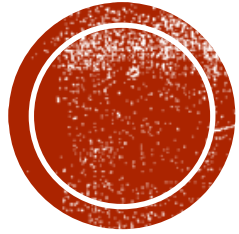
Python is enthusiastic



# PYTHON DATA TYPES







# BUILT-IN DATA TYPES

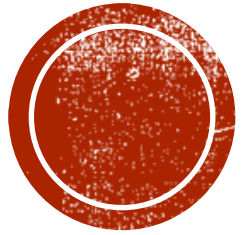
In programming, data type is an important concept.

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





# Getting the Data Type

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

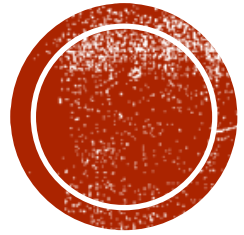
## Example

Print the data type of the variable x:



```
x = 8  
print(type(x))
```

```
<class 'int'>
```



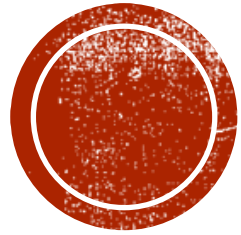
# SETTING THE DATA TYPE



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

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





# **SETTING THE SPECIFIC DATA TYPE**

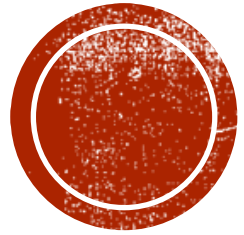


If you want to specify the data type, you can use the following constructor functions:

Example	Data Type	Try it
<code>x = str("Hello World")</code>	str	<a href="#">Try it »</a>
<code>x = int(20)</code>	int	<a href="#">Try it »</a>
<code>x = float(20.5)</code>	float	<a href="#">Try it »</a>
<code>x = complex(1j)</code>	complex	<a href="#">Try it »</a>
<code>x = list(("apple", "banana", "cherry"))</code>	list	<a href="#">Try it »</a>
<code>x = tuple(("apple", "banana", "cherry"))</code>	tuple	<a href="#">Try it »</a>
<code>x = range(6)</code>	range	<a href="#">Try it »</a>
<code>x = dict(name="John", age=36)</code>	dict	<a href="#">Try it »</a>
<code>x = set(("apple", "banana", "cherry"))</code>	set	<a href="#">Try it »</a>
<code>x = frozenset(("apple", "banana", "cherry"))</code>	frozenset	<a href="#">Try it »</a>
<code>x = bool(5)</code>	bool	<a href="#">Try it »</a>
<code>x = bytes(5)</code>	bytes	<a href="#">Try it »</a>
<code>x = bytearray(5)</code>	bytearray	<a href="#">Try it »</a>
<code>x = memoryview(bytes(5))</code>	memoryview	<a href="#">Try it »</a>







# 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:

**Example**

```
x = 1 # int  
y = 2.8 # float  
z = 1j # complex
```

To verify the type of any object in Python, use the `type()` function:

```
✓ 1s ▶ print(type(x))  
print(type(y))  
print(type(z))  
  
<class 'int'>  
<class 'float'>  
<class 'complex'>
```



# Int

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

## Example

Integers

```
x = 3  
y = 12192002121922  
z = -54321
```

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

```
<class 'int'>  
<class 'int'>  
<class 'int'>
```



## Float

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

### Example

```
✓ 0s ▶ x = 1.19  
      y = 1.9  
      z = -22.19  
  
      print(type(x))  
      print(type(y))  
      print(type(z))  
  
      <class 'float'>  
      <class 'float'>  
      <class 'float'>
```



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

### Example

Floats:

```
x = 19e2  
y = 12E4  
z = -97.7e129
```

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

```
<class 'float'>  
<class 'float'>  
<class 'float'>
```



## Complex

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

## Example

Complex:

```
x = 9+2j
y = 2j
z = -2j

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

```
<class 'complex'>
<class 'complex'>
<class 'complex'>
```



# Type Conversion

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

## Example

Convert from one type to another:

