/MY JOURNAL TO PYTHON

4110E231 柯立丹

/CONTENTS

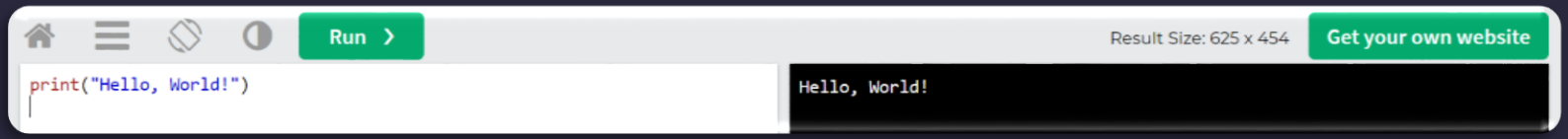
/01

<WHAT IS PYTHON>

/LEARN PYTHON

- Python is a popular programming language.
- Python can be used on a server to create web applications.

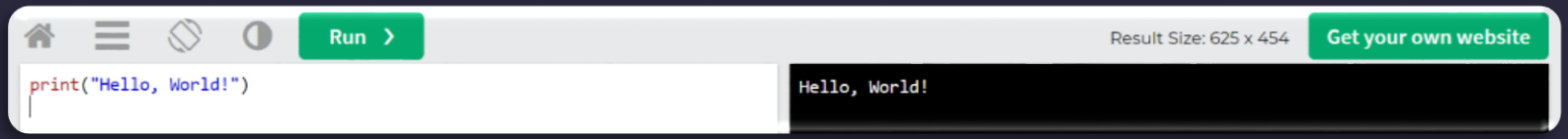
Example:



/WHAT PYTHON

- Python is a popular programming language.
- Python can be used on a server to create web applications.

Example:



/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 IS PYTHON

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.

/02

<PYTHON SYNTAX>

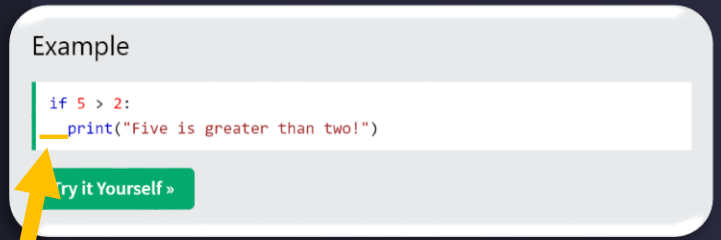
<Python Indentation>

/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.



Indentation

/PYTHON INDENTATION

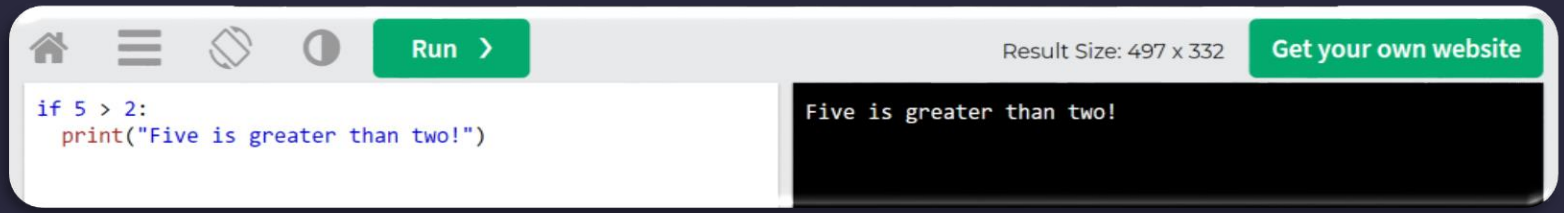
Python uses **indentation** to indicate a block of code.

Example

```
if 5 > 2:
    print("Five is greater than two!")
```

Try it Yourself »

With Indentation



/PYTHON INDENTATION

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

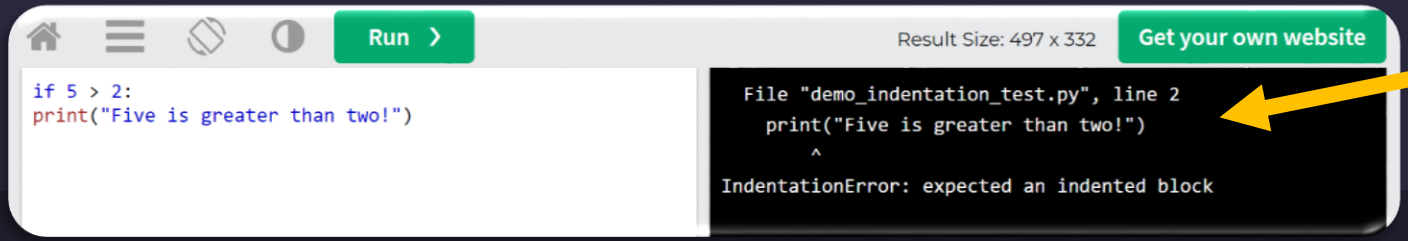
Example

Syntax Error:

```
if 5 > 2:
print("Five is greater than two!")
```

Try it Yourself »

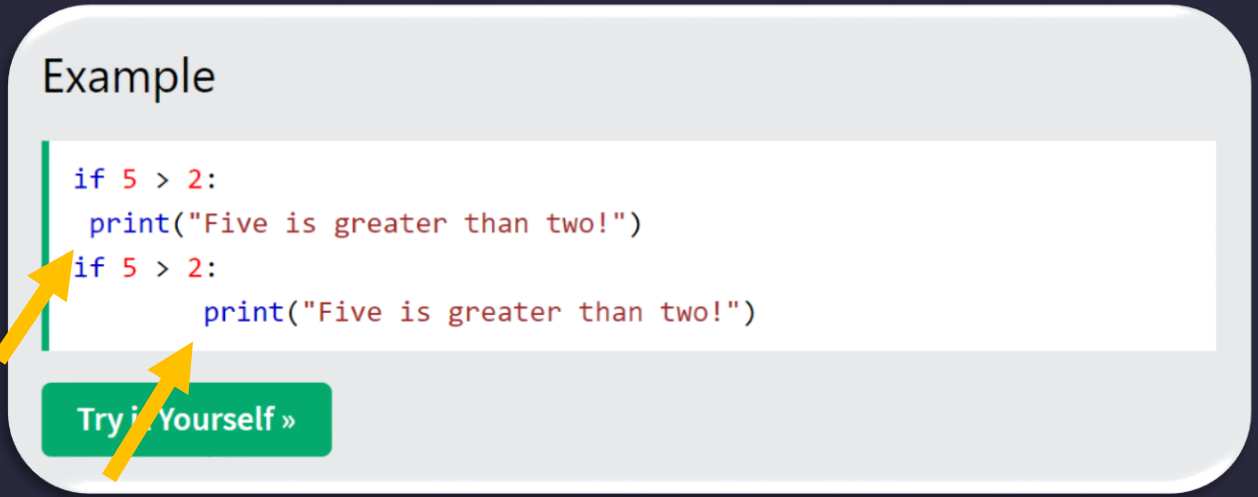
W/O Indentation



RESULT

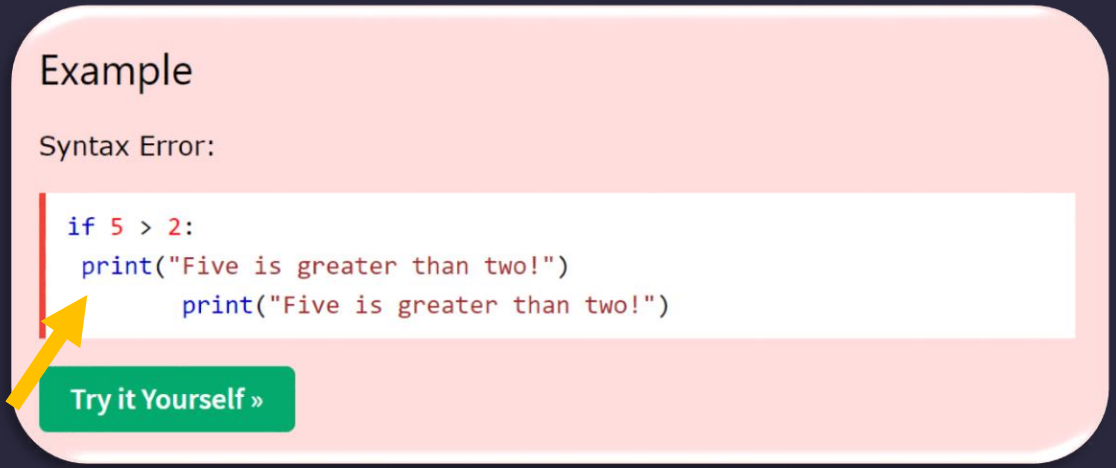
/PYTHON INDENTATION

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.



/PYTHON INDENTATION

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



<Python Comments>

/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.

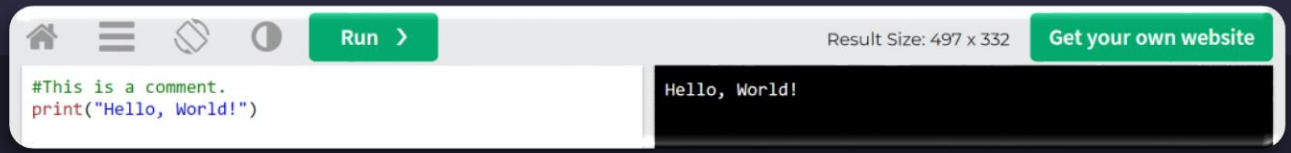
Creating a Comment:

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

Example

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

Try it Yourself »



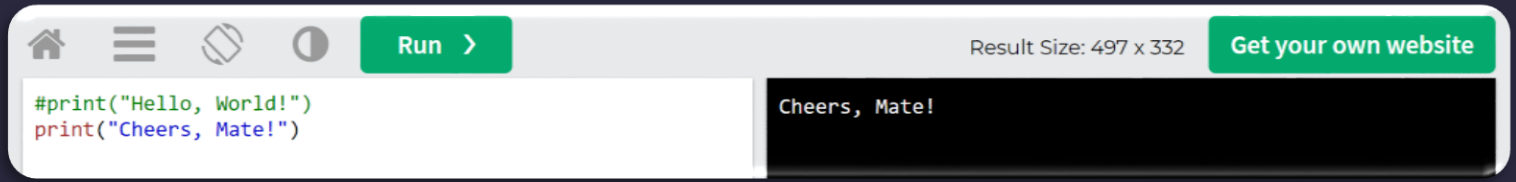
/PYTHON COMMENTS

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

Example

```
#print("Hello, World!")
print("Cheers, Mate!")
```

Try it Yourself »



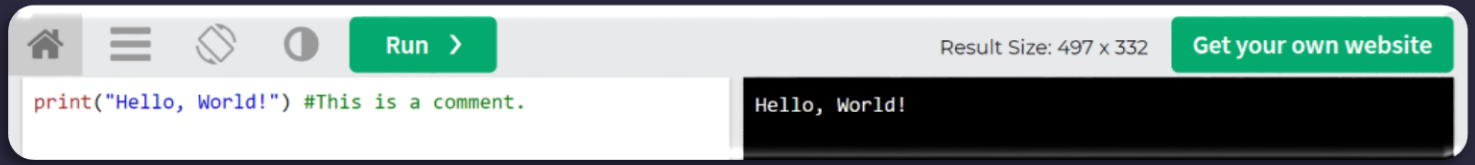
/PYTHON COMMENTS

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

Example

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

Try it Yourself »



/PYTHON COMMENTS

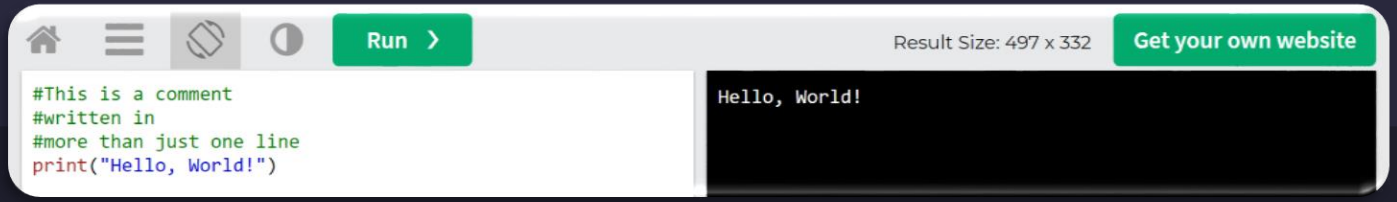
Multi Line Comments

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, World!")
```

Try it Yourself »



/PYTHON COMMENTS

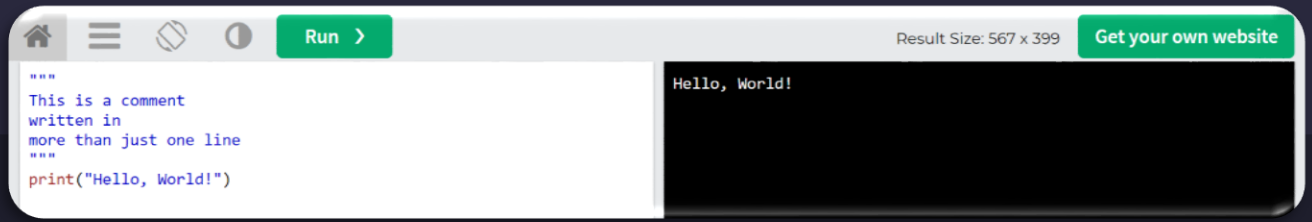
Or, not quite as intended, you can use a multiline string.

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

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

Try it Yourself »



/PYTHON COMMENTS

Note: 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.

<Python Variables>

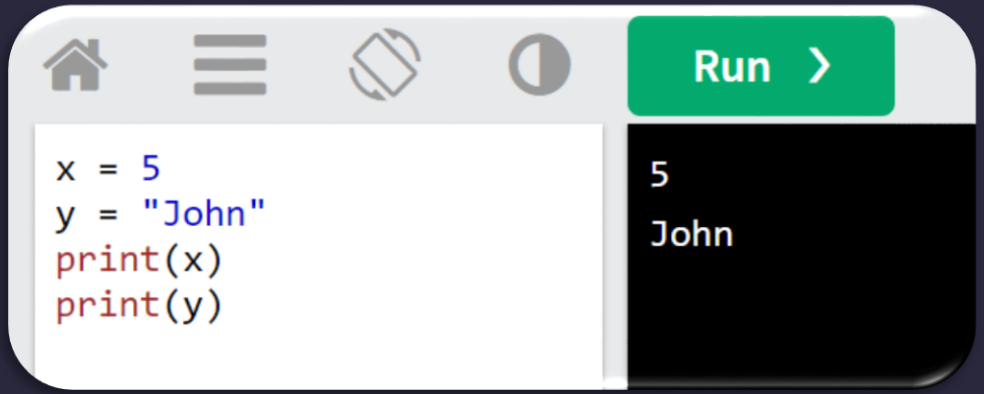
/PYTHON VARIABLES

- Variables are containers for storing data values.

Example

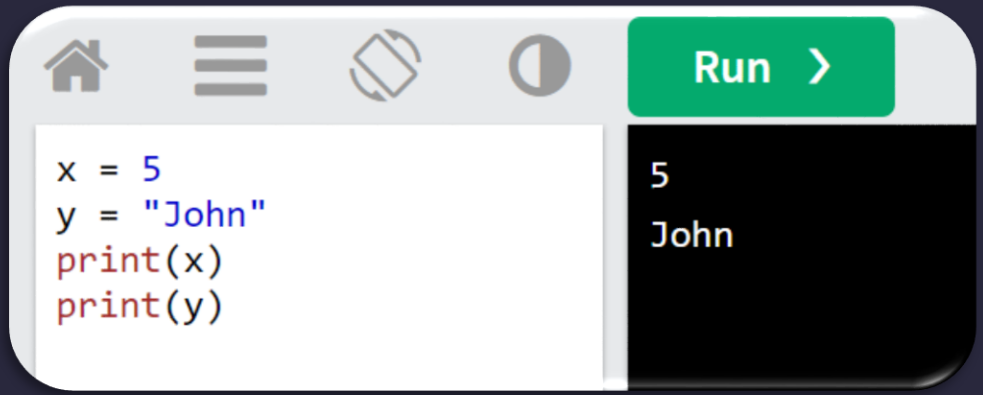
```
x = 5
y = "John"
print(x)
print(y)
```

Try it Yourself »



/PYTHON VARIABLES – CREATING VARIABLES

Python has no command for declaring a variable.
A variable is created the moment you first assign a value to it.



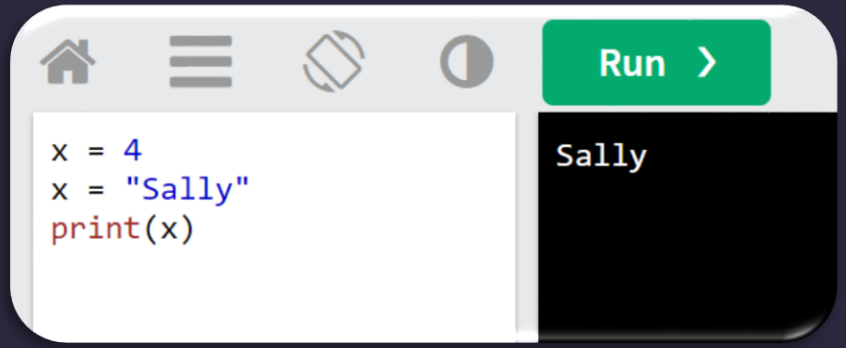
/PYTHON VARIABLES

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

Example

