Python Tutorial
Learn Python
Python is a popular programming language.

Python can be used on a server to create web applications.

Learning by Examples
With our "Try it Yourself" editor, you can edit Python code and view the result.

ExampleGet your own Python Server
print("Hello, World!")
Click on the "Try it Yourself" button to see how it works.

Python File Handling
In our File Handling section you will learn how to open, read, write, and delete files.

Python File Handling

Python Database Handling
In our database section you will learn how to access and work with MySQL and MongoDB databases:

Python MySQL Tutorial

Python MongoDB Tutorial

Python Exercises
Test Yourself With Exercises
Exercise:
Insert the missing part of the code below to output "Hello World".

("Hello World")

Start the Exercise

Python Examples
Learn by examples! This tutorial supplements all explanations with clarifying examples.

Python Quiz
Test your Python skills with a quiz.

My Learning
Track your progress with the free "My Learning" program here at W3Schools.

Log in to your account, and start earning points!

This is an optional feature. You can study W3Schools without using My Learning.

Python Reference
You will also find complete function and method references:

Reference Overview

Built-in Functions

String Methods

List/Array Methods

Dictionary Methods

Tuple Methods

Set Methods

File Methods

Python Keywords

Python Exceptions

Python Glossary

Random Module

Requests Module

Math Module

CMath Module

Download Python
Download Python from the official Python web site: https://python.org

-----

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

It is used for:

web development (server-side),
software development,
mathematics,
system scripting.
What can Python do?
Python can be used on a server to create web applications.
Python can be used alongside software to create workflows.
Python can connect to database systems. It can also read and modify files.
Python can be used to handle big data and perform complex mathematics.
Python can be used for rapid prototyping, or for production-ready software development.
Why Python?
Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
Python has a simple syntax similar to the English language.
Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
Python can be treated in a procedural way, an object-oriented way or a functional way.
Good to know
The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

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

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

Python Getting Started
Python Install
Many PCs and Macs will have python already installed.

To check if you have python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe):

```
C:\Users\Your Name>python --version
```

To check if you have python installed on a Linux or Mac, then on linux open the command line or on Mac open the Terminal and type:

```
python --version
```

If you find that you do not have Python installed on your computer, then you can download it for free from the following website: https://www.python.org/

Python Quickstart
Python is an interpreted programming language, this means that as a developer you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed.

The way to run a python file is like this on the command line:

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

Where "helloworld.py" is the name of your python file.

Let's write our first Python file, called helloworld.py, which can be done in any text editor.

helloworld.py

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

Simple as that. Save your file. Open your command line, navigate to the directory where you saved your file, and run:

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

The output should read:

```
Hello, World!
```

Congratulations, you have written and executed your first Python program.

The Python Command Line
To test a short amount of code in python sometimes it is quickest and easiest not to write the code in a file. This is made possible because Python can be run as a command line itself.

Type the following on the Windows, Mac or Linux command line:

C:\Users\Your Name>python
Or, if the "python" command did not work, you can try "py":
C:\Users\Your Name>py
From there you can write any python, including our hello world example from earlier in the tutorial:

```
C:\Users\Your Name>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
```

Which will write "Hello, World!" in the command line:

```
C:\Users\Your Name>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
Hello, World!
```

Whenever you are done in the python command line, you can simply type the following to quit the python command line interface:

```
exit()
```

Python Syntax
Execute Python Syntax
As we learned in the previous page, Python syntax can be executed by writing directly in the Command Line:

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

On this page
Execute Python Syntax
Python Indentation
Python Variables
Python Comments
Exercises
Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

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

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

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

Python uses indentation to indicate a block of code.

ExampleGet your own Python Server

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

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

Example
Syntax Error:

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

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

Example

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

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

Example
Syntax Error:

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


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

Example
Variables in Python:

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

Python has no command for declaring a variable.

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

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

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

Example
Comments in Python:

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

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:

ExampleGet your own Python Server

```
#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!")
```

ADVERTISEMENT

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.

