PYTHON

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

What is python (programming)?

Python is a powerful high-level language, object-oriented programming language.

Has simple easy-to-use syntax,making it the perfect language for someone trying to learn computer programming for the first time.

Python is an interpreted language like:ABC(ABC has simple easy-to understand syntax)

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.

**FEATURES**

A simple language which is easier to learn

Free and open-source

Expressive Language

Interpreted Language

Object-Oriented Language

Oops concept

GUI(graphical user Interface) interface

Extensible and Embeddable

Dynamic memory allocation

# **Python Comments**

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

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.

# **Python Variables**

## **Variables**

Variables are containers for storing data values.

## **Variable Names**

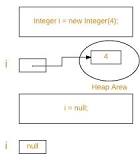
A variable can have a short name (like x and y) or a more descriptive name (age, carname, total\_volume). Rules for Python variables:

* 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)

**What is Python Pep 8?**

Pep 8—**a portable multi-language file preprocessor**. ================================================ There are tons of template systems. This is just a basic preprocessor. Pepe is usable both as a command line app and as a Python module.

**What is garbage collection with example?**



Garbage collection in Java is **the process by which Java programs perform automatic memory management**. Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program

**What does sep =' mean in Python?**

sep='' in the context of a function call **sets the named argument sep to an empty string**. See the print() function; sep is the separator used between multiple values when printing.

**What does end =' do in Python?**

The end parameter in the print function is used to **add any string**. At the end of the output of the print statement in python. By default, the print function ends with a newline. Passing the whitespace to the end parameter (end=' ') indicates that the end character has to be identified by whitespace and not a newline.

**Memory management in Python**

**involves the management of a private heap**. A private heap is a portion of memory that is exclusive to the Python process. All Python objects and data structures are stored in the private heap. The operating system cannot allocate this piece of memory to another process.

What are the two types of memory management in Python?

**There are two types of memory allocation in Python, static and dynamic.**

* Static memory. The stack data structure provides static memory allocation, meaning the variables are in the stack memory. ...
* Dynamic memory.

**What is the difference between static memory and dynamic memory?**

**Static Memory Allocation is done before program execution.** **Dynamic Memory Allocation is done during program execution**. In static memory allocation, once the memory is allocated, the memory size can’t change. In dynamic memory allocation, when memory is allocated the memory size can be changed

### 12. Dynamic Memory Allocation

In Python, we don't need to specify the data-type of the variable. When we assign some value to the variable, it automatically allocates the memory to the variable at run time. Suppose we are assigned integer value 15 to **x,** then we don't need to write **int x = 15.** Just write x = 15.

**PEP:**

PEP stands for **Python Enhancement Proposal**. A PEP is a technical design doc for the Python community which describes a new feature for the language itself, its processes, or its environment.

**Garbage Collection:**

A garbage collection in Python manages the memory automatically and heap allocation. In simpler terms, **the process of automatic deletion of unwanted or unused objects to free the memory** is called garbage collection in Python.

**What is end =' in Python?**

At the end of the output of the print statement in python. By default, the print function ends with a newline. Passing the whitespace to the end parameter (end=' ') **indicates that the end character has to be identified by whitespace and not a newline**

**What is SEP =' in Python?**

sep='' in the context of a function call **sets the named argument sep to an empty string**. See the print() function; sep is the separator used between multiple values when printing.

**Difference:**

Python end and sep parameters in the print function mainly differ due to their location. means where they are used in the function. **The end parameter is used to specify a string that will be printed after the output.** **However, the sep parameter is used as a separator between the items that you want to print**.

# Python Comment

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!")

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

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!")

## **Multiline Comments**

Python does not really have a syntax for multiline 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!")

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!")

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

## **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

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

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)

## **Casting**

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

### Example

x = str(3)    # x will be '3'  
y = int(3)    # y will be 3  
z = float(3)  # z will be 3.0

## **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))

You will learn more about [data types](https://www.w3schools.com/python/python_datatypes.asp) and [casting](https://www.w3schools.com/python/python_casting.asp) later in this tutorial.

## **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'

## **Case-Sensitive**

Variable names are case-sensitive.

### Example

This will create two variables:

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

# 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

Legal variable names:

myvar = "John"  
my\_var = "John"  
\_my\_var = "John"  
myVar = "John"  
MYVAR = "John"  
myvar2 = "John"

### Example

Illegal variable names:

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

Remember that variable names are case-sensitive

## **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)

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

## **One Value to Multiple Variables**

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

### Example

x = y = z = "Orange"  
print(x)  
print(y)  
print(z)

## **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)

# Python - Output Variables

## **Output Variables**

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

### Example

x = "Python is awesome"  
print(x)

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

### Example

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

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

### Example

x = "Python "  
y = "is "  
z = "awesome"  
print(x + y + z)

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)

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)

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)

# 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

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

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

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)

## **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 = "fantastic"  
  
myfunc()  
  
print("Python is " + x)

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

### Example

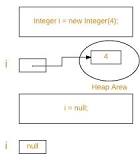
To change the value of a global variable inside a function, refer to the variable by using the global keyword:

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

**What is Python Pepe?**

pepe—**a portable multi-language file preprocessor**. ================================================ There are tons of template systems. This is just a basic preprocessor. Pepe is usable both as a command line app and as a Python module.