```
x = 4       # x is of type int
x = "Sally" # x is now of type str
print(x)
```

Try it Yourself »



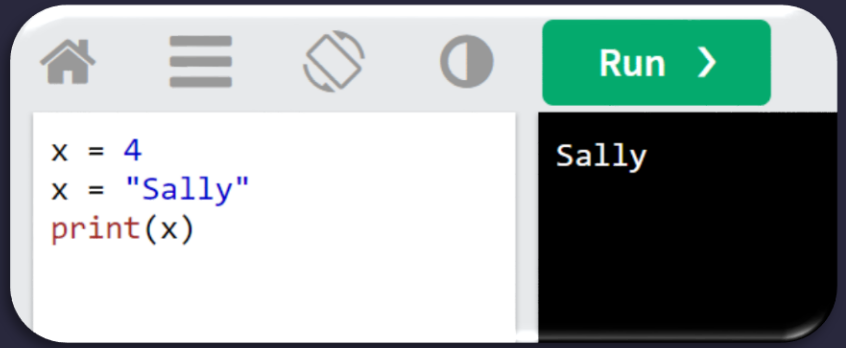
/PYTHON VARIABLES

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

Example

```
x = 4       # x is of type int
x = "Sally" # x is now of type str
print(x)
```

Try it Yourself »



/PYTHON VARIABLES - Get the Type

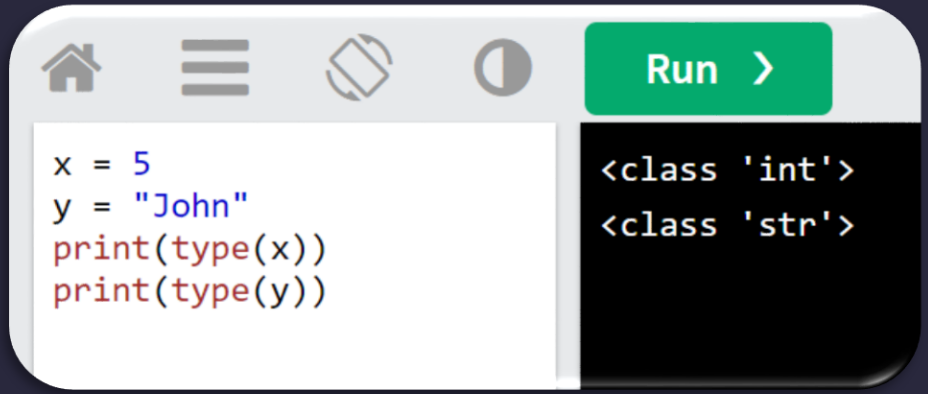
Get the Type

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

Example

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

Try it Yourself »



/PYTHON VARIABLES

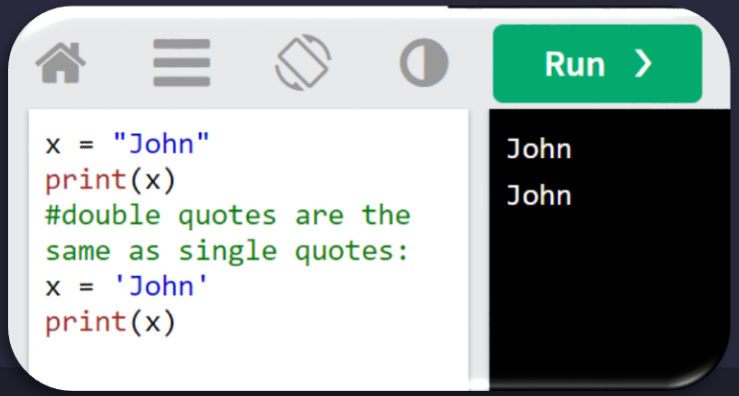
Single or Double Quotes?

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

Example

```
x = "John"
# is the same as
x = 'John'
```

Try it Yourself »



/PYTHON VARIABLES

Case-Sensitive

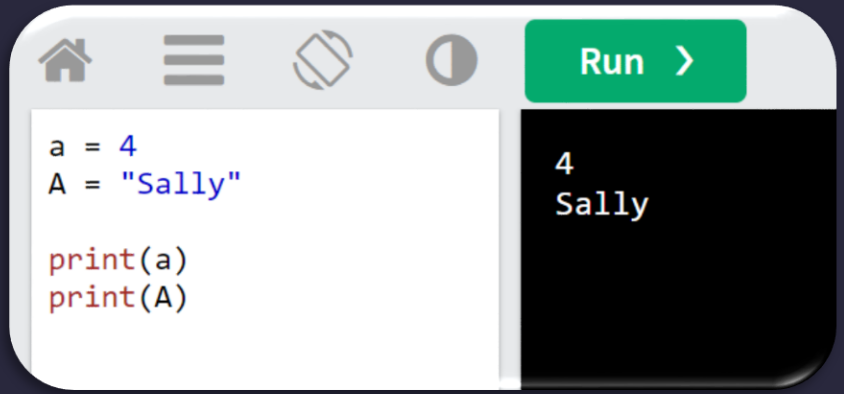
Variable names are case-sensitive.

Example

This will create two variables:

```
a = 4
A = "Sally"
#A will not overwrite a
```

Try it Yourself »



/PYTHON VARIABLES

Casting

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

<Python – Variables Names>

/PYTHON VARIABLES - 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:

/PYTHON VARIABLES - Variable Names

- 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

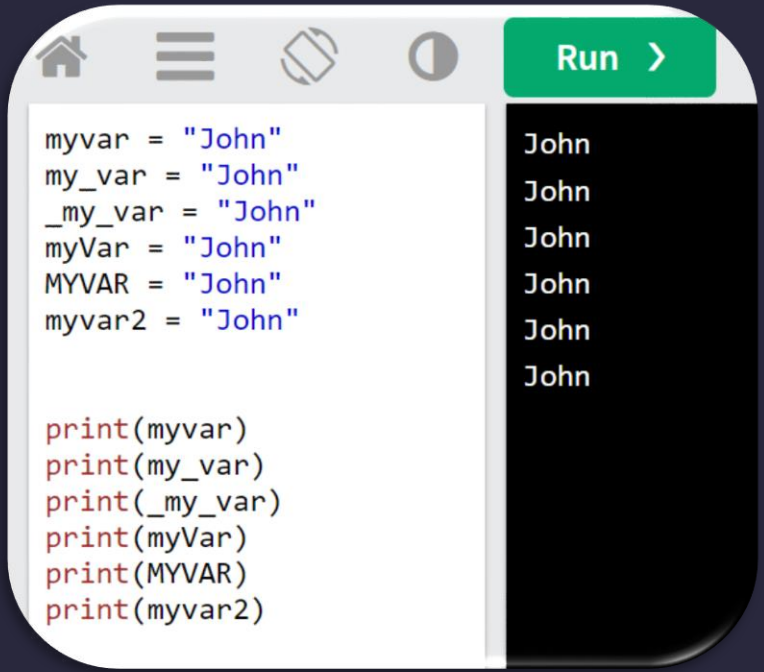
/PYTHON VARIABLES - Variable Names

Example

Legal variable names:

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

Try it Yourself »



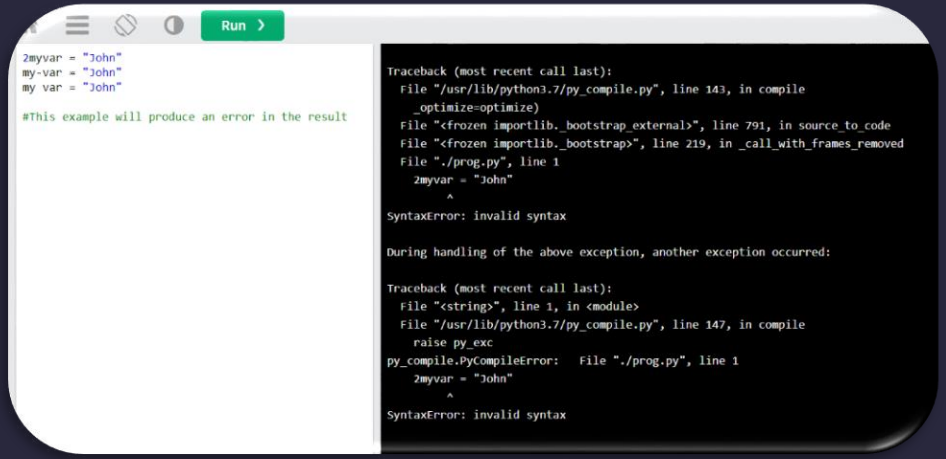
/PYTHON VARIABLES - Variable Names

Example

Illegal variable names:

```
2myvar = "John"
my-var = "John"
my var = "John"
```

Try it Yourself »



- Remember that variable names are case-sensitive

/PYTHON VARIABLES - 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:

```
myVariableName = "John"
```

Pascal Case

Each word starts with a capital letter:

```
MyVariableName = "John"
```

Snake Case

Each word is separated by an underscore character:

```
my_variable_name = "John"
```

/PYTHON VARIABLES - Assign Multiple Values

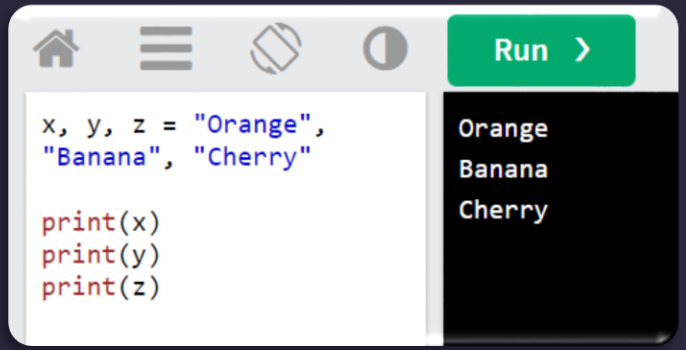
Many Values to Multiple Variables

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

Example

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

Try it Yourself »



/PYTHON VARIABLES - Assign Multiple Values

Unpack a Collection

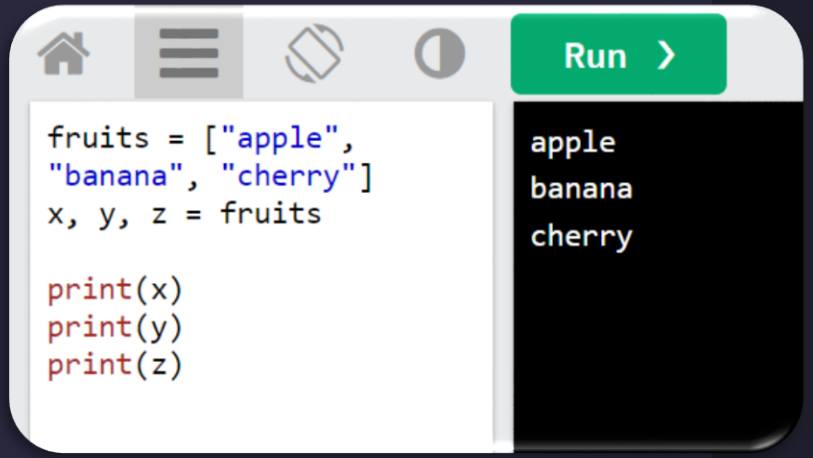
If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

Example

Unpack a list:

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

Try it Yourself »



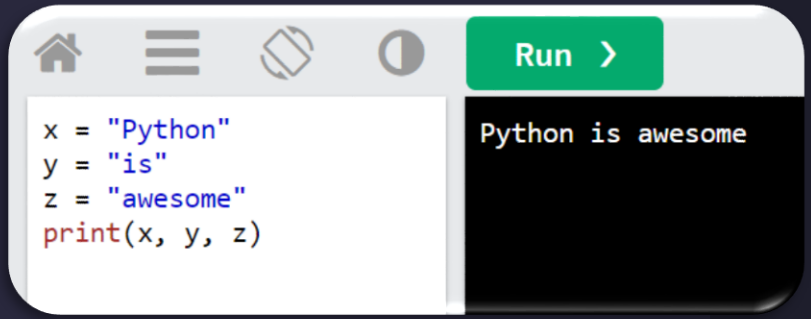
/PYTHON VARIABLES - Output Variables

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

Example

```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

Try it Yourself »



/PYTHON VARIABLES - Output Variables

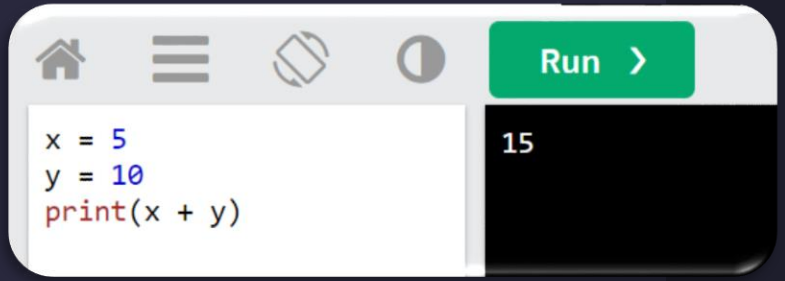
Notice the space character after `"Python "` and `"is "`, without them the result would be "Pythonisawesome".