ExampleGet your own Python Server

```
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
ADVERTISEMENT

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 and casting 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)
A variable name cannot be any of the Python keywords.
ExampleGet your own Python Server
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

ADVERTISEMENT

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:

ExampleGet your own Python Server

```
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)
```

Learn more about unpacking in our Unpack Tuples Chapter.

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

ExampleGet your own Python Server

```
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.

ExampleGet your own Python Server

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)
```
ADVERTISEMENT

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)
```

Python - Variable Exercises
Test Yourself With Exercises
Now you have learned a lot about variables, and how to use them in Python.

Are you ready for a test?

Try to insert the missing part to make the code work as expected:

Exercise:
Create a variable named carname and assign the value Volvo to it.

```
 = ""
```

Go to the Exercise section and test all of our Python Variable Exercises:

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:

ExampleGet your own Python Server
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	
x = 20	int	
x = 20.5	float	
x = 1j	complex	
x = ["apple", "banana", "cherry"]	list	
x = ("apple", "banana", "cherry")	tuple	
x = range(6)	range	
x = {"name" : "John", "age" : 36}	dict	
x = {"apple", "banana", "cherry"}	set	
x = frozenset({"apple", "banana", "cherry"})	frozenset	
x = True	bool	
x = b"Hello"	bytes	
x = bytearray(5)	bytearray	
x = memoryview(bytes(5))	memoryview	
x = None	NoneType	

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	
x = int(20)	int	
x = float(20.5)	float	
x = complex(1j)	complex	
x = list(("apple", "banana", "cherry"))	list	
x = tuple(("apple", "banana", "cherry"))	tuple	
x = range(6)	range	
x = dict(name="John", age=36)	dict	
x = set(("apple", "banana", "cherry"))	set	
x = frozenset(("apple", "banana", "cherry"))	frozenset	
x = bool(5)	bool	
x = bytes(5)	bytes	
x = bytearray(5)	bytearray	
x = memoryview(bytes(5))	memoryview	

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))

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:

ExampleGet your own Python Server
```
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))
```

ADVERTISEMENT

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))
```

In our Random Module Reference you will learn more about the Random module.

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
ExampleGet your own Python Server
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 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:

ExampleGet your own Python Server
```
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 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 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.")
```

Python - Slicing Strings
Slicing
You can return a range of characters by using the slice syntax.

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

ExampleGet your own Python Server
Get the characters from position 2 to position 5 (not included):

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

Note: The first character has index 0.

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

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

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

ADVERTISEMENT

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:])
```

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])
```

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

Upper Case
ExampleGet your own Python Server
The upper() method returns the string in upper case:

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

Lower Case
Example
The lower() method returns the string in lower case:

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

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

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

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


Replace String
Example
The replace() method replaces a string with another string:

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

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!']
```

Learn more about Lists in our Python Lists chapter.

String Methods
Learn more about String Methods with our String Methods Reference

-----

Python - String Concatenation
String Concatenation
To concatenate, or combine, two strings you can use the + operator.

ExampleGet your own Python Server
Merge variable a with variable b into variable c:

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

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

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

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

ExampleGet your own Python Server
```
age = 36
txt = "My name is John, I am " + age
print(txt)
```
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))
```
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))
```
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))
```
Learn more about String Formatting in our String Formatting chapter.

Python - Escape Characters
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:

ExampleGet your own Python Server
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."
```

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

Example
The escape character allows you to use double quotes when you normally would not be allowed:

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

Escape Characters
Other escape characters used in Python:

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

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

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

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
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
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value

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

-----

Python - String Exercises
Test Yourself With Exercises
Now you have learned a lot about Strings, and how to use them in Python.

Are you ready for a test?

Try to insert the missing part to make the code work as expected:

Test Yourself With Exercises
Exercise:
Use the len function to print the length of the string.

```
x = "Hello World"
print()
```

Go to the Exercise section and test all of our Python Strings Exercises:

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:

ExampleGet your own Python Server
```
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))
```
ADVERTISEMENT

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 Operators
Python Operators
Operators are used to perform operations on variables and values.