**What is garbage collection with example?**



Garbage collection in Java is **the process by which Java programs perform automatic memory management**. Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program

**What does sep =' mean in Python?**

sep='' in the context of a function call **sets the named argument sep to an empty string**. See the print() function; sep is the separator used between multiple values when printing.

**What does end =' do in Python?**

The end parameter in the print function is used to **add any string**. At the end of the output of the print statement in python. By default, the print function ends with a newline. Passing the whitespace to the end parameter (end=' ') indicates that the end character has to be identified by whitespace and not a newline.

**Memory management in Python**

**involves the management of a private heap**. A private heap is a portion of memory that is exclusive to the Python process. All Python objects and data structures are stored in the private heap. The operating system cannot allocate this piece of memory to another process.

**What are the two types of memory management in Python?**

**There are two types of memory allocation in Python, static and dynamic.**

* Static memory. The stack data structure provides static memory allocation, meaning the variables are in the stack memory. ...
* Dynamic memory.

**What is the difference between static memory and dynamic memory?**

**Static Memory Allocation is done before program execution.** **Dynamic Memory Allocation is done during program execution**. In static memory allocation, once the memory is allocated, the memory size can not change. In dynamic memory allocation, when memory is allocated the memory size can be changed.

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

## **Getting the Data 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))

## **Setting the Data Type**

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

|  |  |  |
| --- | --- | --- |
| **Example** | **Data Type** | **Try it** |
| x = "Hello World" | str | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_str) |
| x = 20 | int | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_int) |
| x = 20.5 | float | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_float) |
| x = 1j | complex | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_complex) |
| x = ["apple", "banana", "cherry"] | list | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_list) |
| x = ("apple", "banana", "cherry") | tuple | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_tuple) |
| x = range(6) | range | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_range) |
| x = {"name" : "John", "age" : 36} | dict | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_dict) |
| x = {"apple", "banana", "cherry"} | set | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_set) |
| x = frozenset({"apple", "banana", "cherry"}) | frozenset | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_frozenset) |
| x = True | bool | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bool) |
| x = b"Hello" | bytes | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytes) |
| x = bytearray(5) | bytearray | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytearray) |
| x = memoryview(bytes(5)) | memoryview | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_memoryview) |
| x = None | NoneType | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_nonetype) |

## **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** |
| x = str("Hello World") | str | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_str2) |
| x = int(20) | int | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_int2) |
| x = float(20.5) | float | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_float2) |
| x = complex(1j) | complex | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_complex2) |
| x = list(("apple", "banana", "cherry")) | list | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_list2) |
| x = tuple(("apple", "banana", "cherry")) | tuple | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_tuple2) |
| x = range(6) | range | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_range2) |
| x = dict(name="John", age=36) | dict | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_dict2) |
| x = set(("apple", "banana", "cherry")) | set | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_set2) |
| x = frozenset(("apple", "banana", "cherry")) | frozenset | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_frozenset2) |
| x = bool(5) | bool | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bool2) |
| x = bytes(5) | bytes | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytes2) |
| x = bytearray(5) | bytearray | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytearray2) |
| x = memoryview(bytes(5)) | memoryview | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_memoryview2) |

Top of Form

## **Test Yourself With Exercises**

## **Exercise:**

The following code example would print the data type of x, what data type would that be?

x = 5

print(type(x))

Bottom of Form

# 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:

### Example

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

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

### Example

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

## **Int**

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

### Example

Integers:

x = 1  
y = 35656222554887711  
z = -3255522  
  
print(type(x))  
print(type(y))  
print(type(z))

## **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))

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))

## **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))

## **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 = 1    # int  
y = 2.8  # float  
z = 1j   # 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))

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

## **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))

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

### 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

### 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'

# Python Booleans

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.

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)

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")

## **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))

### Example

Evaluate two variables:

x = "Hello"  
y = 15  
  
print(bool(x))  
print(bool(y))

## **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.

### Example

The following will return True:

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

## **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({})

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))

## **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())

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!")

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))

# Python Dictionaries

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}

## **Dictionary**

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered\*, changeable and do not allow duplicates.

As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.