```

x = 2    # int
y = 1.9  # float
z = 9j   # complex

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

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

#convert from int to complex:
c = complex(x)

print(a)
print(b)
print(c)

print(type(a))
print(type(b))
print(type(c))

```

```

2.0
1
(2+0j)
<class 'float'>
<class 'int'>
<class 'complex'>

```





## Random Number

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:

### Example

Import the random module, and display a random number between 1 and 9:

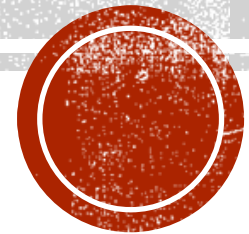
A code execution snippet showing a Python program. On the left, there is a green checkmark and a play button icon. The code consists of two lines: 'import random' and 'print(random.randrange(2, 19))'. Below the code, the output '11' is displayed.

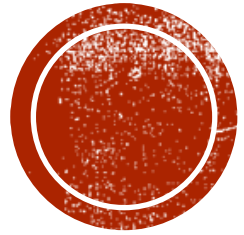
```
import random  
print(random.randrange(2, 19))
```

11



# 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:

```
✓ ▶ x = int(1)    # x will be 1  
    y = int(2.9) # y will be 2  
    z = int("3") # z will be 3
```

Floats:

```
✓ ▶ x = float(1)    # x will be 1.0  
    y = float(2.9)  # y will be 2.9  
    z = float("3")  # z will be 3.0  
    w = float("4.1") # w will be 4.2
```

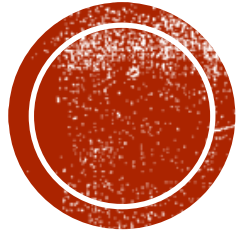
Strings:

```
✓ ▶ x = str("s2") # x will be 's2'  
    y = str(3)    # y will be '3'  
    z = str(4.0)  # z will be '4.0'
```



# 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:

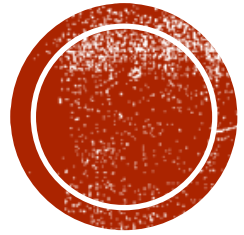
## Example

A screenshot of a Python code execution environment. It shows a code editor with two lines of Python code: `print("Hello")` and `print('Hello')`. The first line uses double quotes and the second uses single quotes. Below the code editor, the output is displayed as two lines of text: `Hello` and `Hello`. To the left of the code editor, there is a green checkmark and a play button icon. To the left of the output, there is a copy icon (a square with an arrow pointing right).

```
print("Hello")  
print('Hello')
```

```
Hello  
Hello
```





# **ASSIGN STRING TO A VARIABLE**



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

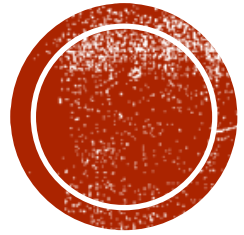
### Example



```
a = "Hello"  
print(a)
```

Hello





# MULTILINE STRINGS



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

## Example

You can use three double quotes:



```
a = """Dont just take but give,  
Dont just see but feel,  
Dont just dream but do."""  
print(a)
```



```
Dont just take but give,  
Dont just see but feel,  
Dont just dream but do.
```



Or three single quotes:

## Example

✓  
0s



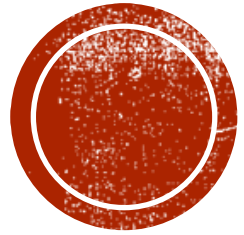
```
a = '''Dont just take but give,  
Dont just see but feel,  
Dont just dream but do.'''  
print(a)
```