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

ExampleGet your own Python Server
```
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	
-	Subtraction	x - y	
*	Multiplication	x * y	
/	Division	x / y	
%	Modulus	x % y	
**	Exponentiation	x ** y	
//	Floor division	x // y	

Python Assignment Operators

Assignment operators are used to assign values to variables:

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

Python Comparison Operators
Comparison operators are used to compare two values:

Operator Name Example Try it
== Equal x == y
!= Not equal x != y
> Greater than x > y
< Less than x < y
>= Greater than or equal to x >= y
<= Less than or equal to x <= y
Python Logical Operators
Logical operators are used to combine conditional statements:

Operator Description Example Try it
and Returns True if both statements are true x < 5 and x < 10
or Returns True if one of the statements is true x < 5 or x < 4
not Reverse the result, returns False if the result is true not(x < 5 and x
< 10)
Python Identity Operators
Identity operators are used to compare the objects, not if they are equal, but
if they are actually the same object, with the same memory location:

Operator Description Example Try it
is Returns True if both variables are the same object x is y
is not Returns True if both variables are not the same object x is not
y
Python Membership Operators
Membership operators are used to test if a sequence is presented in an object:

Operator Description Example Try it
in Returns True if a sequence with the specified value is present in the
object x in y
not in Returns True if a sequence with the specified value is not present
in the object x not in y
Python Bitwise Operators
Bitwise operators are used to compare (binary) numbers:

Operator Name Description Example Try it
& AND Sets each bit to 1 if both bits are 1 x & y
| OR Sets each bit to 1 if one of two bits is 1 x | y
^ XOR Sets each bit to 1 if only one of two bits is 1x ^ y
~ NOT Inverts all the bits ~x
<< Zero fill left shift Shift left by pushing zeros in from the right and
let the leftmost bits fall off x << 2
>> Signed right shift Shift right by pushing copies of the leftmost bit
in from the left, and let the rightmost bits fall offx >> 2

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	
**	Exponentiation	
+x -x ~x	Unary plus, unary minus, and bitwise NOT	
* / // %	Multiplication, division, floor division, and modulus	
+ -	Addition and subtraction	
<< >>	Bitwise left and right shifts	
&	Bitwise AND	
^	Bitwise XOR	
|	Bitwise OR	
== != > >= < <= is is not in not in	Comparisons, identity, and membership operators	
not	Logical NOT	
and	AND	
or	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, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets:

ExampleGet your own Python Server
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 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 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.
*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 - Access List Items
Access Items
List items are indexed and you can access them by referring to the index number:

ExampleGet your own Python Server
Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```
Note: The first item has index 0.

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 list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```
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 list with the specified items.

Example
Return the third, fourth, and fifth item:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```
Note: The search will start at index 2 (included) and end at index 5 (not included).

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 including, "kiwi":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```
By leaving out the end value, the range will go on to the end of the list:

Example
This example returns the items from "cherry" to the end:

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

Range of Negative Indexes
Specify negative indexes if you want to start the search from the end of the list:

Example
This example returns the items from "orange" (-4) to, but NOT including "mango" (-1):

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```
Check if Item Exists
To determine if a specified item is present in a list use the in keyword:

Example
Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

Python - Change List Items
Change Item Value
To change the value of a specific item, refer to the index number:

ExampleGet your own Python Server
Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```
Change a Range of Item Values
To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values:

Example
Change the values "banana" and "cherry" with the values "blackcurrant" and "watermelon":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```
If you insert more items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

Example
Change the second value by replacing it with two new values:

```
thislist = ["apple", "banana", "cherry"]
```

```
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)
```
Note: The length of the list will change when the number of items inserted does not match the number of items replaced.

If you insert less items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

Example
Change the second and third value by replacing it with one value:

```
thislist = ["apple", "banana", "cherry"]
thislist[1:3] = ["watermelon"]
print(thislist)
```
ADVERTISEMENT

Insert Items
To insert a new list item, without replacing any of the existing values, we can use the insert() method.

The insert() method inserts an item at the specified index:

Example
Insert "watermelon" as the third item:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```
Note: As a result of the example above, the list will now contain 4 items.

Python - Add List Items
Append Items
To add an item to the end of the list, use the append() method:

ExampleGet your own Python Server
Using the append() method to append an item:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```
Insert Items
To insert a list item at a specified index, use the insert() method.

The insert() method inserts an item at the specified index:

Example
Insert an item as the second position:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```
Note: As a result of the examples above, the lists will now contain 4 items.

ADVERTISEMENT

Extend List
To append elements from another list to the current list, use the extend() method.

Example
Add the elements of tropical to thislist:

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```
The elements will be added to the end of the list.

Add Any Iterable
The extend() method does not have to append lists, you can add any iterable object (tuples, sets, dictionaries etc.).

Example
Add elements of a tuple to a list:

```
thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")
thislist.extend(thistuple)
print(thislist)
```

Python - Remove List Items
Remove Specified Item
The remove() method removes the specified item.

ExampleGet your own Python Server
Remove "banana":

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```
If there are more than one item with the specified value, the remove() method removes the first occurance:

Example
Remove the first occurance of "banana":

```
thislist = ["apple", "banana", "cherry", "banana", "kiwi"]
thislist.remove("banana")
print(thislist)
```
Remove Specified Index
The pop() method removes the specified index.

Example
Remove the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```
If you do not specify the index, the pop() method removes the last item.

Example
Remove the last item:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```
The del keyword also removes the specified index:

Example
Remove the first item:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

The del keyword can also delete the list completely.

Example
Delete the entire list:

```
thislist = ["apple", "banana", "cherry"]
del thislist
```

Clear the List
The clear() method empties the list.

The list still remains, but it has no content.

Example
Clear the list content:

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

Python - Loop Lists
Loop Through a List
You can loop through the list items by using a for loop:

ExampleGet your own Python Server
Print all items in the list, one by one:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

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

Loop Through the Index Numbers
You can also loop through the list 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:

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

The iterable created in the example above is [0, 1, 2].

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 list, then start at 0 and loop your way through the list items by referring 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

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

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

Looping Using List Comprehension
List Comprehension offers the shortest syntax for looping through lists:

Example
A short hand for loop that will print all items in a list:

```
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```
Learn more about list comprehension in the next chapter: List Comprehension.

Python - List Comprehension
List Comprehension
List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Example:

Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

Without list comprehension you will have to write a for statement with a conditional test inside:

ExampleGet your own Python Server
```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)

print(newlist)
```
With list comprehension you can do all that with only one line of code:

Example
```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```


The Syntax
```
newlist = [expression for item in iterable if condition == True]
```
The return value is a new list, leaving the old list unchanged.

Condition
The condition is like a filter that only accepts the items that valuate to True.

Example
Only accept items that are not "apple":

```
newlist = [x for x in fruits if x != "apple"]
```
The condition if x != "apple" will return True for all elements other than "apple", making the new list contain all fruits except "apple".

The condition is optional and can be omitted:

Example
With no if statement:

newlist = [x for x in fruits]
Iterable
The iterable can be any iterable object, like a list, tuple, set etc.

Example
You can use the range() function to create an iterable:

newlist = [x for x in range(10)]
Same example, but with a condition:

Example
Accept only numbers lower than 5:

newlist = [x for x in range(10) if x < 5]
Expression
The expression is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list:

Example
Set the values in the new list to upper case:

newlist = [x.upper() for x in fruits]
You can set the outcome to whatever you like:

Example
Set all values in the new list to 'hello':

newlist = ['hello' for x in fruits]
The expression can also contain conditions, not like a filter, but as a way to manipulate the outcome:

Example
Return "orange" instead of "banana":

newlist = [x if x != "banana" else "orange" for x in fruits]
The expression in the example above says:

"Return the item if it is not banana, if it is banana return orange".

Python - Sort Lists
Sort List Alphanumerically
List objects have a sort() method that will sort the list alphanumerically, ascending, by default:

ExampleGet your own Python Server
Sort the list alphabetically:

thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
Example
Sort the list numerically:

thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)
Sort Descending
To sort descending, use the keyword argument reverse = True:

Example
Sort the list descending:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

Example
Sort the list descending:

```
thislist = [100, 50, 65, 82, 23]
thislist.sort(reverse = True)
print(thislist)
```


Customize Sort Function
You can also customize your own function by using the keyword argument key = function.

The function will return a number that will be used to sort the list (the lowest number first):

Example
Sort the list based on how close the number is to 50:

```
def myfunc(n):
  return abs(n - 50)

thislist = [100, 50, 65, 82, 23]
thislist.sort(key = myfunc)
print(thislist)
```

Case Insensitive Sort
By default the sort() method is case sensitive, resulting in all capital letters being sorted before lower case letters:

Example
Case sensitive sorting can give an unexpected result:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort()
print(thislist)
```

Luckily we can use built-in functions as key functions when sorting a list.

So if you want a case-insensitive sort function, use str.lower as a key function:

Example
Perform a case-insensitive sort of the list:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort(key = str.lower)
print(thislist)
```

Reverse Order
What if you want to reverse the order of a list, regardless of the alphabet?

The reverse() method reverses the current sorting order of the elements.

Example
Reverse the order of the list items:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

Python - Copy Lists

Copy a List
You cannot copy a list simply by typing list2 = list1, because: list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.

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

ExampleGet your own Python Server
Make a copy of a list with the copy() method:

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

Another way to make a copy is to use the built-in method list().

Example
Make a copy of a list with the list() method:

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

Python - Join Lists
Join Two Lists
There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.

ExampleGet your own Python Server
Join two list:

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

Another way to join two lists is by appending all the items from list2 into list1, one by one:

Example
Append list2 into list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

Or you can use the extend() method, where the purpose is to add elements from one list to another list:

Example
Use the extend() method to add list2 at the end of list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

Python - List Methods
List Methods
Python has a set of built-in methods that you can use on lists.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Python List Exercises
Test Yourself With Exercises
Now you have learned a lot about lists, and how to use them in Python.

Are you ready for a test?

Try to insert the missing part to make the code work as expected:

Exercise:
Print the second item in the fruits list.

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

Go to the Exercise section and test all of our Python List Exercises:

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, 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.

ExampleGet your own Python Server
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)
```

ADVERTISEMENT

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 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.
*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 - Access Tuple Items
Access Tuple Items
You can access tuple items by referring to the index number, inside square brackets:

ExampleGet your own Python Server
Print the second item in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```
Note: The first item has index 0.

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])
```
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])
```

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

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])
```

By leaving out the end value, the range will go on to the end of the tuple:

Example

This example returns the items from "cherry" and to the end:

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


Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the tuple:

Example

This example returns the items from index -4 (included) to index -1 (excluded)

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[-4:-1])
```

Check if Item Exists

To determine if a specified item is present in a tuple use the in keyword:

Example

Check if "apple" is present in the tuple:

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

Python - 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.

ExampleGet your own Python Server

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)
```

Add Items
Since tuples are immutable, they do not have a built-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.

Example
Convert the tuple into a list, add "orange", and convert it back into a tuple:

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

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)
```

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.

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)
```

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
```

Python - Unpack Tuples
Unpacking a Tuple
When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

ExampleGet your own Python Server
Packing a tuple:

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

ADVERTISEMENT

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)
```
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)
```

Python - Loop Tuples
Loop Through a Tuple
You can loop through the tuple items by using a for loop.

ExampleGet your own Python Server
Iterate through the items and print the values:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```
Learn more about for loops in our Python For Loops Chapter.

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])
```


Using a While Loop
You can loop through the tuple 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 referring 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
```

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

Python - Join Tuples
Join Two Tuples
To join two or more tuples you can use the + operator:

ExampleGet your own Python Server
Join two tuples:

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

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

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)
```

Python - Tuple Methods
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 - Tuple Exercises
Test Yourself With Exercises
Now you have learned a lot about tuples, and how to use them in Python.

Are you ready for a test?

Try to insert the missing part to make the code work as expected:

Exercise:
Print the first item in the fruits tuple.

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

Go to the Exercise section and test all of our Python Tuple Exercises:

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.

Sets are written with curly brackets.

ExampleGet your own Python Server
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)
```

Note: The values False and 0 are considered the same value in sets, and are treated as duplicates:

Example
False and 0 is considered the same value:

```
thisset = {"apple", "banana", "cherry", False, True, 0}

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

**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 - Access Set Items
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.

ExampleGet your own Python Server
Loop through the set, and print the values:

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

for x in thisset:
  print(x)
```
Example
Check if "banana" is present in the set:

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

print("banana" in thisset)
```
Change Items
Once a set is created, you cannot change its items, but you can add new items.

Python - Add Set Items
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.

ExampleGet your own Python Server
Add an item to a set, using the add() method:

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

thisset.add("orange")

print(thisset)
```
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)
```

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)
```

Python - Remove Set Items
Remove Item
To remove an item in a set, use the remove(), or the discard() method.

ExampleGet your own Python Server
Remove "banana" by using the remove() method:

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

thisset.remove("banana")

print(thisset)
```

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)
```

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 a random item, so you cannot be sure what item that gets removed.

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

Example
Remove a random item by using the pop() method:

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

x = thisset.pop()

print(x)
```

print(thisset)
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)
```
Example
The del keyword will delete the set completely:

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

del thisset

print(thisset)
```

Python - Loop Sets
Loop Items
You can loop through the set items by using a for loop:

ExampleGet your own Python Server
Loop through the set, and print the values:

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

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

Python - Join 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:

ExampleGet your own Python Server
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)
```
Example
The update() method inserts the items in set2 into set1:

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

set1.update(set2)
print(set1)
```
Note: Both union() and update() will exclude any duplicate items.

ADVERTISEMENT

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)
```

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)
```

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)
```

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)
```

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:

```
x = {"apple", "banana", "cherry", True}
y = {"google", 1, "apple", 2}

z = x.symmetric_difference(y)
```

print(z)

Python - Set Methods
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
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

-----

Python - Set Exercises
Test Yourself With Exercises
Now you have learned a lot about sets, and how to use them in Python.

Are you ready for a test?

Try to insert the missing part to make the code work as expected:

Exercise:
Check if "apple" is present in the fruits set.

```
fruits = {"apple", "banana", "cherry"}
if "apple"  fruits:
  print("Yes, apple is a fruit!")
```

Go to the Exercise section and test all of our Python Set Exercises:

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:

ExampleGet your own Python Server
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 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.
*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 - Access Dictionary Items
Accessing Items
You can access the items of a dictionary by referring to its key name, inside square brackets:

ExampleGet your own Python Server
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:

ExampleGet your own Python Server
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:

ExampleGet your own Python Server

```
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:

ExampleGet your own Python Server
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.

ExampleGet your own Python Server
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)
```

Example
Loop through both keys and values, by using the items() method:

```
for x, y in thisdict.items():
  print(x, y)
```

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().

ExampleGet your own Python Server
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.

ExampleGet your own Python Server
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()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Python Dictionary Exercises
Test Yourself With Exercises
Now you have learned a lot about dictionaries, and how to use them in Python.

Are you ready for a test?

Try to insert the missing part to make the code work as expected:

Test Yourself With Exercises
Exercise:
Use the get method to print the value of the "model" key of the car dictionary.

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

Start the Exercise

Go to the Exercise section and test all of our Python Dictionary Exercises:

Python If ... Else
Python Conditions and If statements
Python supports the usual logical conditions from mathematics:

Equals: a == b
Not Equals: a != b
Less than: a < b
Less than or equal to: a <= b
Greater than: a > b
Greater than or equal to: a >= b
These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the if keyword.

ExampleGet your own Python Server
If statement:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```
In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

Indentation
Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Example
If statement, without indentation (will raise an error):

```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```


Elif
The elif keyword is Python's way of saying "if the previous conditions were not true, then try this condition".

Example
```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```
In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

Else
The else keyword catches anything which isn't caught by the preceding conditions.

Example
```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```
In this example a is greater than b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b".