Dictionaries are written with curly brackets, and have keys and values:

### Example

Create and print a dictionary:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(thisdict)

## **Dictionary Items**

Dictionary items are ordered, changeable, and does not allow duplicates.

Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

### Example

Print the "brand" value of the dictionary:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(thisdict["brand"])

## **Ordered or Unordered?**

As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.

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

Unordered means that the items does not have a defined order, you cannot refer to an item by using an index.

## **Changeable**

Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

## **Duplicates Not Allowed**

Dictionaries cannot have two items with the same key:

### Example

Duplicate values will overwrite existing values:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964,  
  "year": 2020  
}  
print(thisdict)

## **Dictionary Length**

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

### Example

Print the number of items in the dictionary:

print(len(thisdict))

## **Dictionary Items - Data Types**

The values in dictionary items can be of any data type:

### Example

String, int, boolean, and list data types:

thisdict = {  
  "brand": "Ford",  
  "electric": False,  
  "year": 1964,  
  "colors": ["red", "white", "blue"]  
}

## **type()**

From Python's perspective, dictionaries are defined as objects with the data type 'dict':

<class 'dict'>

### Example

Print the data type of a dictionary:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(type(thisdict))

## **The dict() Constructor**

It is also possible to use the dict() constructor to make a dictionary.

### Example

Using the dict() method to make a dictionary:

thisdict = dict(name = "John", age = 36, country = "Norway")  
print(thisdict)

## **Python Collections (Arrays)**

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

* [**List**](https://www.w3schools.com/python/python_lists.asp) is a collection which is ordered and changeable. Allows duplicate members.
* [**Tuple**](https://www.w3schools.com/python/python_tuples.asp) is a collection which is ordered and unchangeable. Allows duplicate members.
* [**Set**](https://www.w3schools.com/python/python_sets.asp) 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 - Access Dictionary Items

## **Accessing Items**

You can access the items of a dictionary by referring to its key name, inside square brackets:

### Example

Get the value of the "model" key:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
x = thisdict["model"]

There is also a method called get() that will give you the same result:

### Example

Get the value of the "model" key:

x = thisdict.get("model")

## **Get Keys**

The keys() method will return a list of all the keys in the dictionary.

### Example

Get a list of the keys:

x = thisdict.keys()

The list of the keys is a view of the dictionary, meaning that any changes done to the dictionary will be reflected in the keys list.

### Example

Add a new item to the original dictionary, and see that the keys list gets updated as well:

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
  
x = car.keys()  
  
print(x) #before the change  
  
car["color"] = "white"  
  
print(x) #after the change

## **Get Values**

The values() method will return a list of all the values in the dictionary.

### Example

Get a list of the values:

x = thisdict.values()

The list of the values is a view of the dictionary, meaning that any changes done to the dictionary will be reflected in the values list.

### Example

Make a change in the original dictionary, and see that the values list gets updated as well:

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
  
x = car.values()  
  
print(x) #before the change  
  
car["year"] = 2020  
  
print(x) #after the change

### Example

Add a new item to the original dictionary, and see that the values list gets updated as well:

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
  
x = car.values()  
  
print(x) #before the change  
  
car["color"] = "red"  
  
print(x) #after the change

## **Get Items**

The items() method will return each item in a dictionary, as tuples in a list.

### Example

Get a list of the key:value pairs

x = thisdict.items()

The returned list is a view of the items of the dictionary, meaning that any changes done to the dictionary will be reflected in the items list.

### Example

Make a change in the original dictionary, and see that the items list gets updated as well:

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
  
x = car.items()  
  
print(x) #before the change  
  
car["year"] = 2020  
  
print(x) #after the change

### Example

Add a new item to the original dictionary, and see that the items list gets updated as well:

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
  
x = car.items()  
  
print(x) #before the change  
  
car["color"] = "red"  
  
print(x) #after the change

## **Check if Key Exists**

To determine if a specified key is present in a dictionary use the in keyword:

### Example

Check if "model" is present in the dictionary:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
if "model" in thisdict:  
  print("Yes, 'model' is one of the keys in the thisdict dictionary")

# Python - Change Dictionary Items

## **Change Values**

You can change the value of a specific item by referring to its key name:

### Example

Change the "year" to 2018:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict["year"] = 2018

## **Update Dictionary**

The update() method will update the dictionary with the items from the given argument.

The argument must be a dictionary, or an iterable object with key:value pairs.

### Example

Update the "year" of the car by using the update() method:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.update({"year": 2020})

# Python - Add Dictionary Items

## **Adding Items**

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

### Example

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict["color"] = "red"  
print(thisdict)

## **Update Dictionary**

The update() method will update the dictionary with the items from a given argument. If the item does not exist, the item will be added.