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

Example

```
x = 5
y = 10
print(x + y)
```

Try it Yourself »



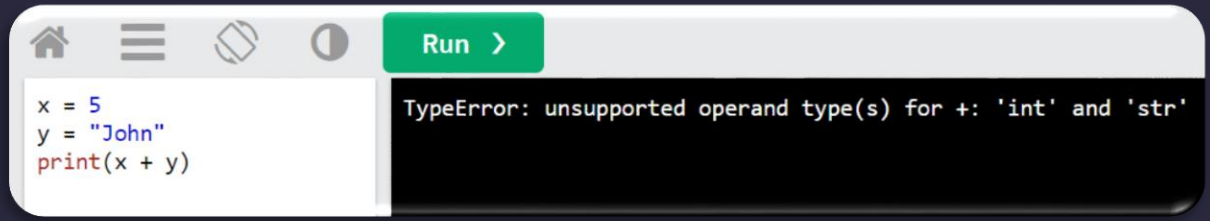
/PYTHON VARIABLES - Output Variables

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 = 5
y = "John"
print(x + y)
```

Try it Yourself »



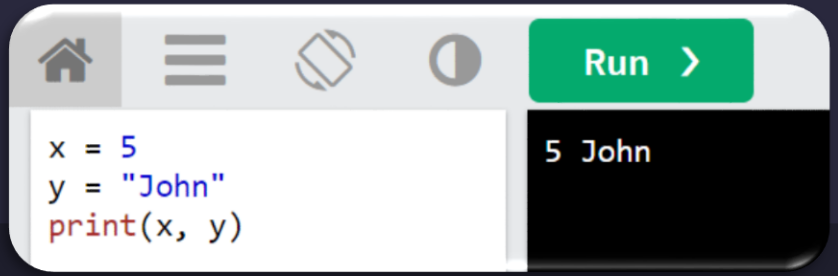
/PYTHON VARIABLES - Output Variables

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

Example

```
x = 5
y = "John"
print(x, y)
```

Try it Yourself »



/PYTHON VARIABLES - Global Variables

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

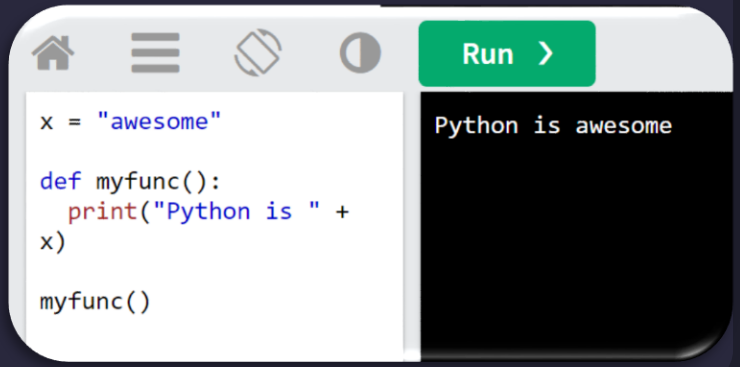
Global variables can be used by everyone, both inside of functions and outside.

Example

Create a variable outside of a function, and use it inside the function

```
x = "awesome"

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

myfunc()
```

Try it Yourself »



/PYTHON VARIABLES - Global Variables

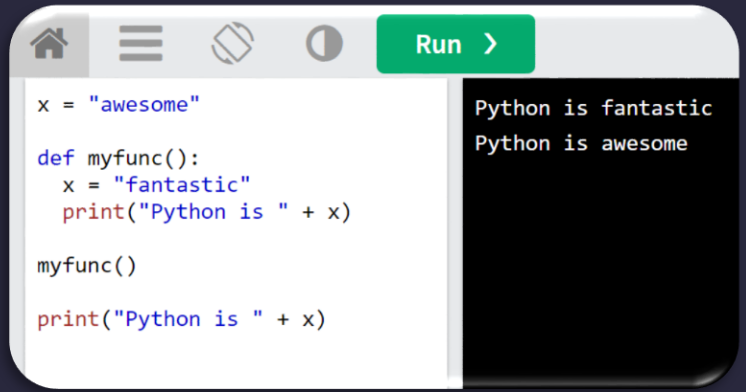
If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Example

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

```
x = "awesome"

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

myfunc()

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

Try it Yourself »



/PYTHON VARIABLES - Global Variables

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.

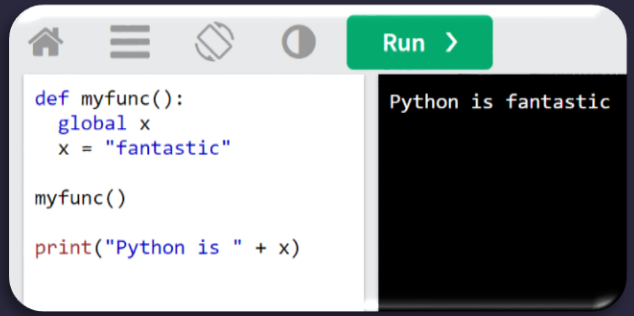
/PYTHON VARIABLES - Global Variables

Example

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

```
def myfunc():
  global x
  x = "fantastic"

myfunc()

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

Try it Yourself »



/PYTHON VARIABLES - Global Variables

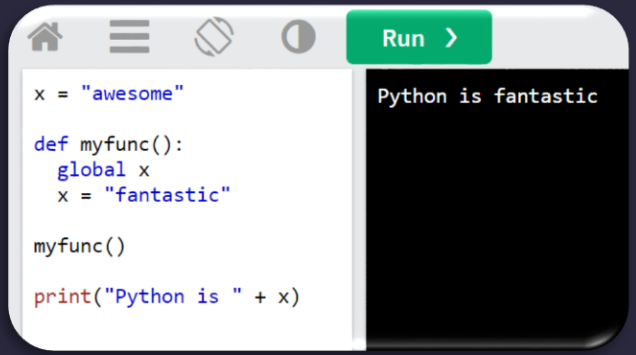
Also, use the global keyword if you want to change a global variable inside a function.

Example

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

```
x = "awesome"

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

myfunc()

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

Try it Yourself »



<Python Data Types>

/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 DATA TYPES

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`

/PYTHON DATA TYPES - Getting the Data Type

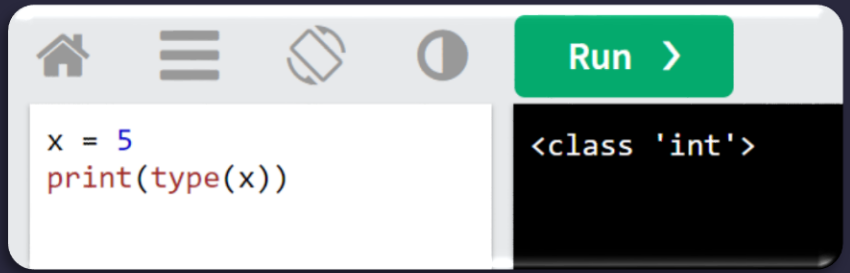
You can get the data type of any object by using the type() function:

Example

Print the data type of the variable x:

```
x = 5
print(type(x))
```

Try it Yourself »



/PYTHON DATA TYPES - Getting the Data Type

Example:	Data Type	Try it
`x = "Hello World"`	str	Try it »
`x = 20`	int	Try it »
`x = 20.5`	float	Try it »
`x = 1j`	complex	Try it »
`x = ["apple", "banana", "cherry"]`	list	Try it »
`x = ("apple", "banana", "cherry")`	tuple	Try it »
`x = range(6)`	range	Try it »

/PYTHON DATA TYPES - Getting the Data Type

`x = {"name" : "John", "age" : 36}`	dict	Try it »
`x = {"apple", "banana", "cherry"}`	set	Try it »
`x = frozenset({"apple", "banana", "cherry"})`	frozenset	Try it »
`x = True`	bool	Try it »
`x = b"Hello"`	bytes	Try it »
`x = bytearray(5)`	bytearray	Try it »
`x = memoryview(bytes(5))`	memoryview	Try it »
`x = None`	NoneType	Try it »

/PYTHON DATA TYPES - Setting the Specific Data Type

Example	Data Type	Try it
`x = str("Hello World")`	str	Try it »
`x = int(20)`	int	Try it »
`x = float(20.5)`	float	Try it »
`x = complex(1j)`	complex	Try it »
`x = list(("apple", "banana", "cherry"))`	list	Try it »
`x = tuple(("apple", "banana", "cherry"))`	tuple	Try it »

/PYTHON DATA TYPES - Setting the Specific Data Type

`x = range(6)`	range	Try it »
`x = dict(name="John", age=36)`	dict	Try it »
`x = set(("apple", "banana", "cherry"))`	set	Try it »
`x = frozenset(("apple", "banana", "cherry"))`	frozenset	Try it »
`x = bool(5)`	bool	Try it »
`x = bytes(5)`	bytes	Try it »
`x = bytearray(5)`	bytearray	Try it »
`x = memoryview(bytes(5))`	memoryview	Try it »

/03

<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:

/PYTHON NUMBERS

Example

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

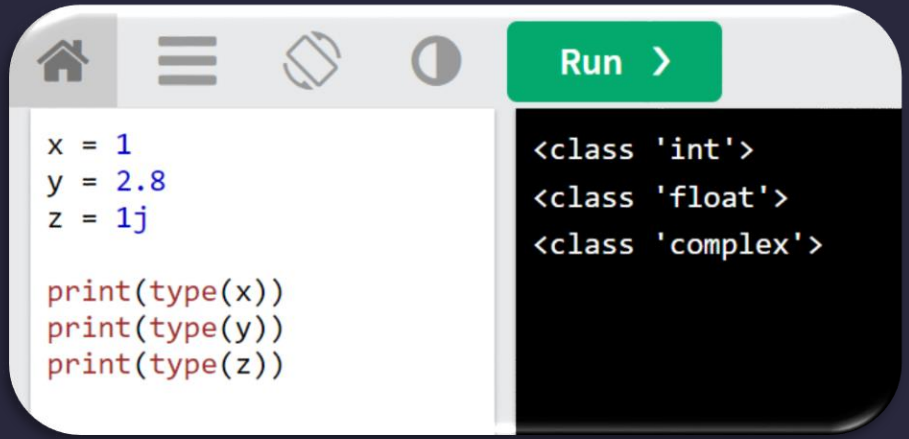
/PYTHON NUMBERS

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

Example

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

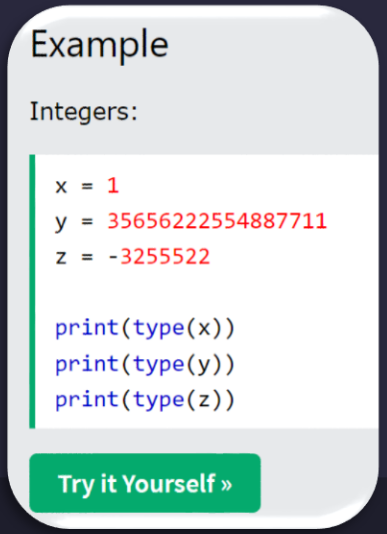
Try it Yourself »

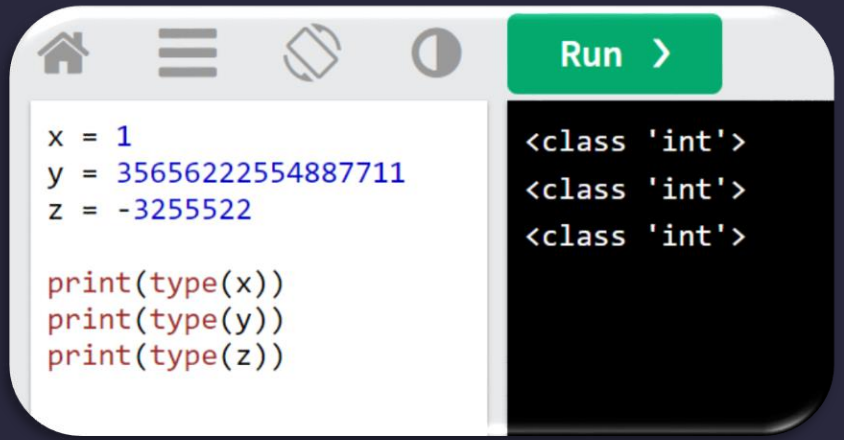


<Int/int>

/PYTHON NUMBERS - Int

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





<Float>

/PYTHON NUMBERS - Float

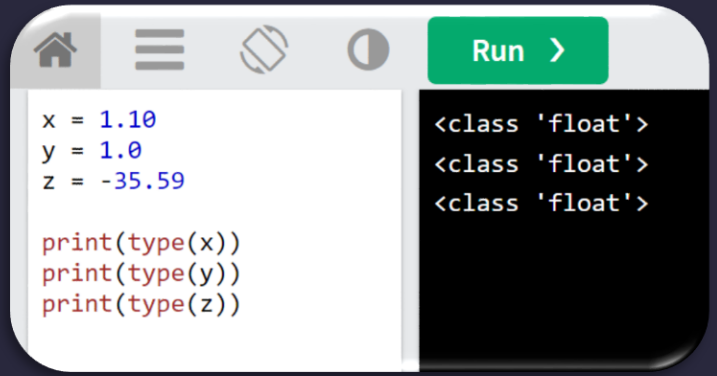
Float, or "**floating point number**" is a number, positive or negative, containing one or more decimals.

Example

Floats:

```
x = 1.10
y = 1.0
z = -35.59

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

Try it Yourself »



/PYTHON NUMBERS - Float

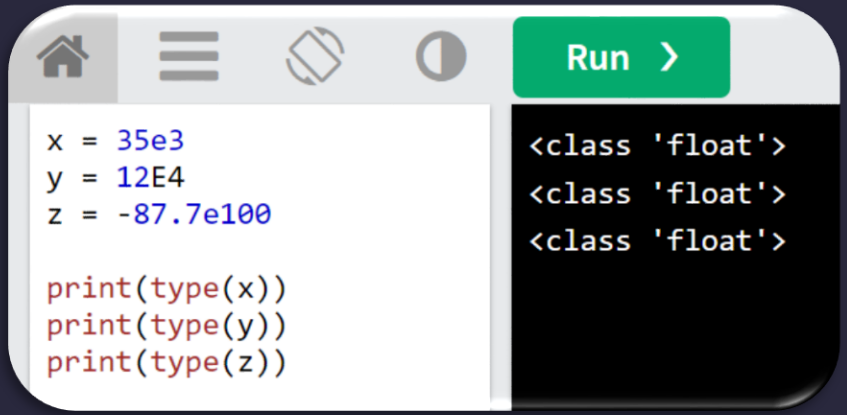
Float can also be scientific numbers with an "**e**" to indicate the power of 10.

Example

Floats:

```
x = 35e3
y = 12E4
z = -87.7e100

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

Try it Yourself »



<Complex>

/PYTHON NUMBERS - Complex

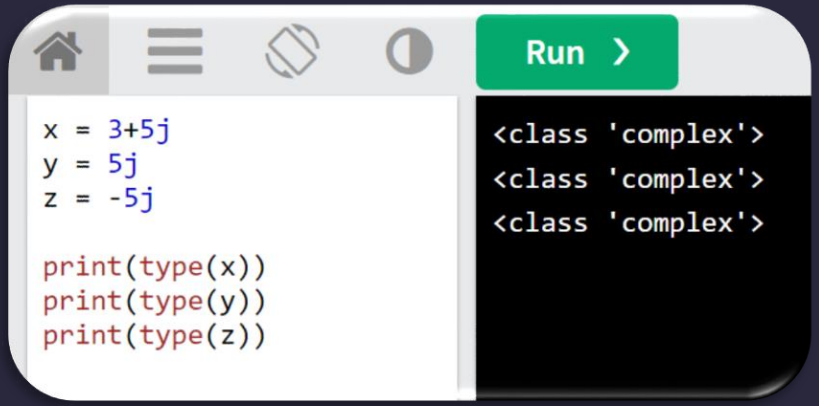
Complex numbers are written with a "**j**" as the imaginary part:

Example

Complex:

```
x = 3+5j
y = 5j
z = -5j

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

Try it Yourself »



/PYTHON NUMBERS - Type Conversion

Type Conversion

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

```
#convert from int to float:
x = float(1)