```
Dont just take but give,  
Dont just see but feel,  
Dont just dream but do.
```







**STRINGS ARE ARRAYS**



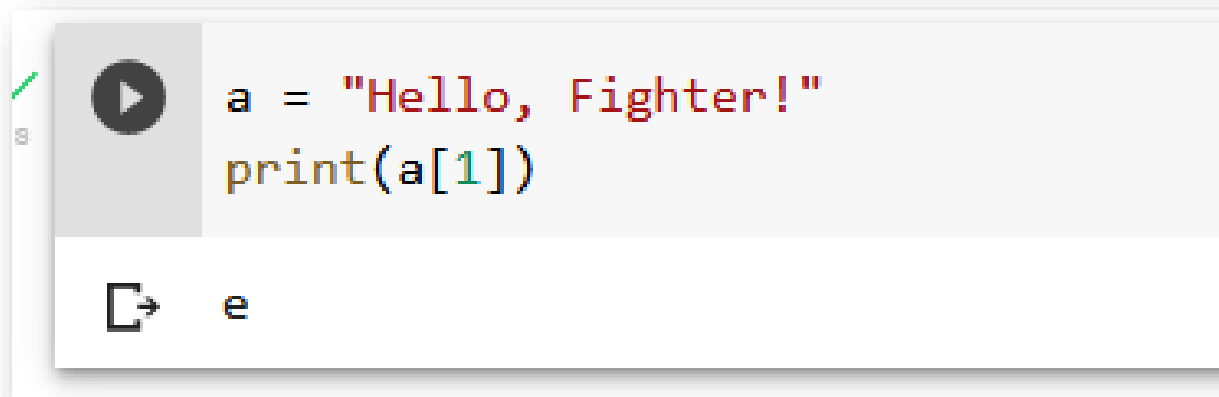
Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

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.

### Example

Get the character at position 1 (remember that the first character has the position 0):

A screenshot of a Python REPL (Read-Eval-Print Loop) window. The window has a light gray background. On the left side, there is a green checkmark icon and a play button icon. The main area shows two lines of code: `a = "Hello, Fighter!"` and `print(a[1])`. Below the code, there is a prompt `[>]` followed by the output `e`.

```
a = "Hello, Fighter!"  
print(a[1])  
[> e
```





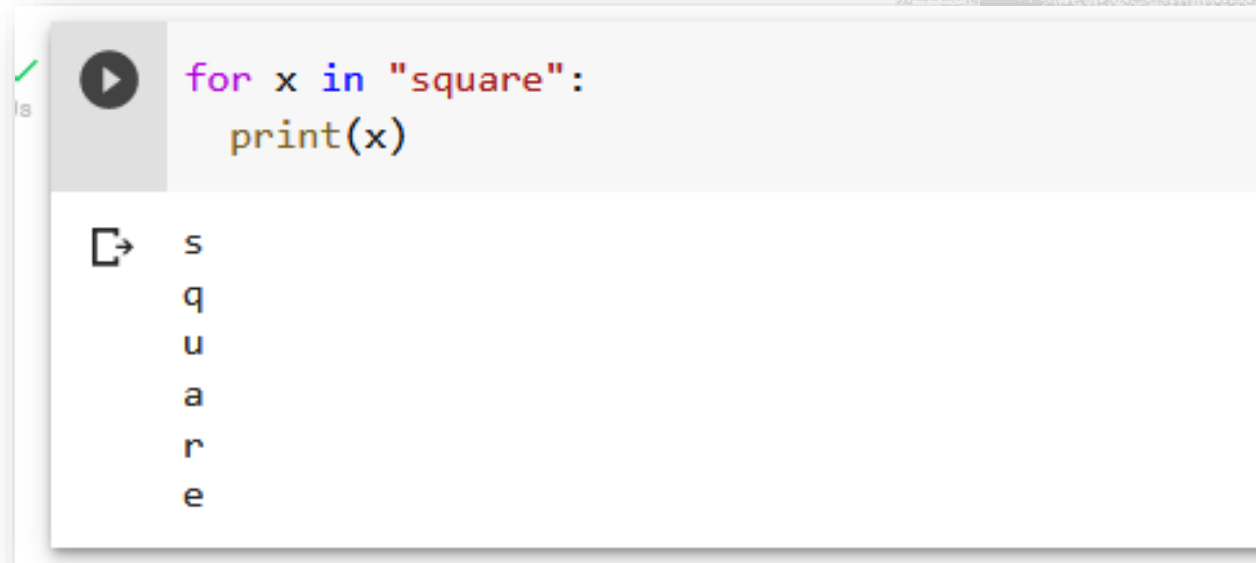
# **LOOPING THROUGH A STRING**



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

## Example

Loop through the letters in the word “square”:

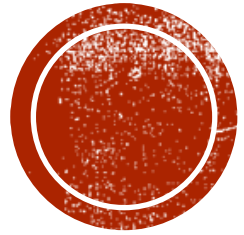


```
for x in "square":  
    print(x)
```

The output of the code is:

```
s  
q  
u  
a  
r  
e
```





# STRING LENGTH



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

## Example

The len() function returns the length of a string:

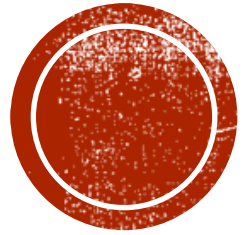


```
a = "Hello, Fighter!"  
print(len(a))
```



```
15
```



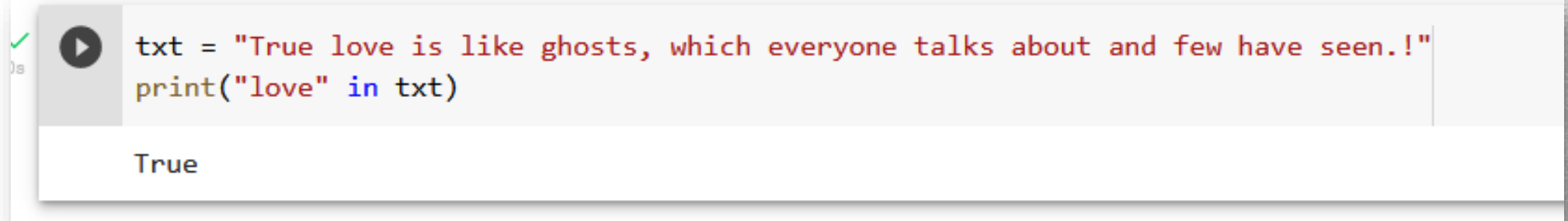


# CHECK STRING

To check if a certain phrase or character is present in a string, we can use the keyword `in`.

## Example

Check if "love" is present in the following text:

A screenshot of a code editor or terminal window. On the left, there is a green checkmark and a play button icon. The main area contains two lines of Python code: `txt = "True love is like ghosts, which everyone talks about and few have seen.!"` and `print("love" in txt)`. Below the code, the output `True` is displayed.

```
txt = "True love is like ghosts, which everyone talks about and few have seen.!"  
print("love" in txt)  
  
True
```





Use it in an if statement:

## Example

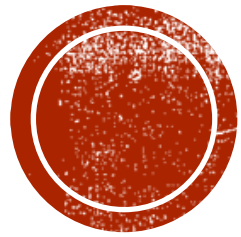
Print only if "love" is present:



```
txt = "True love is like ghosts, which everyone talks about and few have seen.!"  
if "love" in txt:  
    print("Yes, 'love' is present.")
```

Yes, 'love' is present.





# CHECK IF NOT

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



## Example

Check if “**friendship**” is NOT present in the following text:

```
txt = "True love is like ghosts, which everyone talks about and few have seen.!"  
print("friendship" not in txt)
```

True



Use it in an **if** statement:

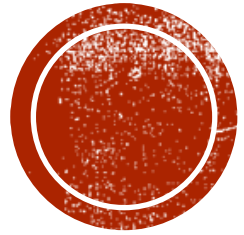
## Example

print only if “friendship” is NOT present:

```
txt = "True love is like ghosts, which everyone talks about and few have seen.!"  
if "friendship" not in txt:  
    print("No, 'friendship' is NOT present.")
```

No, 'friendship' is NOT present.





# **PYTHON - SLICING STRINGS**



## Example

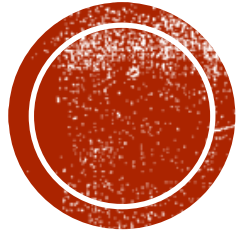
Get the characters from position 2 to position 5 (not included):