The argument must be a dictionary, or an iterable object with key:value pairs.

### Example

Add a color item to the dictionary by using the update() method:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.update({"color": "red"})

# Python - Remove Dictionary Items

## **Removing Items**

There are several methods to remove items from a dictionary:

### Example

The pop() method removes the item with the specified key name:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.pop("model")  
print(thisdict)

### Example

The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead):

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.popitem()  
print(thisdict)

### Example

The del keyword removes the item with the specified key name:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
del thisdict["model"]  
print(thisdict)

### Example

The del keyword can also delete the dictionary completely:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
del thisdict  
print(thisdict) #this will cause an error because "thisdict" no longer exists.

### Example

The clear() method empties the dictionary:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.clear()  
print(thisdict)

# Python - Loop Dictionaries

## **Loop Through a Dictionary**

You can loop through a dictionary by using a for loop.

When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

### Example

Print all key names in the dictionary, one by one:

for x in thisdict:  
  print(x)

### Example

Print all values in the dictionary, one by one:

for x in thisdict:  
  print(thisdict[x])

### Example

You can also use the values() method to return values of a dictionary:

for x in thisdict.values():  
  print(x)

### Example

You can use the keys() method to return the keys of a dictionary:

for x in thisdict.keys():  
  print(x)

# Python - Copy Dictionaries

## **Copy a Dictionary**

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.

There are ways to make a copy, one way is to use the built-in Dictionary method copy().

### Example

Make a copy of a dictionary with the copy() method:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
mydict = thisdict.copy()  
print(mydict)

Another way to make a copy is to use the built-in function dict().

### Example

Make a copy of a dictionary with the dict() function:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
mydict = dict(thisdict)  
print(mydict)

# Python - Nested Dictionaries

## **Nested Dictionaries**

A dictionary can contain dictionaries, this is called nested dictionaries.

### Example

Create a dictionary that contain three dictionaries:

myfamily = {  
  "child1" : {  
    "name" : "Emil",  
    "year" : 2004  
  },  
  "child2" : {  
    "name" : "Tobias",  
    "year" : 2007  
  },  
  "child3" : {  
    "name" : "Linus",  
    "year" : 2011  
  }  
}

Or, if you want to add three dictionaries into a new dictionary:

### Example

Create three dictionaries, then create one dictionary that will contain the other three dictionaries:

child1 = {  
  "name" : "Emil",  
  "year" : 2004  
}  
child2 = {  
  "name" : "Tobias",  
  "year" : 2007  
}  
child3 = {  
  "name" : "Linus",  
  "year" : 2011  
}  
  
myfamily = {  
  "child1" : child1,  
  "child2" : child2,  
  "child3" : child3  
}

## **Access Items in Nested Dictionaries**

To access items from a nested dictionary, you use the name of the dictionaries, starting with the outer dictionary:

### Example

Print the name of child 2:

print(myfamily["child2"]["name"])

# Python Dictionary Methods

## **Dictionary Methods**

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

|  |  |
| --- | --- |
| **Method** | **Description** |
| [clear()](https://www.w3schools.com/python/ref_dictionary_clear.asp) | Removes all the elements from the dictionary |
| [copy()](https://www.w3schools.com/python/ref_dictionary_copy.asp) | Returns a copy of the dictionary |
| [fromkeys()](https://www.w3schools.com/python/ref_dictionary_fromkeys.asp) | Returns a dictionary with the specified keys and value |
| [get()](https://www.w3schools.com/python/ref_dictionary_get.asp) | Returns the value of the specified key |
| [items()](https://www.w3schools.com/python/ref_dictionary_items.asp) | Returns a list containing a tuple for each key value pair |
| [keys()](https://www.w3schools.com/python/ref_dictionary_keys.asp) | Returns a list containing the dictionary's keys |
| [pop()](https://www.w3schools.com/python/ref_dictionary_pop.asp) | Removes the element with the specified key |
| [popitem()](https://www.w3schools.com/python/ref_dictionary_popitem.asp) | Removes the last inserted key-value pair |
| [setdefault()](https://www.w3schools.com/python/ref_dictionary_setdefault.asp) | Returns the value of the specified key. If the key does not exist: insert the key, with the specified value |
| [update()](https://www.w3schools.com/python/ref_dictionary_update.asp) | Updates the dictionary with the specified key-value pairs |
| [values()](https://www.w3schools.com/python/ref_dictionary_values.asp) | Returns a list of all the values in the dictionary |

## **Python Operators**

Operators are used to perform operations on variables and values.