#convert from float to int:
y = int(2.8)

#convert from int to complex:
z = complex(1)

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

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

```
1.0
2
(1+0j)
<class 'float'>
<class 'int'>
<class 'complex'>
```

/PYTHON NUMBERS - Type Conversion

Type Conversion

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

/PYTHON NUMBERS - Random Number

Random Number

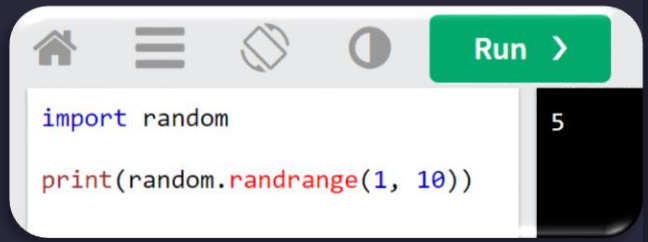
Python does not have a **random()** function to make a random number, but Python has a built-in module called random that can be used to make **random** numbers:

Example

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

```
import random

print(random.randrange(1, 10))
```

Try it Yourself »



<PYTHON CASTING>

/PYTHON CASTING - Specify a Variable Type

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

/PYTHON CASTING - Specify a Variable Type

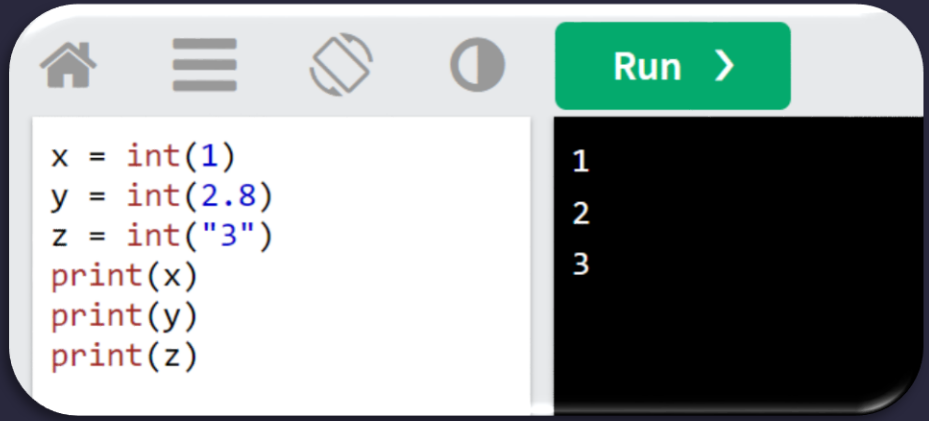
EXAMPLE: int

Example

Integers:

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

Try it Yourself »



/PYTHON CASTING - Specify a Variable Type

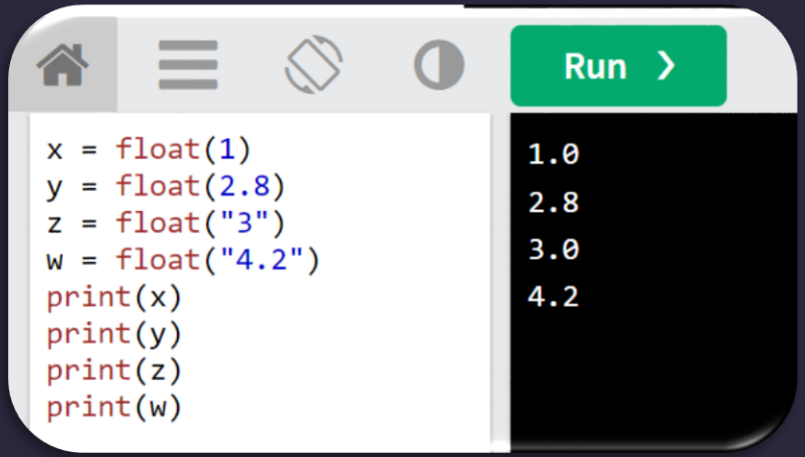
EXAMPLE: float

Example

Floats:

```
x = float(1)      # x will be 1.0
y = float(2.8)    # y will be 2.8
z = float("3")    # z will be 3.0
w = float("4.2") # w will be 4.2
```

Try it Yourself »



/PYTHON CASTING - Specify a Variable Type

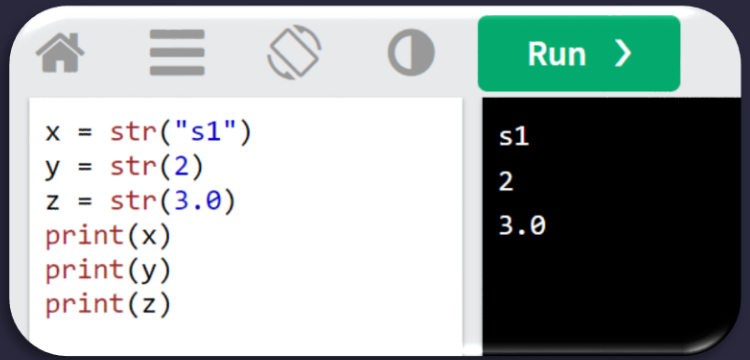
EXAMPLE: string

Example

Strings:

```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

Try it Yourself »




<PYTHON STRINGS>




<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:

Example

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

Try it Yourself »

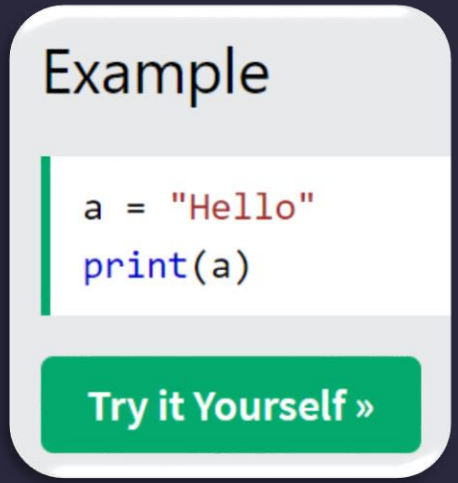
Run ›

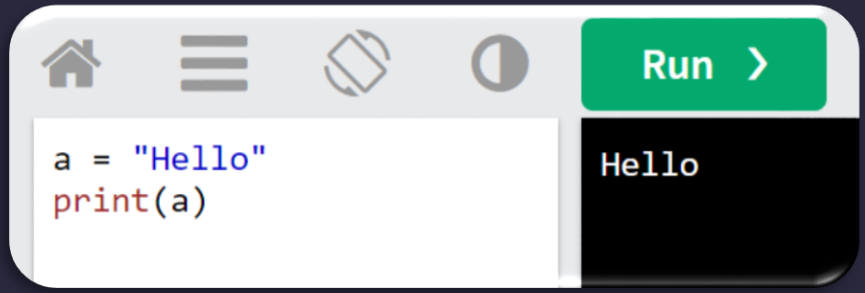
```
#You can use double or single quotes:

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

```
Hello
Hello
```

/PYTHON STRINGS - 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:





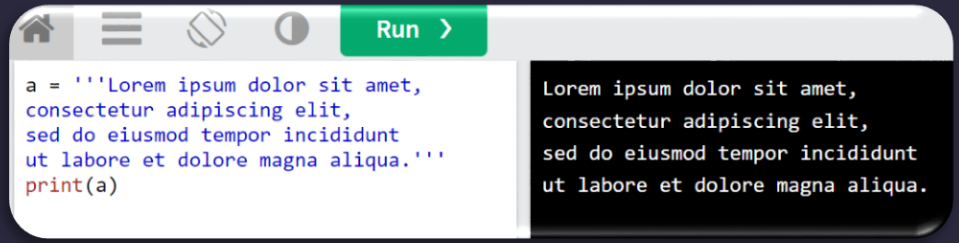
/PYTHON STRINGS - Multiline Strings

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

Example

```
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

Try it Yourself »



/PYTHON STRINGS - Multiline Strings

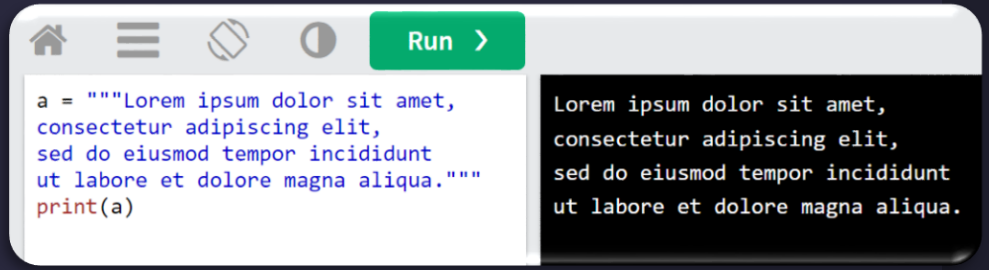
Or three single quotes:

Example

You can use three double quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

Try it Yourself »



<SLICING STRINGS>

/PYTHON STRINGS - Slicing

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

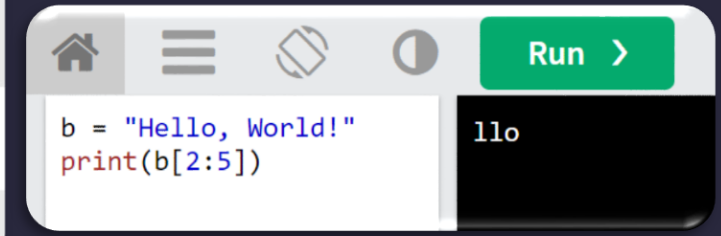
Specify the start index and the end index, separated by a colon, to return a part of the string.

Example

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

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

Try it Yourself »



Note: The first character has index 0.

/PYTHON STRINGS - Slice From the Start

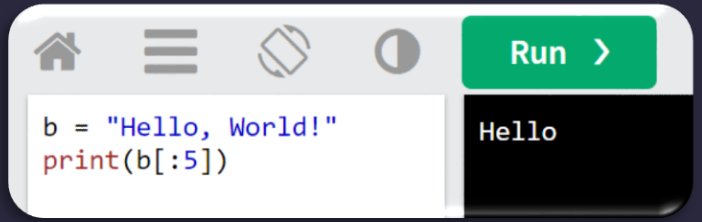
By leaving out the start index, the range will start at the first character:

Example

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

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

Try it Yourself »



/PYTHON STRINGS - Slice to the End

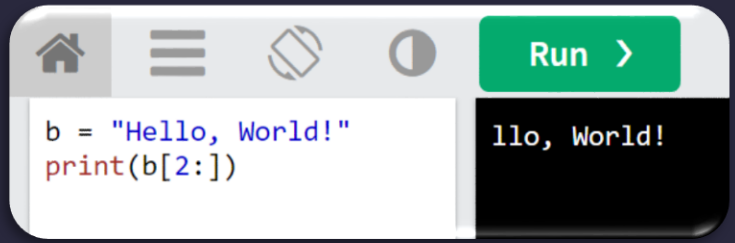
By leaving out the end index, the range will go to the end:

Example

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

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

Try it Yourself »



/PYTHON STRINGS - Negative Indexing

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

Example

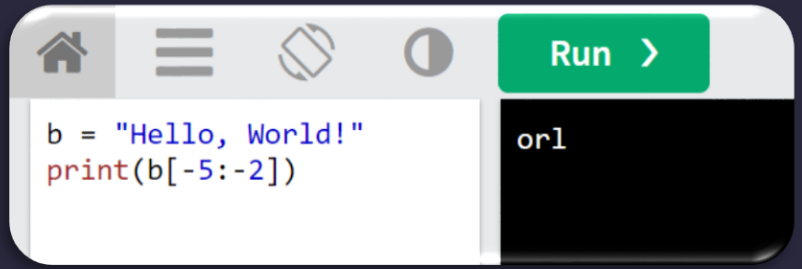
Get the characters:

From: "o" in "World!" (position -5)

To, but not included: "d" in "World!" (position -2):

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

Try it Yourself »



<MODIFY STRINGS>

/PYTHON STRINGS – MODIFY STRINGS

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

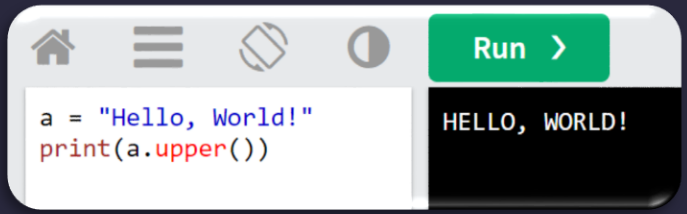
Upper Case

Example

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

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

Try it Yourself »



/PYTHON STRINGS – MODIFY STRINGS

Remove Whitespace

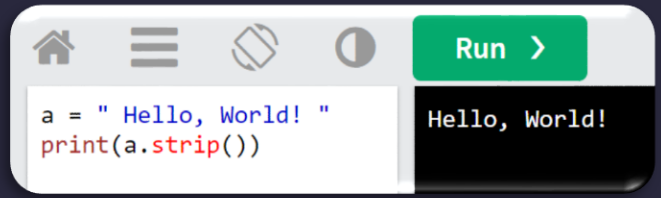
Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

Example

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

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

Try it Yourself »



/PYTHON STRINGS – MODIFY STRINGS

Replace String

Example

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

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

Try it Yourself »

/PYTHON STRINGS – MODIFY STRINGS

Split String

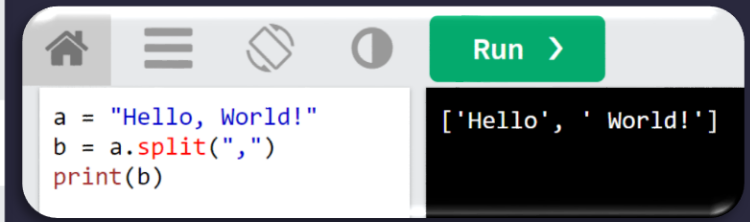
The **split()** method returns a list where the text between the specified separator becomes the list items.

Example

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

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

Try it Yourself »



/PYTHON STRINGS – “STRINGS METHOD”

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

Note: All string methods returns 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

/PYTHON STRINGS – “STRINGS METHOD”

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

/PYTHON STRINGS – “STRINGS METHOD”

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
isascii()	Returns True if all characters in the string are ascii characters
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

/PYTHON STRINGS – “STRINGS METHOD”

join()	Converts the elements of an iterable into a 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
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

/PYTHON STRINGS – MODIFY STRINGS

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

<String Concatenation>

/PYTHON STRINGS – STRING CONCATENATION

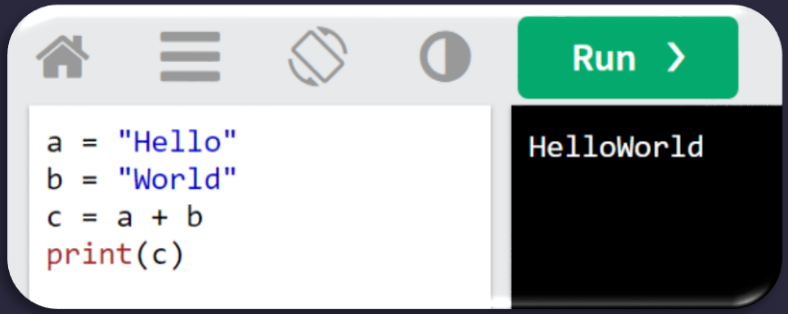
To concatenate, or combine, two strings you can use the + operator.

Example

Merge variable `a` with variable `b` into variable `c`:

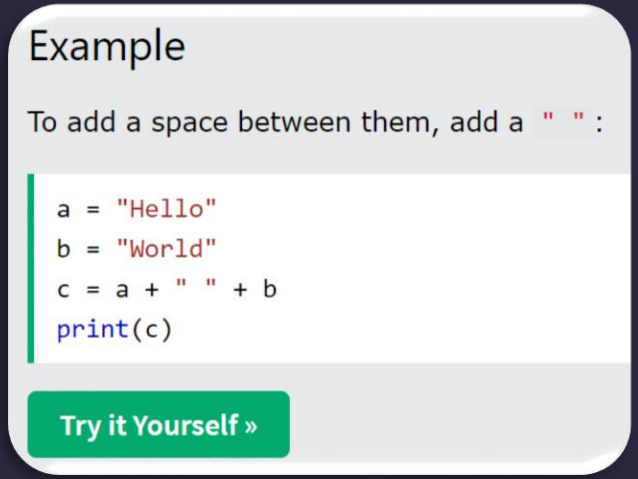
```
a = "Hello"
b = "World"
c = a + b
print(c)
```

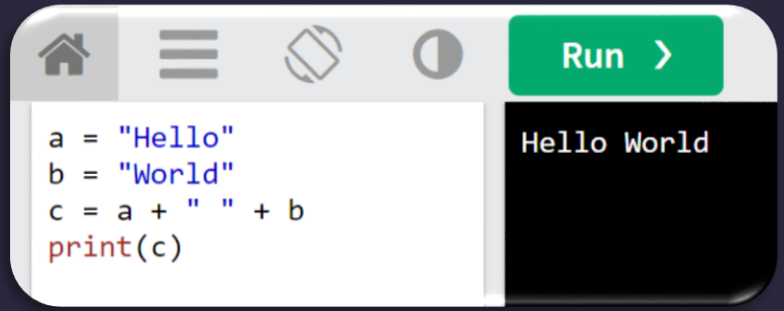
Try it Yourself »



/PYTHON STRINGS – STRING CONCATENATION

EXAMPLE:





<Format - String>

/PYTHON STRINGS – String Format

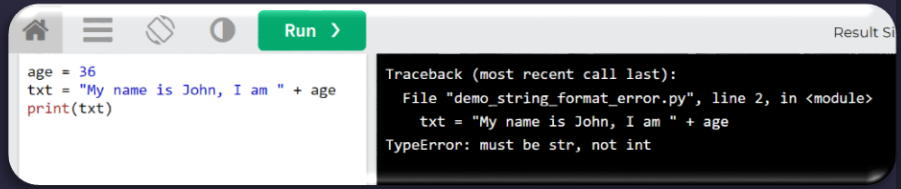
String Format

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

Example

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

Try it Yourself »



/PYTHON STRINGS – String Format

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:

Example

Use the `format()` method to insert numbers into strings:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

Try it Yourself »

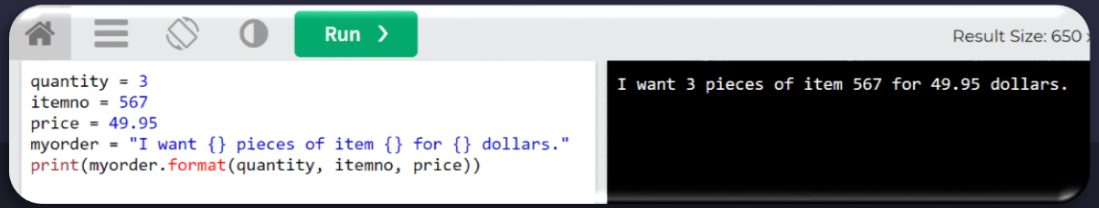
/PYTHON STRINGS – String Format

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

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

Try it Yourself »



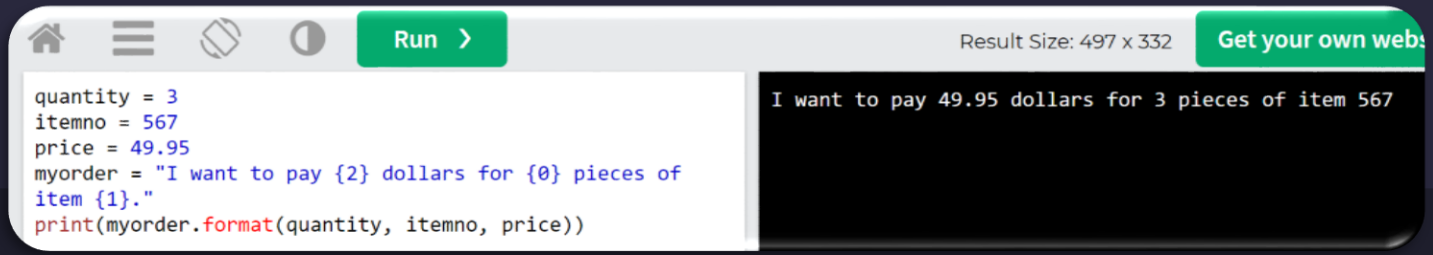
/PYTHON STRINGS – String Format

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

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

Try it Yourself »



/PYTHON STRINGS – String Format

Learn more about String Formatting in our String Formatting chapter.

<Escape Characters>

/PYTHON STRINGS – Escape Character

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:

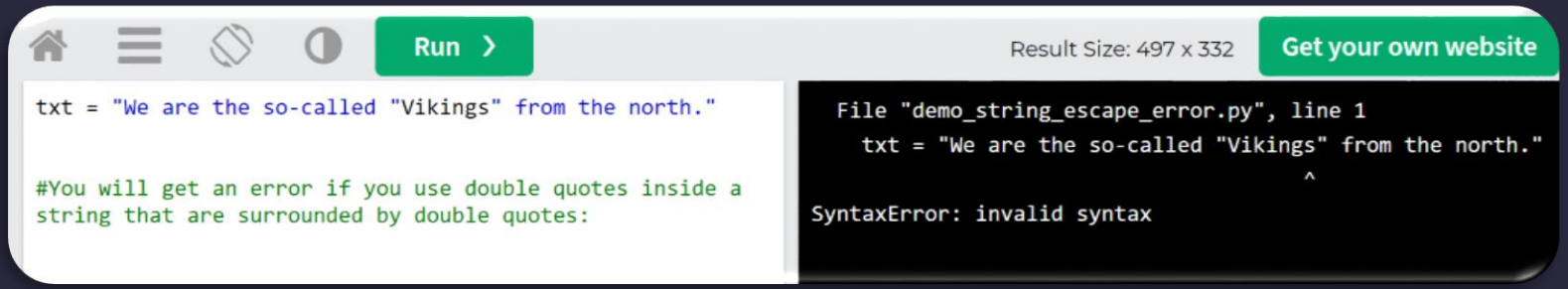
/PYTHON STRINGS – Escape Character

Example

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