```
b = "Hello, fighter!"  
print(b[2:5])
```

```
llo
```

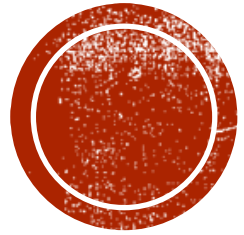




# SLICING

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.



# **SLICE FROM THE START**

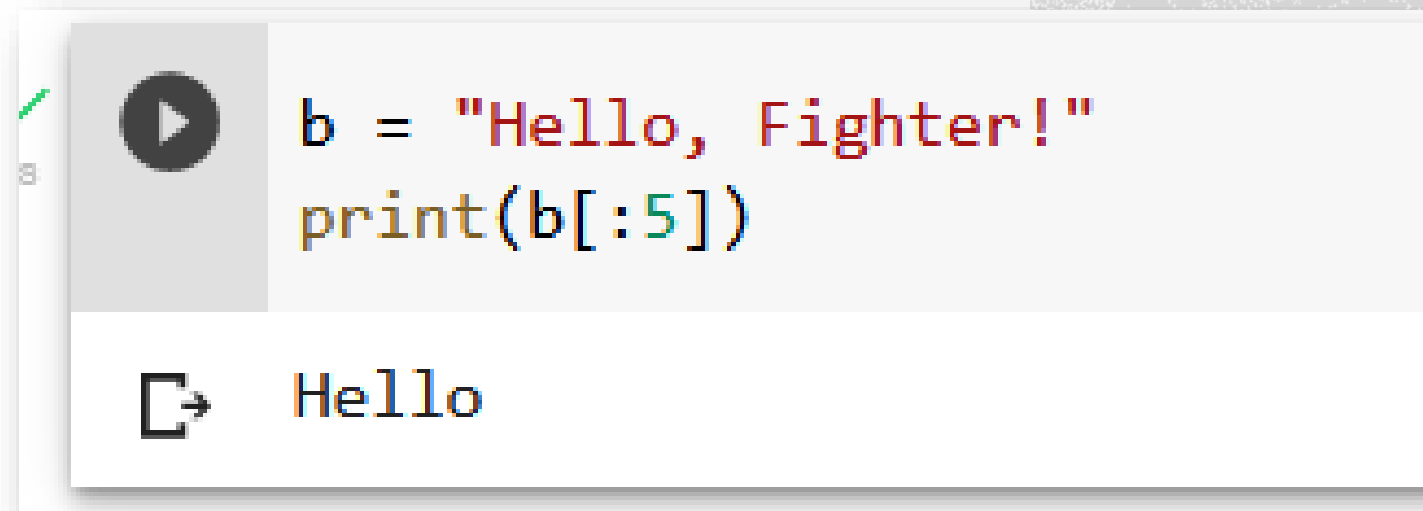
By leaving out the start index, the range will start at the first character:





## Example

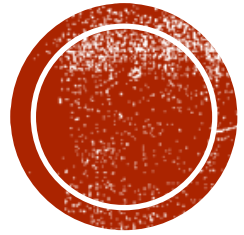
Get the characters from the start to position 5 (not included):

A screenshot of a code execution environment. It features a light gray background with a dark gray play button icon on the left. The code is written in a monospaced font with syntax highlighting: 'b' is blue, '=' is black, 'Hello, Fighter!' is red, 'print' is blue, and 'b[:5]' is green. Below the code, there is a white box with a dark gray copy icon and the output 'Hello' in a monospaced font.

```
b = "Hello, Fighter!"  
print(b[:5])
```

→ Hello





# **SLICE TO THE END**

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





## Example

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

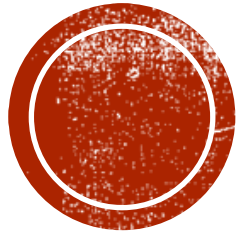


```
b = "Hello, Fighter!"  
print(b[2:])
```



```
llo, Fighter!
```





# NEGATIVE INDEXING

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

## Example

Get the characters:

From: "g" in "Fighter!" (position -5)

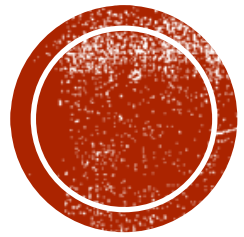
To, but not included: "r" in "Fighter!" (position -2)



```
b = "Hello, Fighter!"  
print(b[-5:-2])
```

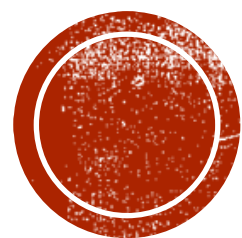
```
hte
```





# **PYTHON - MODIFY STRINGS**

Python has a set of built-in methods that you can use on strings.



**UPPER CASE**



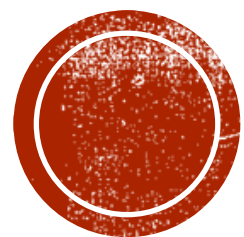
## Example

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