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

### Example

print(10 + 5)

Python divides the operators in the following groups:

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

## **Python Arithmetic Operators**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| + | Addition | x + y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_add) |
| - | Subtraction | x - y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_sub) |
| \* | Multiplication | x \* y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_mult) |
| / | Division | x / y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_div) |
| % | Modulus | x % y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_mod) |
| \*\* | Exponentiation | x \*\* y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_exp) |
| // | Floor division | x // y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_floordiv) |

## **Python Assignment Operators**

Assignment operators are used to assign values to variables:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Example** | **Same As** | **Try it** |
| = | x = 5 | x = 5 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass1) |
| += | x += 3 | x = x + 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass2) |
| -= | x -= 3 | x = x - 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass3) |
| \*= | x \*= 3 | x = x \* 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass4) |
| /= | x /= 3 | x = x / 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass5) |
| %= | x %= 3 | x = x % 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass6) |
| //= | x //= 3 | x = x // 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass7) |
| \*\*= | x \*\*= 3 | x = x \*\* 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass8) |
| &= | x &= 3 | x = x & 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass9) |
| |= | x |= 3 | x = x | 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass10) |
| ^= | x ^= 3 | x = x ^ 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass11) |
| >>= | x >>= 3 | x = x >> 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass12) |
| <<= | x <<= 3 | x = x << 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass13) |

## **Python Comparison Operators**

Comparison operators are used to compare two values:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| == | Equal | x == y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare1) |
| != | Not equal | x != y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare2) |
| > | Greater than | x > y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare4) |
| < | Less than | x < y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare5) |
| >= | Greater than or equal to | x >= y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare6) |
| <= | Less than or equal to | x <= y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare7) |

## **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 »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_logical1) |
| Or | Returns True if one of the statements is true | x < 5 or x < 4 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_logical2) |
| not | Reverse the result, returns False if the result is true | not(x < 5 and x < 10) | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_logical3) |

## **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 »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_identity1) |
| is not | Returns True if both variables are not the same object | x is not y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_identity2) |

## **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 »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_membership1) |
| not in | Returns True if a sequence with the specified value is not present in the object | x not in y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_membership2) |

## **Python Bitwise Operators**

Bitwise operators are used to compare (binary) numbers:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | **Description** | **Example** | **Try it** |
| & | AND | Sets each bit to 1 if both bits are 1 | x & y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_and) |
| | | OR | Sets each bit to 1 if one of two bits is 1 | x | y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_or) |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 | x ^ y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_xor) |
| ~ | NOT | Inverts all the bits | ~x | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_not) |
| << | Zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off | x << 2 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_left_shift) |
| >> | Signed right shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off | x >> 2 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_right_shift) |

## **Operator Precedence**

Operator precedence describes the order in which operations are performed.

### Example

Parentheses has the highest precedence, meaning that expressions inside parentheses must be evaluated first:

print((6 + 3) - (6 + 3))

### Example

Multiplication \* has higher precedence than addition +, and therefor multiplications are evaluated before additions:

print(100 + 5 \* 3)

The precedence order is described in the table below, starting with the highest precedence at the top:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Try it** |
| () | Parentheses | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_parentheses) |
| \*\* | Exponentiation | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_exponent) |
| +x  -x  ~x | Unary plus, unary minus, and bitwise NOT | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_bitwise_not) |
| \*  /  //  % | Multiplication, division, floor division, and modulus | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_multiplication) |
| +  - | Addition and subtraction | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_subtraction) |
| <<  >> | Bitwise left and right shifts | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_shift) |
| & | Bitwise AND | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_bitwise_and) |
| ^ | Bitwise XOR | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_bitwise_xor) |
| | | Bitwise OR | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_bitwise_or) |
| ==  !=  >  >=  <  <=  is  is not  in  not in | Comparisons, identity, and membership operators | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_like) |
| Not | Logical NOT | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_not) |
| And | AND | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_and) |
| Or | OR | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_or) |

If two operators have the same precedence, the expression is evaluated from left to right.

### Example

Addition + and subtraction - has the same precedence, and therefor we evaluate the expression from left to right:

print(5 + 4 - 7 + 3)

# Python Lists

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](https://www.w3schools.com/python/python_tuples.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

Lists are created using square brackets:

### Example

Create a List:

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

## **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](https://www.w3schools.com/python/python_lists_methods.asp) 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.

## **Allow Duplicates**

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

### Example

Lists allow duplicate values:

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

## **List Length**

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

### Example

Print the number of items in the list:

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

## **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]

A list can contain different data types:

### Example

A list with strings, integers and boolean values:

list1 = ["abc", 34, True, 40, "male"]

## **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))

## **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)

## **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**](https://www.w3schools.com/python/python_tuples.asp) is a collection which is ordered and unchangeable. Allows duplicate members.
* [**Set**](https://www.w3schools.com/python/python_sets.asp) is a collection which is unordered, unchangeable\*, and unindexed. No duplicate members.
* [**Dictionary**](https://www.w3schools.com/python/python_dictionaries.asp) is a collection which is ordered\*\* and changeable. No duplicate members.