```
txt = "We are the so-called "Vikings" from the north."
```

Try it Yourself »



/PYTHON STRINGS – Escape Character

Code	Result	Try it
\'	Single Quote	Try it »
\\	Backslash	Try it »
\n	New Line	Try it »
\r	Carriage Return	Try it »
\t	Tab	Try it »
\b	Backspace	Try it »
\f	Form Feed	
\ooo	Octal value	Try it »
\xhh	Hex value	Try it »


<PYTHON BOOLEANS>




/PYTHON BOOLEANS – Boolean Values

Booleans represent one of two values: **True** or **False**.

Boolean 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**.

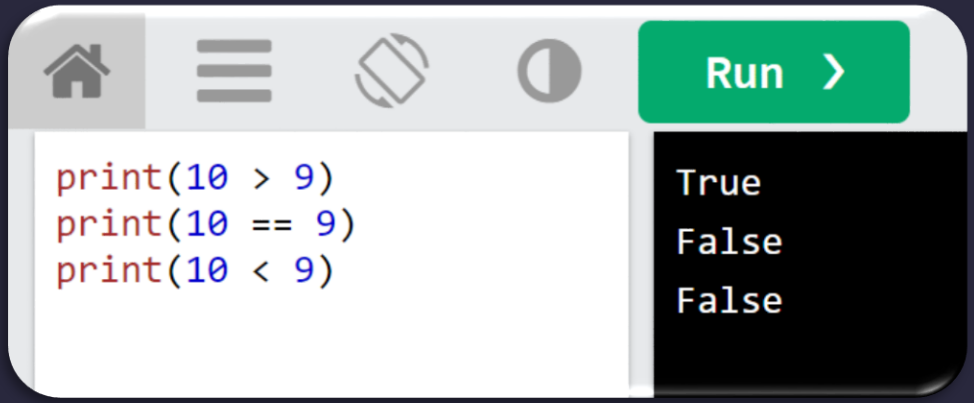
/PYTHON BOOLEANS – Boolean Values

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

Example

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

Try it Yourself »



/PYTHON BOOLEANS – Boolean Values

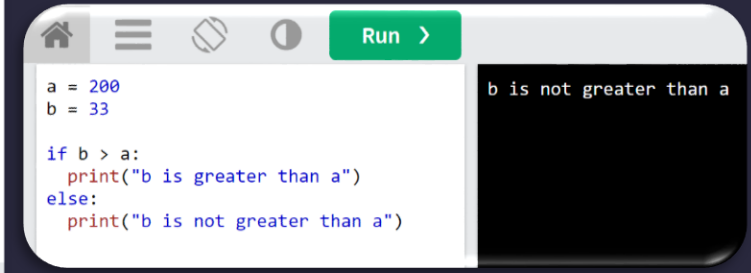
When you run a condition in an if statement, Python returns **True** or **False**:

Example

Print a message based on whether the condition is `True` or `False`:

```
a = 200
b = 33

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

Try it Yourself »



/PYTHON BOOLEANS – Evaluate Values and Variables

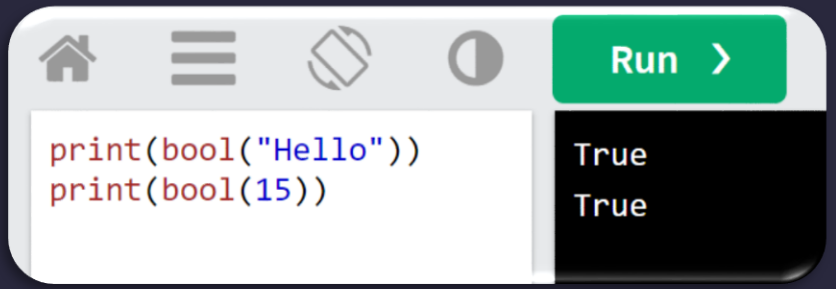
The **bool()** function allows you to evaluate any value, and give you **True** or **False** in return,

Example

Evaluate a string and a number:

```
print(bool("Hello"))
print(bool(15))
```

Try it Yourself »



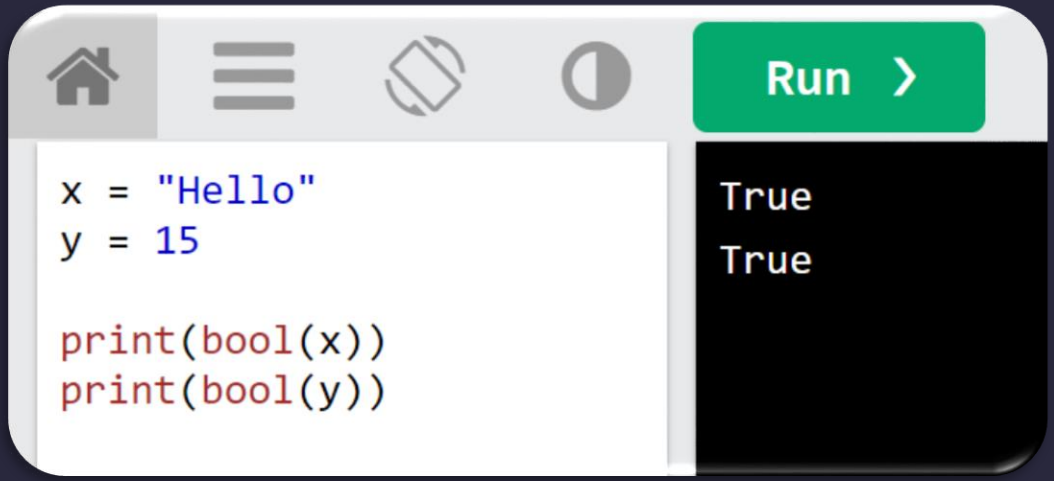
/PYTHON BOOLEANS – Evaluate Values and Variables

Example

Evaluate two variables:

```
x = "Hello"
y = 15

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

Try it Yourself »



/PYTHON BOOLEANS – 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.

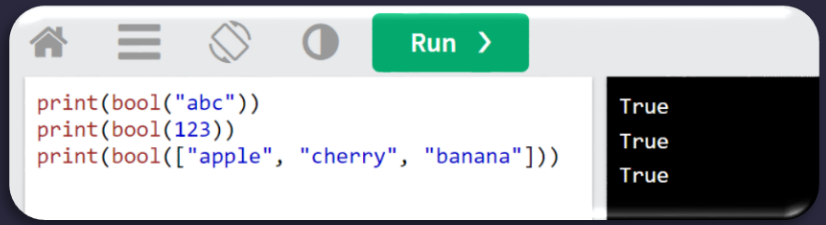
/PYTHON BOOLEANS – Most Values are True

Example

The following will return True:

```
bool("abc")
bool(123)
bool(["apple", "cherry", "banana"])
```

Try it Yourself »



/PYTHON BOOLEANS – 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**.

Example

The following will return False:

```
bool(False)
bool(None)
bool(0)
bool("")
bool(())
bool([])
bool({})
```

Try it Yourself »

Run ›

```
print(bool(False))
print(bool(None))
print(bool(0))
print(bool(""))
print(bool(()))
print(bool([]))
print(bool({}))
```

```
False
False
False
False
False
False
False
```

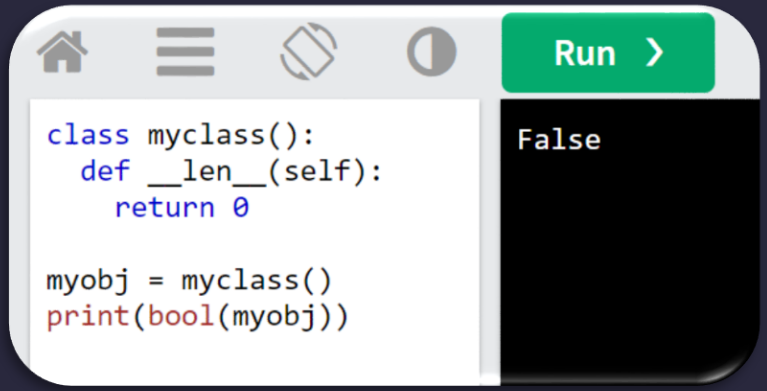
/PYTHON BOOLEANS – Some Values are False

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**:

Example

```
class myclass():
  def __len__(self):
    return 0

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

Try it Yourself »



/PYTHON BOOLEANS – Functions can Return a Boolean

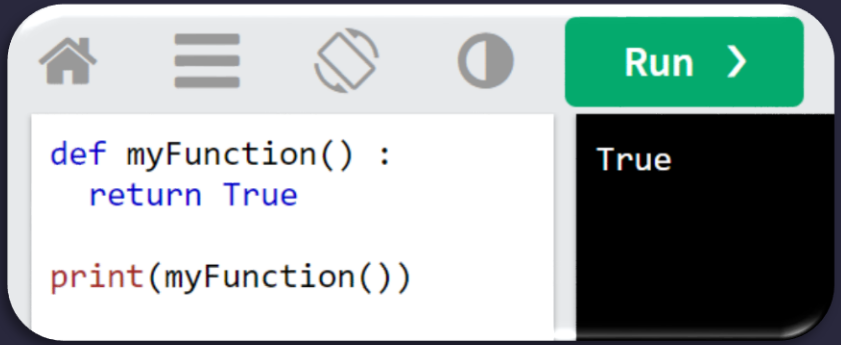
You can create functions that returns a Boolean Value:

Example

Print the answer of a function:

```
def myFunction() :
  return True

print(myFunction())
```

Try it Yourself »



/PYTHON BOOLEANS – Functions can Return a Boolean

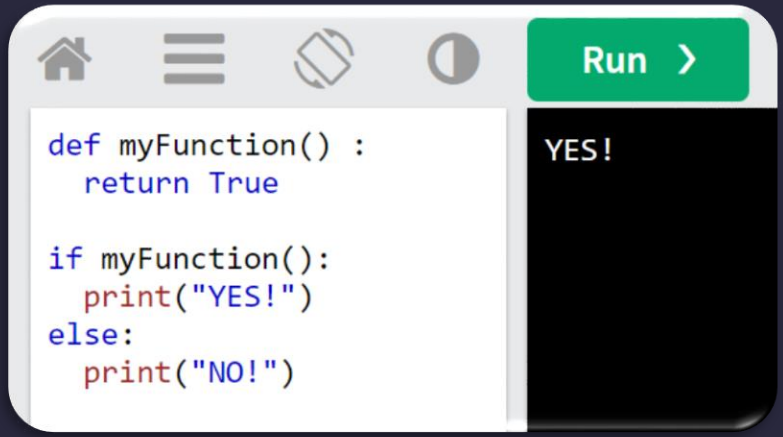
You can execute code based on the Boolean answer of a function:

Example

Print "YES!" if the function returns True, otherwise print "NO!":

```
def myFunction() :
  return True

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

Try it Yourself »



/PYTHON BOOLEANS – Functions can Return a Boolean

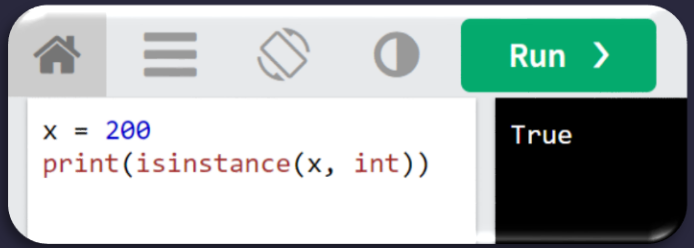
Python also has many built-in functions that return a boolean value, like the **isinstance()** function, which can be used to determine if an object is of a certain data type:

Example

Check if an object is an integer or not:

```
x = 200
print(isinstance(x, int))
```

Try it Yourself »



<PYTHON OPERATORS>

/PYTHON OPERATORS – Functions can Return a Boolean

Operators are used to perform operations on variables and values.

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

/PYTHON OPERATORS

Python divides the operators in the following groups:

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

/PYTHON OPERATORS - Python Arithmetic Operators

Python Arithmetic Operators

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

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

/PYTHON OPERATORS - Python Arithmetic Operators

Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As	Try it
=	x = 5	x = 5	Try it »
+=	x += 3	x = x + 3	Try it »
-=	x -= 3	x = x - 3	Try it »
*=	x *= 3	x = x * 3	Try it »
/=	x /= 3	x = x / 3	Try it »
%=	x %= 3	x = x % 3	Try it »

/PYTHON OPERATORS - Python Arithmetic Operators

//=	x //= 3	x = x // 3	Try it »
**=	x **= 3	x = x ** 3	Try it »
&=	x &= 3	x = x & 3	Try it »
|=	x |= 3	x = x | 3	Try it »
^=	x ^= 3	x = x ^ 3	Try it »
>>=	x >>= 3	x = x >> 3	Try it »
<<=	x <<= 3	x = x << 3	Try it »

/PYTHON OPERATORS - Python Arithmetic Operators

Python Comparison Operators

Comparison operators are used to compare two values:

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

/PYTHON OPERATORS - Python Arithmetic Operators

Python Logical Operators

Logical operators are used to combine conditional statements:

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

/PYTHON OPERATORS - Python Identity Operators

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	Try it
is	Returns True if both variables are the same object	x is y	Try it »
is not	Returns True if both variables are not the same object	x is not y	Try it »

/PYTHON OPERATORS - Python Membership Operators

Python Membership Operators

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

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

/PYTHON OPERATORS - Python Bitwise Operators

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

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


<PYTHON LIST>




INDEX.HTML

/PYTHON LIST

```
mylist = ["apple", "banana", "cherry“]
```

List

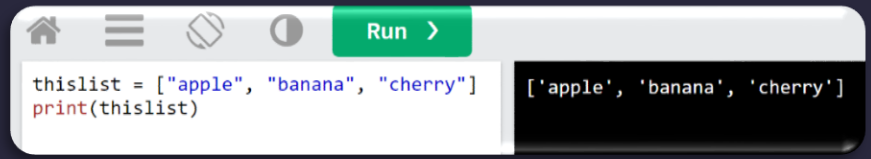
- Lists are used to store multiple items in a single variable.
- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
- Lists are created using square brackets:

Example

Create a List:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

Try it Yourself »



/PYTHON LIST - List Items

List Items

- List items are ordered, changeable, and allow duplicate values.
- List items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

- When we say that lists are ordered, it means that the items have a defined order, and that order will not change.
- If you add new items to a list, the new items will be placed at the end of the list.

Note: There are some list methods that will change the order, but in general: the order of the items will not change.

Changeable

- The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

/PYTHON LIST - List Items

Allow Duplicates

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Example

Lists allow duplicate values:

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

Try it Yourself »

/PYTHON LIST - List Items – Data Types

List Items - Data Types

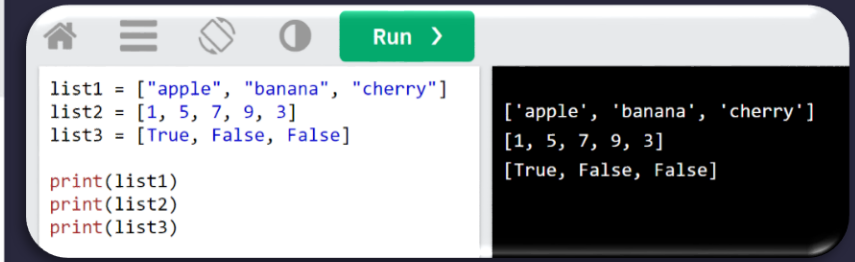
List items can be of any data type:

Example

String, int and boolean data types:

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

Try it Yourself »



/PYTHON LIST - List Items – Data Types

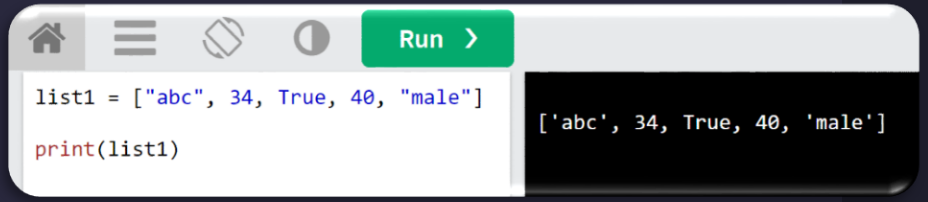
A list can contain different data types:

Example

A list with strings, integers and boolean values:

```
list1 = ["abc", 34, True, 40, "male"]
```

Try it Yourself »



/PYTHON LIST - type()

From Python's perspective, lists are defined as objects with the data type 'list':

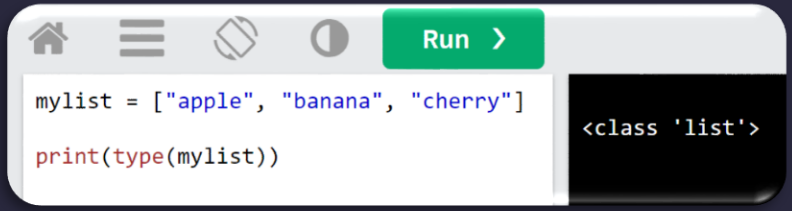
```
<class 'list'>
```

Example

What is the data type of a list?

```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

Try it Yourself »



/PYTHON LIST - The list() Constructor

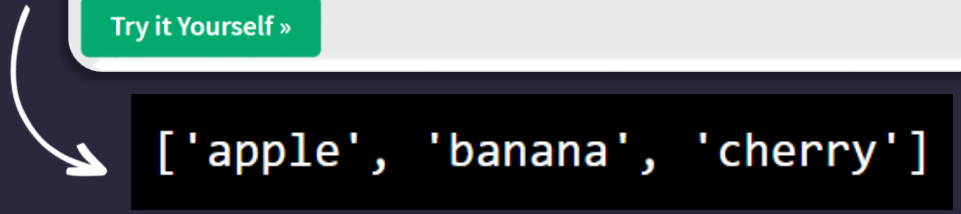
It is also possible to use the list() constructor when creating a new list.

Example

Using the `list()` constructor to make a List:

```
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(thislist)
```

Try it Yourself »



/PYTHON LIST - Python Collections (Arrays)

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered** and changeable. No duplicate members.

<PYTHON TUPLES>

/PYTHON LIST - Python Collections (Arrays)

Tuple

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.

A tuple is a collection which is ordered and unchangeable.

Tuples are written with round brackets.

/PYTHON LIST

Example

Create a Tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Try it Yourself »

```
('apple', 'banana', 'cherry')
```

/PYTHON TUPLES – Tuples Items

Tuple Items

Tuple items are ordered, unchangeable, and allow duplicate values.

Tuple items are indexed, the first item has index **[0]**, the second item has index **[1]** etc.

Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

/PYTHON LIST -

Allow Duplicates

Since tuples are indexed, they can have items with the same value:


```
('apple', 'banana', 'cherry', 'apple', 'cherry')
```

/PYTHON LIST -

Tuple Length

To determine how many items a tuple has, use the len() function:

Example

Print the number of items in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

Try it Yourself »

/PYTHON LIST

Create Tuple With One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

Example

One item tuple, remember the comma:

```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

Try it Yourself »

Run ›

```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

```
<class 'tuple'>
<class 'str'>
```

/PYTHON LIST

Tuple Items - Data Types

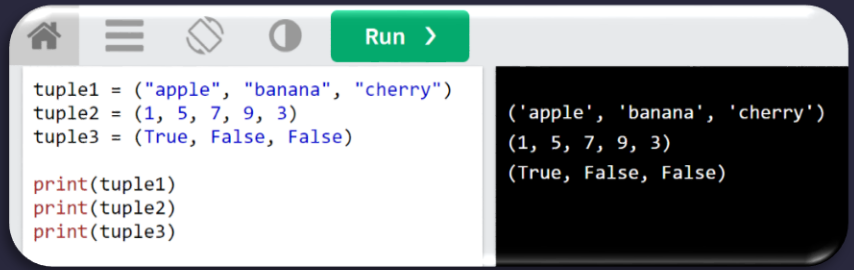
Tuple items can be of any data type:

Example

String, int and boolean data types:

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
```

Try it Yourself »



/PYTHON TUPLE - type()

The tuple() Constructor

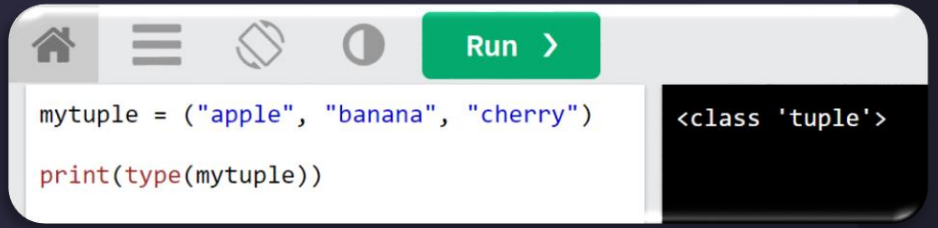
From Python's perspective, tuples are defined as objects with the data type 'tuple':

Example

What is the data type of a tuple?

```
mytuple = ("apple", "banana", "cherry")
print(type(mytuple))
```

Try it Yourself »



<Access Tuple Items>

/PYTHON TUPLE - Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

Example

Print the second item in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Try it Yourself »

Note: The first item has index 0.

/PYTHON TUPLE - Negative Indexing

Negative Indexing

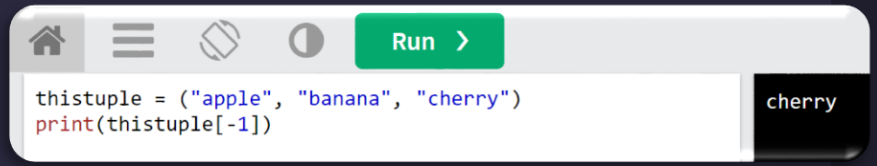
Negative indexing means start from the end.
-1 refers to the last item, **-2** refers to the second last item etc.

Example

Print the last item of the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

Try it Yourself »



/PYTHON TUPLE – Range of Indexes

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

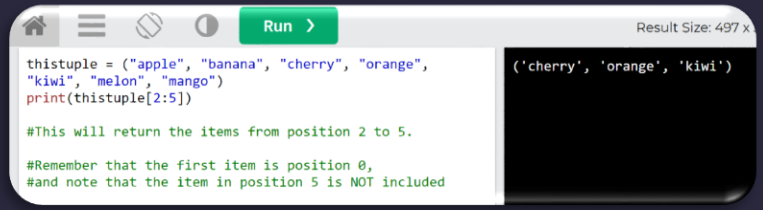
When specifying a range, the return value will be a new tuple with the specified items.

Example

Return the third, fourth, and fifth item:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

Try it Yourself »



Note: The search will start at index 2 (included) and end at index 5 (not included).

/PYTHON TUPLE – Range of Indexes

Remember that the first item has index 0.

By leaving out the start value, the range will start at the first item:

Example

This example returns the items from the beginning to, but NOT included, "kiwi":

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])
```

Try it Yourself »

```
('apple', 'banana', 'cherry', 'orange')
```

<Update Tuples>

/PYTHON TUPLE – Update Tuples

- Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created.
- But there are some workarounds.

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

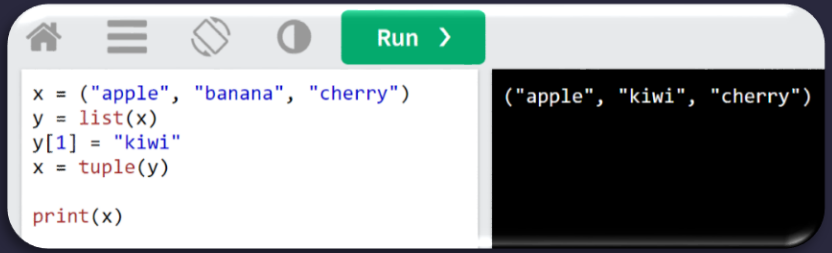
/PYTHON TUPLE – Update Tuples

Example

Convert the tuple into a list to be able to change it:

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

Try it Yourself »

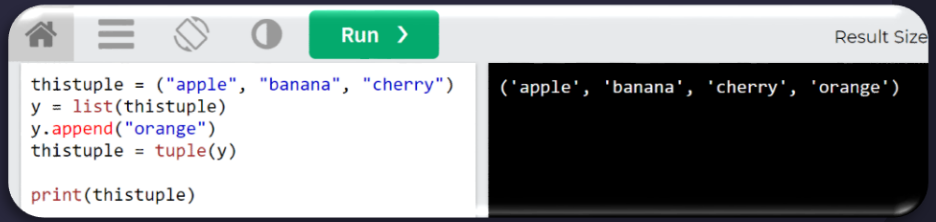


/PYTHON TUPLE – Update Tuples

Add Items

Since tuples are immutable, they do not have a build-in append() method, but there are other ways to add items to a tuple.

1. **Convert into a list:** Just like the workaround for changing a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.