```
a = "Hello, Fighter!"  
print(a.upper())
```





**LOWER CASE**





## Example

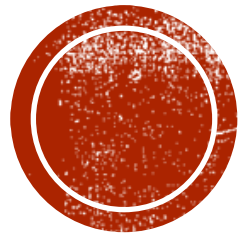
The lower() method returns the string in lower case:



```
a = "Hello, Fighter!"  
print(a.lower())
```







# REMOVE WHITESPACE

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

## Example

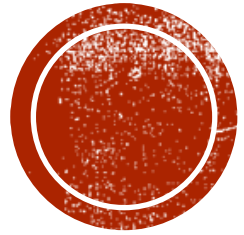
The `strip()` method removes any whitespace from the beginning or the end:



```
a = " Hello, Fighter! "  
print(a.strip()) # returns "Hello, Fighter!"
```

```
Hello, Fighter!
```





# REPLACE STRING



## Example

The `replace()` method replaces a string with another string:

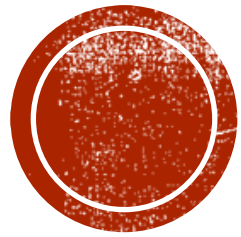


```
a = "Hello, Fighter!"  
print(a.replace("H", "H"))
```



```
Hello, Fighter!
```





# **SPLIT STRING**

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

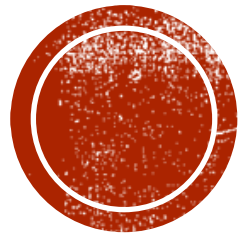
## Example

The `split()` method splits the string into substrings if it finds instances of the separator:

```
▶ a = "Hello, Fighter!"  
  print(a.split(",")) # returns ['Hello', ' Fighter!']  
  
['Hello', ' Fighter!']
```








# PYTHON - STRING CONCATENATION

## **String Concatenation**

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

## Example

Merge variable a with variable b into variable c:



```
a = "Hello"  
b = "Fighter"  
c = a + b  
print(c)
```

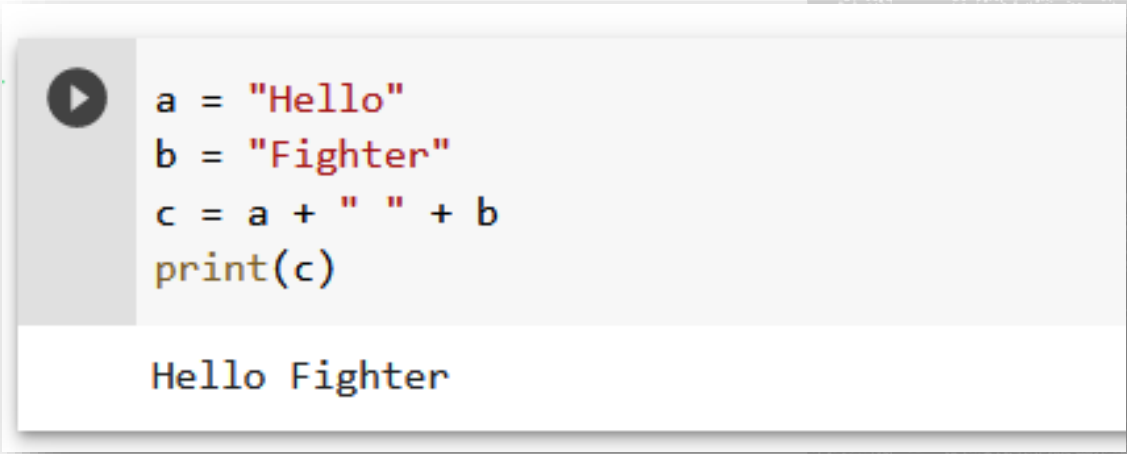
HelloFighter





## Example

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

A code execution snippet showing Python code and its output. The code is: 

```
a = "Hello"  
b = "Fighter"  
c = a + " " + b  
print(c)
```

 The output is: 

```
Hello Fighter
```

```
a = "Hello"  
b = "Fighter"  
c = a + " " + b  
print(c)
```

Hello Fighter