\*Set items are unchangeable, but you can remove and/or add items whenever you like.

\*\*As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

# Python Tuples

mytuple = ("apple", "banana", "cherry")

## **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](https://www.w3schools.com/python/python_lists.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

A tuple is a collection which is ordered and **unchangeable**.

Tuples are written with round brackets.

### Example

Create a Tuple:

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

## **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.

## **Allow Duplicates**

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

### Example

Tuples allow duplicate values:

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

## **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))

## **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))

## **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)

A tuple can contain different data types:

### Example

A tuple with strings, integers and boolean values:

tuple1 = ("abc", 34, True, 40, "male")

## **type()**

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

<class 'tuple'>

### Example

What is the data type of a tuple?

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

## **The tuple() Constructor**

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

### Example

Using the tuple() method to make a tuple:

thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets  
print(thistuple)

## **Python Collections (Arrays)**

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

* [**List**](https://www.w3schools.com/python/python_lists.asp) is a collection which is ordered and changeable. Allows duplicate members.
* **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
* [**Set**](https://www.w3schools.com/python/python_sets.asp) is a collection which is unordered, unchangeable\*, and unindexed. No duplicate members.
* [**Dictionary**](https://www.w3schools.com/python/python_dictionaries.asp) is a collection which is ordered\*\* and changeable. No duplicate members.

\*Set items are unchangeable, but you can remove and/or add items whenever you like.

\*\*As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

# 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](https://www.w3schools.com/python/python_lists.asp), [Tuple](https://www.w3schools.com/python/python_tuples.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), 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.

Sets are written with curly brackets.

### Example

Create a Set:

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

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

## **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.

## **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)

**Note:** The values True and 1 are considered the same value in sets, and are treated as duplicates:

### Example

True and 1 is considered the same value:

thisset = {"apple", "banana", "cherry", True, 1, 2}  
  
print(thisset)

## **Get the Length of a Set**

To determine how many items a set has, use the len() function.

### Example

Get the number of items in a set:

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

## **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}

A set can contain different data types:

### Example

A set with strings, integers and boolean values:

set1 = {"abc", 34, True, 40, "male"}

## **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))

## **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)

## **Python Collections (Arrays)**

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

* [**List**](https://www.w3schools.com/python/python_lists.asp) is a collection which is ordered and changeable. Allows duplicate members.
* [**Tuple**](https://www.w3schools.com/python/python_tuples.asp) 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**](https://www.w3schools.com/python/python_dictionaries.asp) is a collection which is ordered\*\* and changeable. No duplicate members.

# 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')

## **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)

## **Multiline Strings**

You can assign a multiline string to a variable by using three 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)

Or three single quotes:

### Example

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

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

## **Strings are Arrays**

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

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

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

### Example

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

a = "Hello, World!"  
print(a[1])

## **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 "banana":

for x in "banana":  
  print(x)

Learn more about For Loops in our [Python For Loops](https://www.w3schools.com/python/python_for_loops.asp) chapter.

## **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, World!"  
print(len(a))

## **Check String**

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

### Example

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

txt = "The best things in life are free!"  
print("free" in txt)

Use it in an if statement:

### Example

Print only if "free" is present:

txt = "The best things in life are free!"  
if "free" in txt:  
  print("Yes, 'free' is present.")

Learn more about If statements in our [Python If...Else](https://www.w3schools.com/python/python_conditions.asp) chapter.

## **Check if NOT**

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

### Example

Check if "expensive" is NOT present in the following text:

txt = "The best things in life are free!"  
print("expensive" not in txt)

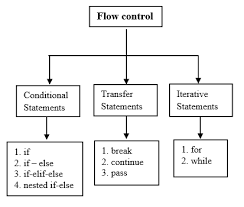
Use it in an if statement:

### Example

print only if "expensive" is NOT present:

txt = "The best things in life are free!"  
if "expensive" not in txt:  
  print("No, 'expensive' is NOT present.")

**Control flow statement**

****

# Control Statements in Python

Loops are employed in Python to iterate over a section of code continually. Control statements are designed to serve the purpose of modifying a loop's execution from its default behaviour. Based on a condition, control statements are applied to alter how the loop executes. In this tutorial, we are covering every type of control statement that exists in Python.