/PYTHON TUPLE – Update Tuples

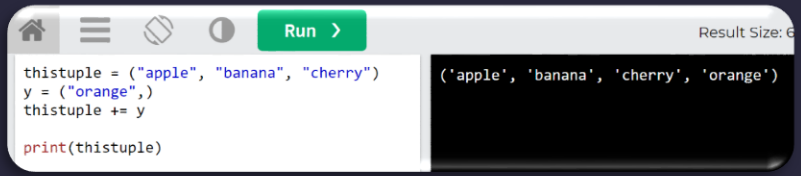
2. **Add tuple to a tuple.** You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

Example

Create a new tuple with the value "orange", and add that tuple:

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)
```

Try it Yourself »



Note: When creating a tuple with only one item, remember to include a comma after the item, otherwise it will not be identified as a tuple.

/PYTHON TUPLE – Remove Items

Note: You cannot remove items in a tuple.

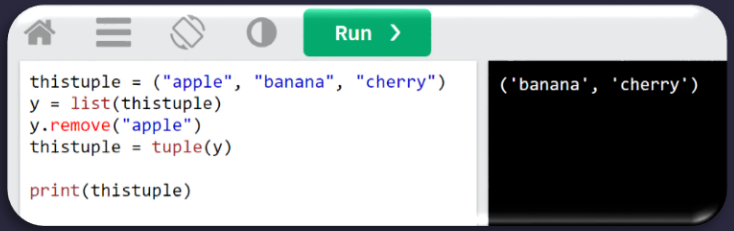
Tuples are **unchangeable**, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items:

Example

Convert the tuple into a list, remove "apple", and convert it back into a tuple:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
```

Try it Yourself »



/PYTHON TUPLE – Remove Items

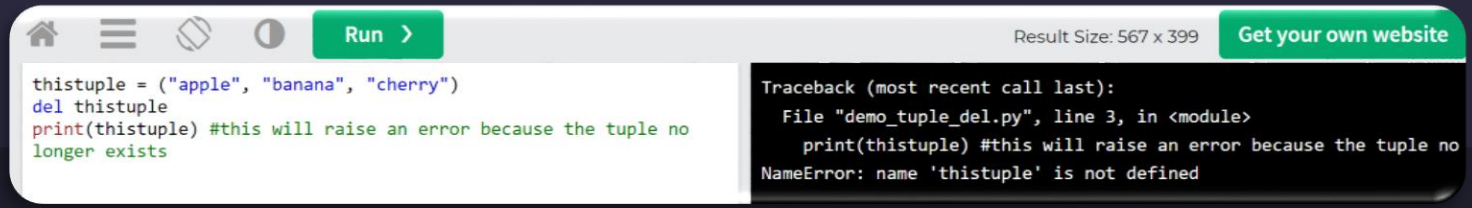
Or you can delete the tuple completely

Example

The `del` keyword can delete the tuple completely:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no
longer exists
```

Try it Yourself »



<Unpacking Tuples>

/PYTHON TUPLE – Unpacking a Tuple

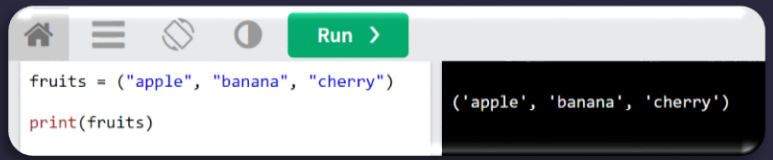
When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

Example

Packing a tuple:

```
fruits = ("apple", "banana", "cherry")
```

Try it Yourself »



/PYTHON TUPLE – Unpacking a Tuple

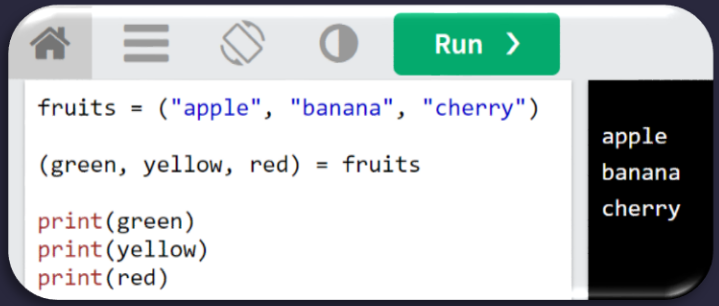
But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":

Example

Unpacking a tuple:

```
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

Try it Yourself »



Note: The number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list.

/PYTHON TUPLE – Using Asterisk*

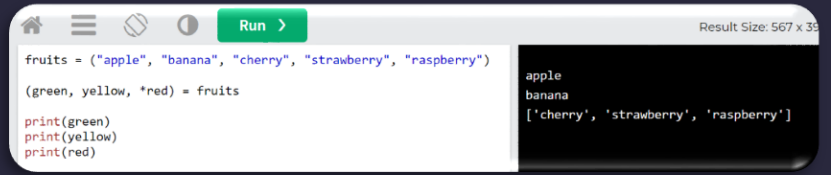
If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list:

Example

Assign the rest of the values as a list called "red":

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green)
print(yellow)
print(red)
```

Try it Yourself »



/PYTHON TUPLE – Using Asterisk*

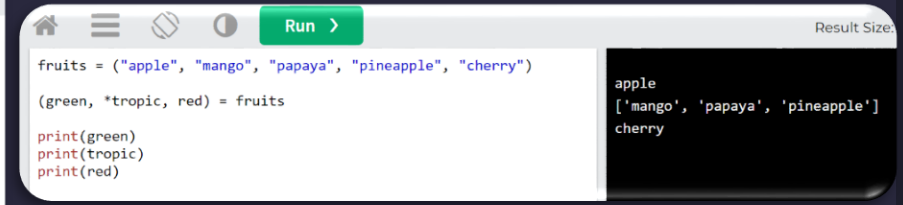
If the asterisk is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left.

Example

Add a list of values the "tropic" variable:

```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")

(green, *tropic, red) = fruits

print(green)
print(tropic)
print(red)
```

Try it Yourself »



<Loop Tuples>

/PYTHON TUPLE – Loop Through a Tuple

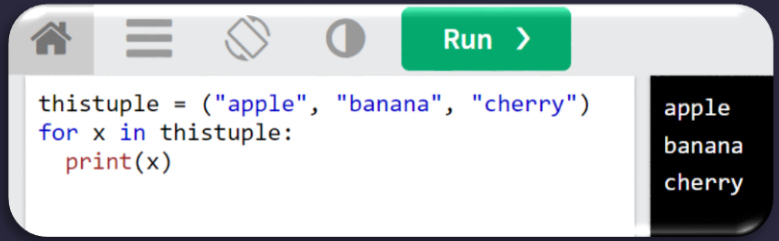
You can loop through the tuple items by using a for loop.

Example

Iterate through the items and print the values:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

Try it Yourself »



Learn more about for loops in our Python For Loops Chapter.

/PYTHON TUPLE – Loop Through the Index Numbers

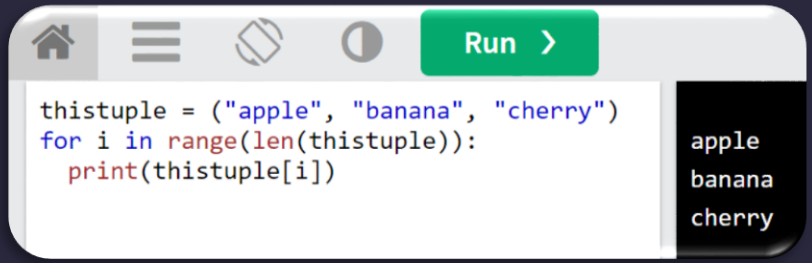
You can also loop through the tuple items by referring to their index number.
Use the **range()** and **len()** functions to create a suitable iterable.

Example

Print all items by referring to their index number:

```
thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
  print(thistuple[i])
```

Try it Yourself »



/PYTHON TUPLE – Using a While Loop

You can loop through the list items by using a **while** loop.

Use the **len()** function to determine the length of the tuple, then start at 0 and loop your way through the tuple items by refering to their indexes.

Remember to increase the index by 1 after each iteration.

Example

Print all items, using a `while` loop to go through all the index numbers:

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

Try it Yourself »

Learn more about while loops in our Python While Loops Chapter.

<Join Tuples>

/PYTHON TUPLE – JOIN TWO TUPLES

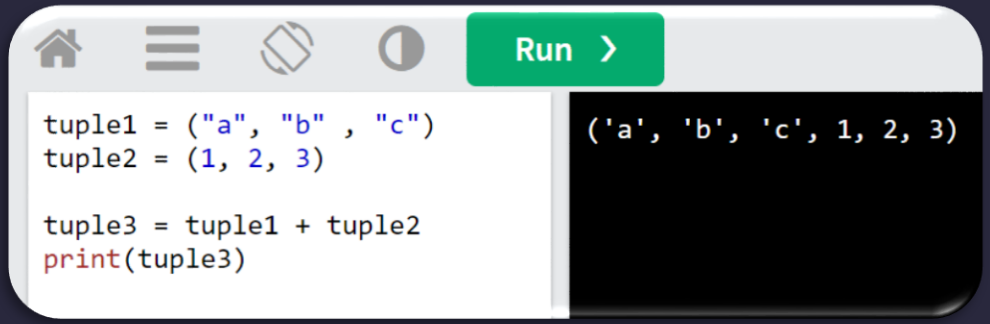
To join two or more tuples you can use the + operator:

Example

Join two tuples:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

Try it Yourself »



/PYTHON TUPLE – MULTIPLY TUPLES

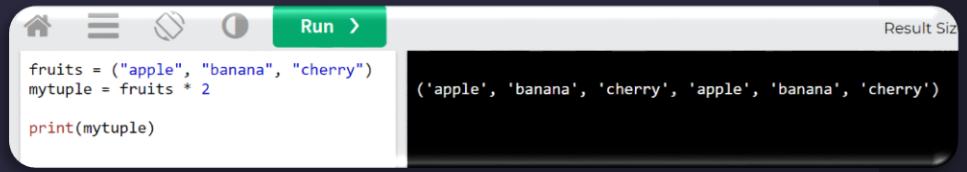
If you want to multiply the content of a tuple a given number of times, you can use the * operator:

Example

Multiply the fruits tuple by 2:

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

Try it Yourself »



<Tuple Methods>

/PYTHON TUPLES – TUPLE METHODS

Python has two built-in methods that you can use on tuples.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

<PYTHON SETS>

/PYTHON SETS – `myset = {"apple", "banana", "cherry"}`

Set

- Sets are used to store multiple items in a single variable.
- Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.
- A set is a collection which is unordered, unchangeable*, and unindexed.

*** Note:** Set items are unchangeable, but you can remove items and add new items.

/PYTHON SETS

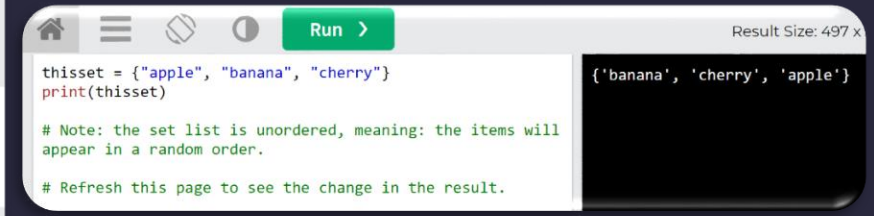
Sets are written with curly brackets.

Example

Create a Set:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

Try it Yourself »



Note: Sets are unordered, so you cannot be sure in which order the items will appear.

/PYTHON SETS

Set Items

Set items are unordered, unchangeable, and do not allow duplicate values.

Unordered

Unordered means that the items in a set do not have a defined order.

Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Unchangeable

Set items are unchangeable, meaning that we cannot change the items after the set has been created.

* Once a set is created, you cannot change its items, but you can remove items and add new items.

/PYTHON SETS

Duplicates Not Allowed

Sets cannot have two items with the same value.

Example

Duplicate values will be ignored:

```
thisset = {"apple", "banana", "cherry", "apple"}

print(thisset)
```

Try it Yourself »

/PYTHON SETS - Set Items - Data Types

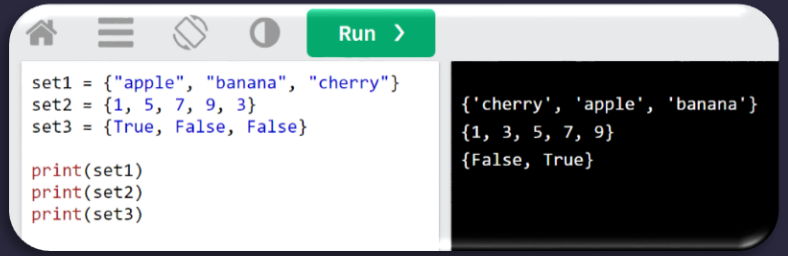
Set items can be of any data type:

Example

String, int and boolean data types:

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
```

Try it Yourself »



/PYTHON SETS - Set Items - Data Types

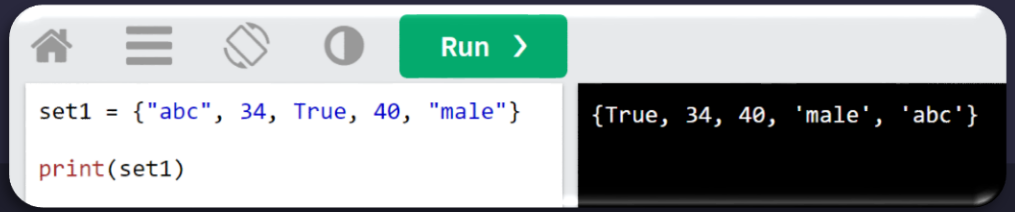
A set can contain different data types:

Example

A set with strings, integers and boolean values:

```
set1 = {"abc", 34, True, 40, "male"}
```

Try it Yourself »



/PYTHON SETS – type()

From Python's perspective, sets are defined as objects with the data type 'set':

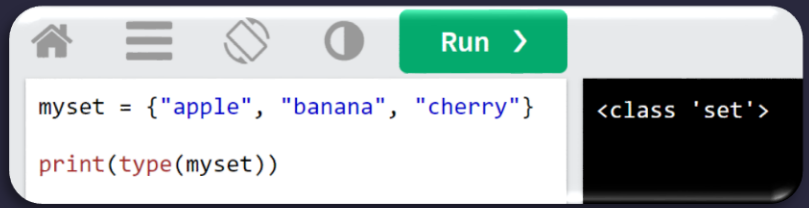
```
<class 'set'>
```

Example

What is the data type of a set?

```
myset = {"apple", "banana", "cherry"}
print(type(myset))
```

Try it Yourself »



/PYTHON SETS – The set() Constructor

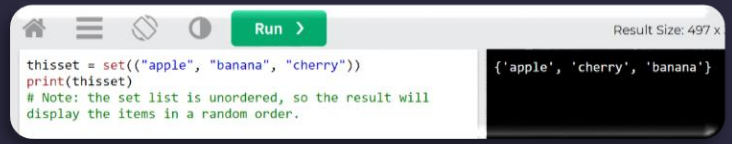
It is also possible to use the **set()** constructor to make a set.

Example

Using the set() constructor to make a set:

```
thisset = set(("apple", "banana", "cherry")) # note the double
round-brackets
print(thisset)
```

Try it Yourself »



<Access Set Items>

/PYTHON SETS – Access Items

You cannot access items in a set by referring to an index or a key.