## **Control Statements in Python**

### if Statements:

The if statement is arguably the most used statement to control loops. For instance:

The if statement is a conditional statement in python, that is used to determine whether a block of code will be executed or not. Meaning if the program finds the condition defined in the if statement to be true, it will go ahead and execute the code block inside the if statement.

if condition:

*# execute code block*

## **if-else Statement:**

As discussed above, the if statement executes the code block when the condition is true. Similarly, the else statement works in conjuncture with the if statement to execute a code block when the defined if condition is false.

Syntax:

Syntax:

**if** condition:

*# execute code if condition is true*

**else**:

*# execute code if condition if False*

**Elif:**

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.

1. **if** expression 1:
2. # block of statements
4. **elif** expression 2:
5. # block of statements
7. **elif** expression 3:
8. # block of statements
10. **else**:
11. # block of statements

**Nested-if Statement:**

We can have an if…elif…else statement inside another if…elif…else statement. This is called nesting in computer programming. Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. This can get confusing, so it must be avoided if we can.

**Syntax:**

if (condition1):

# Executes when condition1 is true

if (condition2):

# Executes when condition2 is true

# if Block is end here

# if Block is end here

# Python While Loops

In coding, loops are designed to execute a specified code block repeatedly. We'll learn how to construct a while loop in Python, the syntax of a while loop, loop controls like break and continue, and other exercises in this tutorial.

## **Introduction of Python While Loop**

The Python while loop iteration of a code block is executed as long as the given condition, i.e., conditional\_expression, is true.

If we don't know how many times we'll execute the iteration ahead of time, we can write an indefinite loop.

**Syntax of Python While Loop**

1. **while** conditional\_expression:
2. Code block of **while**

The given condition, i.e., conditional\_expression, is evaluated initially in the Python while loop. Then, if the conditional expression gives a boolean value True, the while loop statements are executed. The conditional expression is verified again when the complete code block is executed. This procedure repeatedly occurs until the conditional expression returns the boolean value False.

* The statements of the Python while loop are dictated by indentation.
* The code block begins when a statement is indented & ends with the very first unindented statement.
* Any non-zero number in Python is interpreted as boolean True. False is interpreted as None and 0.

##### **Transfer Statements**

Transfer statements alter the way a logic gets executed. These statements are often used in loops [for](https://www.waytoeasylearn.com/learn/iteration-statements/) and [while](https://www.waytoeasylearn.com/learn/iteration-statements/). Let’s have a look at the transfer statements below.

###### **1. break**

We can use break statement inside loops to break loop execution based on some condition.

###### **2. continue**

We can use continue statement to skip current [iteration](https://www.waytoeasylearn.com/learn/iteration-statements/) and continue next iteration

###### **3. pass statement**

* pass is a keyword in Python.
* In our programming syntactically if block is required which won’t do anything then we can define that empty block with pass keyword.
* This statement does nothing. It is used to define an empty block of code or a class. When written in a loop statement, it’s usually the last statement.

**Data structure in python:**

The basic Python data structures in Python include list, set, tuples, and dictionary. Each of the data structures is unique in its own way. Data structures are “containers” that organize and group data according to type. The data structures differ based on mutability and order.

1. Lists are mutable, meaning that after creation, we can modify their elements.
2. It consists of data separated by commas inside square braces-[].
3. Lists allow duplicate elements.
4. We use **indexing** to access the list elements **(0 to length - 1)**, and negative indexing is allowed too **(-1 to -length).**
5. We can nest another data structure as an element inside a list. Nesting another list makes it a **multi-dimensional list**.
6. Using **Slicing**, we can get sub-lists of a list for which we need to use the slice operator - [:].
7. **List comprehension** is used to create new lists from existing lists or other data structures. It is faster than creating a list using the regular for loop.
8. We can use functions like max(), min() and sum() to find the maximum, minimum and sum of the elements in the list.
9. We can concatenate lists using + and repeat elements using \* operators.
10. To take a list as an input, we use input() and then the split(separator) to convert the string into a list.

|  |  |
| --- | --- |
| **1. List.len()** | To find the length of the list |
| **2. List.append()** | Add an element at the end of the list. |
| **3. List.insert()** | To add an element at a specified index. |
| **4. List.remove()** | To remove a specified element from the list. |
| **5. List.reverse()** | Returns the reverse ordered list. |
| **6. List.index()** | Returns the first-found index of the specified element. |
| **7. List.extend()** | Adds the elements of another list as elements to the specified list. |
| **8. List.pop()** | Removes the returns of the last element of the specified list. |
| **9. String.split()** | Converts the given string into a list |

### Tuples:

A Tuple is like an **immutable list**. It is also **a sequence data type** in Python and can store data of different data types, like a list, but unlike a list, we cannot alter a tuple once created; Python raises an error if we try.