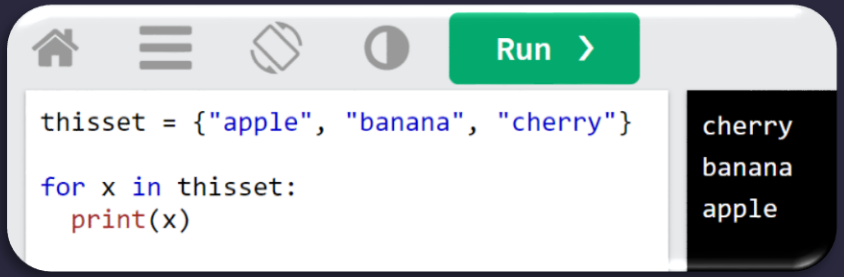
But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Example

Loop through the set, and print the values:

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

Try it Yourself »



/PYTHON SETS – Access Items

You cannot access items in a set by referring to an index or a key.

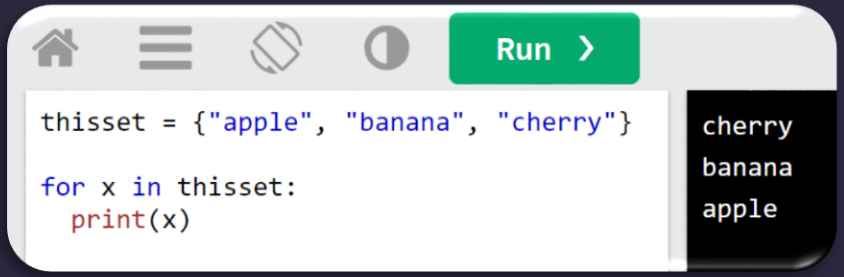
But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Example

Loop through the set, and print the values:

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

Try it Yourself »



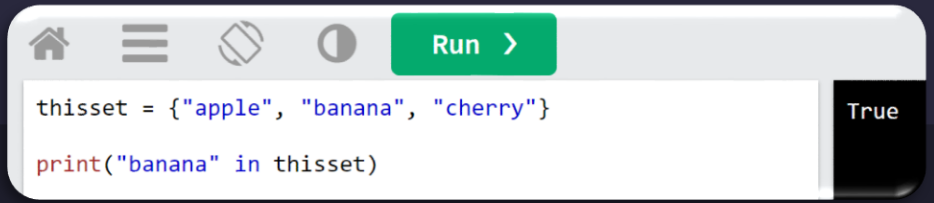
/PYTHON SETS – Access Items

Example

Check if "banana" is present in the set:

```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

Try it Yourself »



<Add Set Items>

/PYTHON SETS – Add Items

Once a set is created, you cannot change its items, but you can add new items.

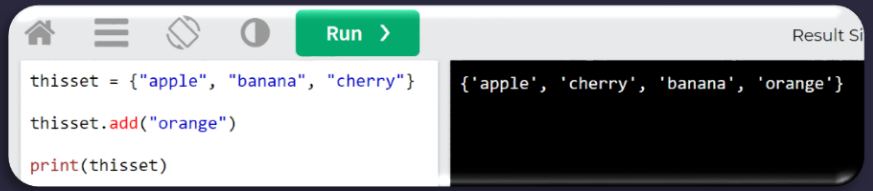
To add one item to a set use the add() method.

Example

Add an item to a set, using the `add()` method:

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

Try it Yourself »



/PYTHON SETS – Add Sets

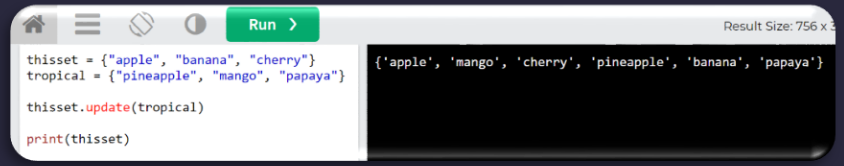
To add items from another set into the current set, use the update() method.

Example

Add elements from `tropical` into `thisset`:

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)

print(thisset)
```

Try it Yourself »



/PYTHON SETS – Add Any Iterable

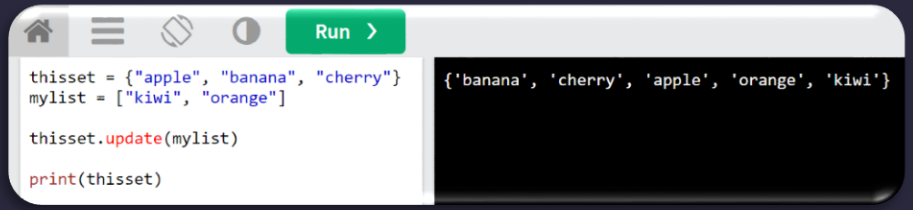
The object in the **update()** method does not have to be a set, it can be any iterable object (tuples, lists, dictionaries etc.).

Example

Add elements of a list to at set:

```
thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]

thisset.update(mylist)

print(thisset)
```

Try it Yourself »



<Remove Set Items>

/PYTHON SETS – Remove Set Items

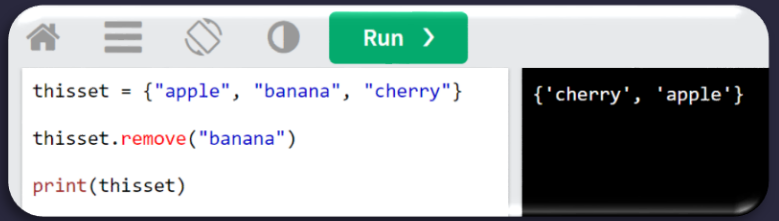
To remove an item in a set, use the remove(), or the discard() method.

Example

Remove "banana" by using the `remove()` method:

```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

Try it Yourself »



/PYTHON SETS – Remove Set Items

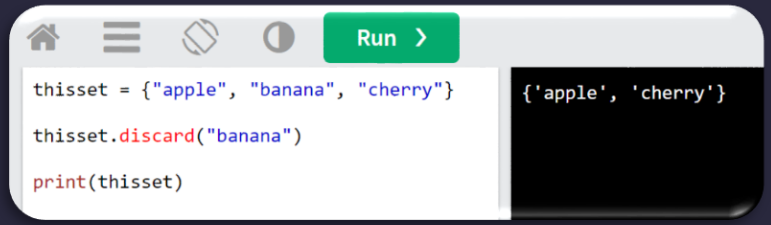
Note: If the item to remove does not exist, **remove()** will raise an error.

Example

Remove "banana" by using the `discard()` method:

```
thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)
```

Try it Yourself »



/PYTHON SETS – Remove Set Items

Note: If the item to remove does not exist, **discard()** will NOT raise an error.

You can also use the **pop()** method to remove an item, but this method will remove the last item. Remember that sets are unordered, so you will not know what item that gets removed.

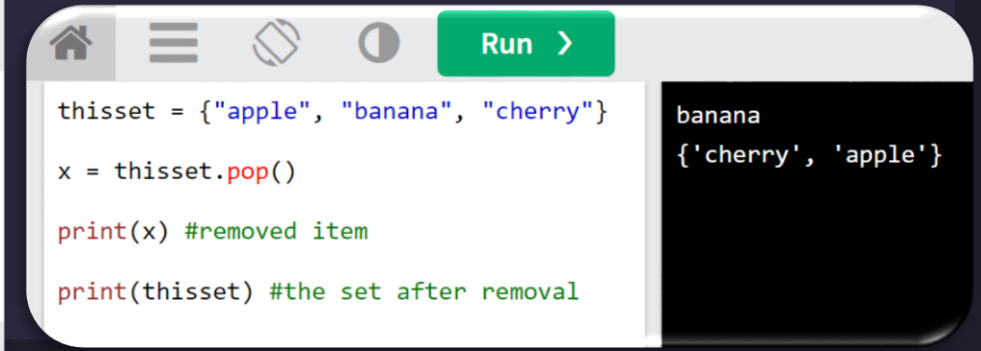
The return value of the **pop()** method is the removed item.

Example

Remove the last item by using the pop() method:

```
thisset = {"apple", "banana", "cherry"}

x = thisset.pop()

print(x)

print(thisset)
```

Try it Yourself »



/PYTHON SETS – Remove Set Items

Note: Sets are ***unordered***, so when using the **pop()** method, you do not know which item that gets removed.

Example

The `clear()` method empties the set:

```
thisset = {"apple", "banana", "cherry"}

thisset.clear()

print(thisset)
```

Try it Yourself »

Run ›

```
thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)
```

```
{'apple', 'cherry'}
```

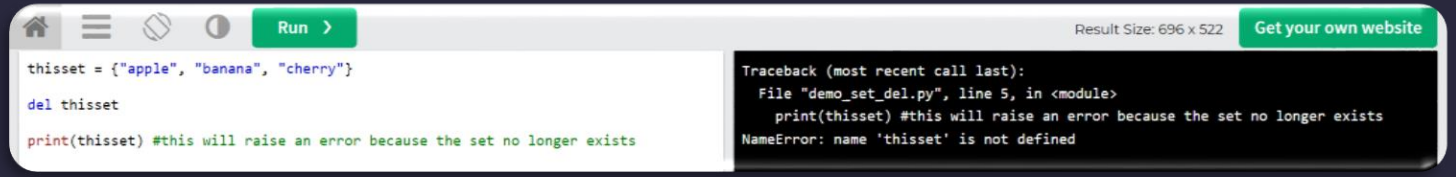
/PYTHON SETS – Remove Set Items

Example

The `del` keyword will delete the set completely:

```
thisset = {"apple", "banana", "cherry"}

del thisset

print(thisset)
```

Try it Yourself »



<Loop Sets>

/PYTHON SETS – Loop Items

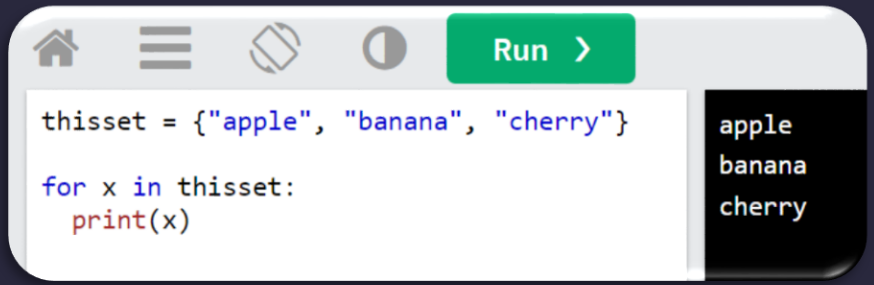
You can loop through the set items by using a **for** loop:

Example

Loop through the set, and print the values:

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

Try it Yourself »



<Join Sets>

/PYTHON SETS – Join Two Sets

There are several ways to join two or more sets in Python.

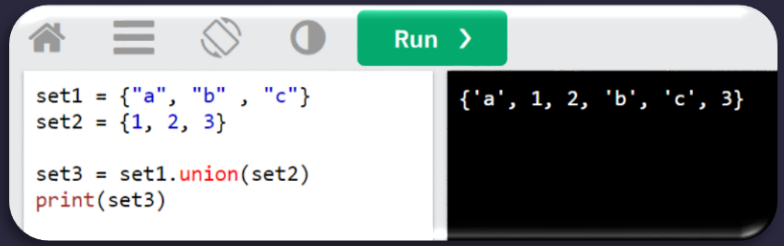
You can use the **union()** method that returns a new set containing all items from both sets, or the **update()** method that inserts all the items from one set into another:

Example

The union() method returns a new set with all items from both sets:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

Try it Yourself »



/PYTHON SETS – Join Two Sets

Example

The `update()` method inserts the items in set2 into set1:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```

Try it Yourself »

Note: Both **union()** and **update()** will exclude any duplicate items.

/PYTHON SETS – Keep ONLY the Duplicates

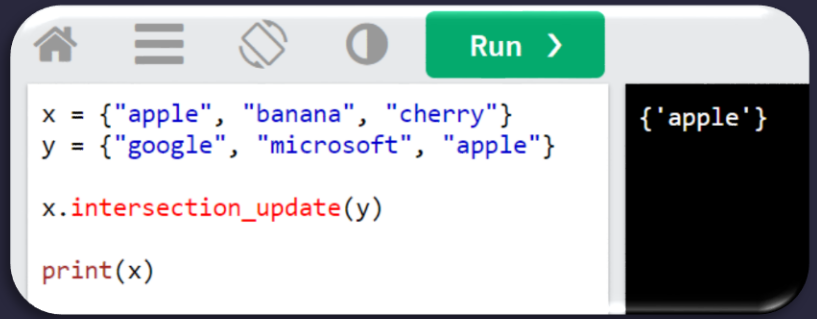
The **intersection_update()** method will keep only the items that are present in both sets.

Example

Keep the items that exist in both set x, and set y:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.intersection_update(y)

print(x)
```

Try it Yourself »



/PYTHON SETS – Keep ONLY the Duplicates

The **intersection()** method will return a new set, that only contains the items that are present in both sets.

Example

Return a set that contains the items that exist in both set x, and set y:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.intersection(y)

print(z)
```

Try it Yourself »

/PYTHON SETS – Keep All, But NOT the Duplicates

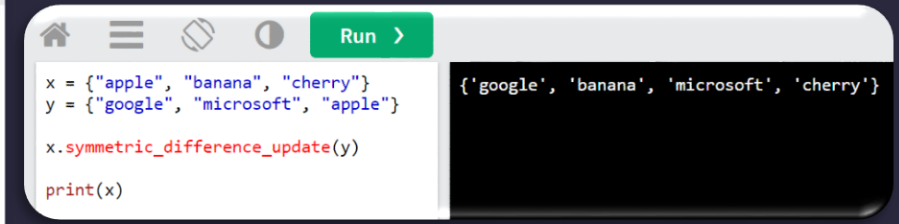
The **symmetric_difference_update()** method will keep only the elements that are NOT present in both sets.

Example

Keep the items that are not present in both sets:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.symmetric_difference_update(y)

print(x)
```

Try it Yourself »



/PYTHON SETS – Keep All, But NOT the Duplicates

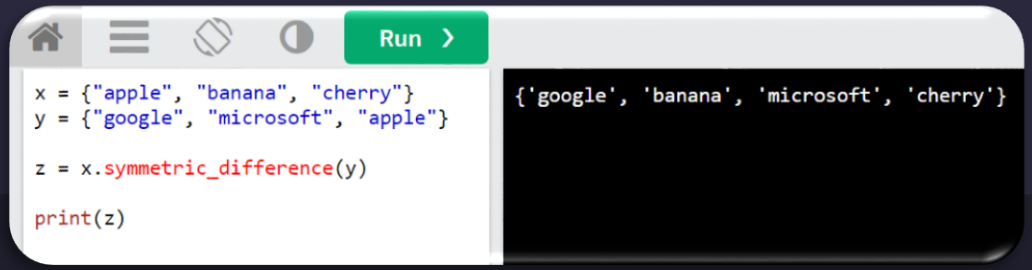
The **symmetric_difference()** method will return a new set, that contains only the elements that are NOT present in both sets.

Example

Return a set that contains all items from both sets, except items that are present in both:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.symmetric_difference(y)

print(z)
```

Try it Yourself »



/PYTHON SETS – Set Methods

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

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not

/PYTHON SETS – Set Methods

pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

/03

<WHY LEARN PYTHON>

/WHY TO LEARN PYTHON

Python is a very popular general-purpose interpreted, interactive, 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

/WHY TO LEARN PYTHON

Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning Python:

- **Python is Interpreted** - Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** - You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** - Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** - Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

/05

<Python Online Compiler/Interpreter>

/ONLINE COMPILER/INTERPRETER

We have provided **Python Online Compiler/Interpreter** which helps you to **Edit** and **Execute** the code directly from your browser.

JUST CLICK HERE:
1. Choice One
2. Choice Two

/06

<GOOGLE COLLAB>

/07

<PYTHON CODE>