**Important points about Tuples:**

1. A tuple consists of data separated by commas inside the parenthesis, although **parenthesis is not mandatory** but followed for readability in convention.
2. Creating a tuple without parenthesis is called "**Tuple packing**" and to access a tuple, we can also use "**Tuple unpacking**".
3. Tuples allow duplicate elements like lists.
4. We use indexing to access the tuple elements **(0 to length - 1)**, and negative indexing is allowed too **(-1 to -length).**
5. We can nest another data structure as an element inside a tuple. Nesting another tuple makes it a multi-dimensional tuple.
6. Using **Slicing,** we can get sub-tuples of a tuple for which we need to use the slice operator - [:].
7. We can concatenate two tuples, repeat, reassign or delete the whole tuple but once created, we can't perform operations like append, remove, insert elements or delete elements which means we cannot disturb the original elements from when created-**Tuples are immutable**.
8. To perform all the operations we can't perform on a tuple, we generally convert a tuple into a list using list(), do what we want and then convert it back to a tuple using tuple().
9. As a tuple is immutable, it can be used as a **key in dictionaries**.
10. Iterating through a tuple is much faster than iterating through a list.

### Most used Functions and Methods:

|  |  |
| --- | --- |
| **1. Tuple.len()** | To find the length of the tuple |
| **2. sorted(Tuple)** | Sorts the specified tuple |
| **3. max(Tuple)** | To find the maximum valued element in the tuple. |
| **4. min(Tuple)** | To find the minimum valued element in the tuple. |
| **5. sum(Tuple)** | To find the sum of the elements in the tuple. |
| **6. Tuple.index()** | Returns the first-found index of the specified element. |
| **7. all(Tuple)** | Returns True if all the elements in the tuple are True. |
| **8. any(Tuple)** | Returns true if atleast one element in the tuple is True. |
| **9. Tuple.count(element)** | Returns the number of occurrences of the specified element in the tuple. |

* len(), sorted(), max(), min(), sum(), all(), any() work on any iterable data type in Python.
* Do not confuse sort() with sorted(). sort() is limited to lists.
* A Python set is the collection of the unordered items. Each element in the set must be unique, immutable, and the sets remove the duplicate elements. Sets are mutable which means we can modify it after its creation.
* Unlike other collections in Python, there is no index attached to the elements of the set, i.e., we cannot directly access any element of the set by the index. However, we can print them all together, or we can get the list of elements by looping through the set.

## **Creating a set**

* The set can be created by enclosing the comma-separated immutable items with the curly braces {}. Python also provides the set() method, which can be used to create the set by the passed sequence.

|  |  |
| --- | --- |
| **Method** | Description |
| [**add()**](https://www.programiz.com/python-programming/methods/set/add) | Adds an element to the set |
| [**clear()**](https://www.programiz.com/python-programming/methods/set/clear) | Removes all elements from the set |
| [**copy()**](https://www.programiz.com/python-programming/methods/set/copy) | Returns a copy of the set |
| [**difference()**](https://www.programiz.com/python-programming/methods/set/difference) | Returns the difference of two or more sets as a new set |
| [**difference\_update()**](https://www.programiz.com/python-programming/methods/set/difference_update) | Removes all elements of another set from this set |
| [**discard()**](https://www.programiz.com/python-programming/methods/set/discard) | Removes an element from the set if it is a member. (Do nothing if the element is not in set) |
| [**intersection()**](https://www.programiz.com/python-programming/methods/set/intersection) | Returns the intersection of two sets as a new set |
| [**intersection\_update()**](https://www.programiz.com/python-programming/methods/set/intersection_update) | Updates the set with the intersection of itself and another |
| [**isdisjoint()**](https://www.programiz.com/python-programming/methods/set/isdisjoint) | Returns True if two sets have a null intersection |
| [**issubset()**](https://www.programiz.com/python-programming/methods/set/issubset) | Returns True if another set contains this set |
| [**issuperset()**](https://www.programiz.com/python-programming/methods/set/issuperset) | Returns True if this set contains another set |
| [**pop()**](https://www.programiz.com/python-programming/methods/set/pop) | Removes and returns an arbitrary set element. Raises KeyError if the set is empty |
| [**remove()**](https://www.programiz.com/python-programming/methods/set/remove) | Removes an element from the set. If the element is not a member, raises a KeyError |
| [**symmetric\_difference()**](https://www.programiz.com/python-programming/methods/set/symmetric_difference) | Returns the symmetric difference of two sets as a new set |
| [**symmetric\_difference\_update()**](https://www.programiz.com/python-programming/methods/set/symmetric_difference_update) | Updates a set with the symmetric difference of itself and another |
| [**union()**](https://www.programiz.com/python-programming/methods/set/union) | Returns the union of sets in a new set |
| **[update()](https://www.programiz.com/python-programming/methods/set/update)** | Updates the set with the union of itself and others |