You can also have an else without the elif:

Example
```
a = 200
```

```
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

Example

One line if statement:

```
if a > b: print("a is greater than b")
```

Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

Example

One line if else statement:

```
a = 2
b = 330
print("A") if a > b else print("B")
```

This technique is known as Ternary Operators, or Conditional Expressions.

You can also have multiple else statements on the same line:

Example

One line if else statement, with 3 conditions:

```
a = 330
b = 330
print("A") if a > b else print("=") if a == b else print("B")
```

And

The and keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, AND if c is greater than a:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

Or

The or keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, OR if a is greater than c:

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

Not

The not keyword is a logical operator, and is used to reverse the result of the conditional statement:

Example

Test if a is NOT greater than b:

```
a = 33
b = 200
if not a > b:
  print("a is NOT greater than b")
```
Nested If
You can have if statements inside if statements, this is called nested if statements.

Example
```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```
The pass Statement
if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

Example
```
a = 33
b = 200

if b > a:
  pass
```

Python While Loops
Python Loops
Python has two primitive loop commands:

while loops
for loops
The while Loop
With the while loop we can execute a set of statements as long as a condition is true.

ExampleGet your own Python Server
Print i as long as i is less than 6:

```
i = 1
while i < 6:
  print(i)
  i += 1
```
Note: remember to increment i, or else the loop will continue forever.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

The break Statement
With the break statement we can stop the loop even if the while condition is true:

Example
Exit the loop when i is 3:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
```

```
  i += 1
```

ADVERTISEMENT

The continue Statement
With the continue statement we can stop the current iteration, and continue with the next:

Example
Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

The else Statement
With the else statement we can run a block of code once when the condition no longer is true:

Example
Print a message once the condition is false:

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

Python For Loops
Python For Loops
A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

ExampleGet your own Python Server
Print each fruit in a fruit list:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

The for loop does not require an indexing variable to set beforehand.

Looping Through a String
Even strings are iterable objects, they contain a sequence of characters:

Example
Loop through the letters in the word "banana":

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

The break Statement
With the break statement we can stop the loop before it has looped through all the items:

Example
Exit the loop when x is "banana":

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

Example
Exit the loop when x is "banana", but this time the break comes before the print:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

The continue Statement
With the continue statement we can stop the current iteration of the loop, and continue with the next:

Example
Do not print banana:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

The range() Function
To loop through a set of code a specified number of times, we can use the range() function,
The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example
Using the range() function:

```
for x in range(6):
  print(x)
```

Note that range(6) is not the values of 0 to 6, but the values 0 to 5.

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

Example
Using the start parameter:

```
for x in range(2, 6):
  print(x)
```

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3):

Example
Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):
  print(x)
```

Else in For Loop
The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

Example
Print all numbers from 0 to 5, and print a message when the loop has ended:

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```
Note: The else block will NOT be executed if the loop is stopped by a break statement.

Example
Break the loop when x is 3, and see what happens with the else block:

```
for x in range(6):
  if x == 3: break
  print(x)
else:
  print("Finally finished!")
```
Nested Loops
A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example
Print each adjective for every fruit:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```
The pass Statement
for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

Example
```
for x in [0, 1, 2]:
  pass
```

Python Functions
A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

Creating a Function
In Python a function is defined using the def keyword:

ExampleGet your own Python Server
```
def my_function():
  print("Hello from a function")
```
Calling a Function
To call a function, use the function name followed by parenthesis:

Example
```
def my_function():
  print("Hello from a function")

my_function()
```

Arguments
Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Example
```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```
Arguments are often shortened to args in Python documentations.

Parameters or Arguments?
The terms parameter and argument can be used for the same thing: information that are passed into a function.

From a function's perspective:

A parameter is the variable listed inside the parentheses in the function definition.

An argument is the value that is sent to the function when it is called.

Number of Arguments
By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

Example
This function expects 2 arguments, and gets 2 arguments:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```
If you try to call the function with 1 or 3 arguments, you will get an error:
Example
This function expects 2 arguments, but gets only 1:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil")
```
Arbitrary Arguments, *args
If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

This way the function will receive a tuple of arguments, and can access the items accordingly:

Example
If the number of arguments is unknown, add a * before the parameter name:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])
```

```
my_function("Emil", "Tobias", "Linus")
```

Arbitrary Arguments are often shortened to *args in Python documentations.

Keyword Arguments
You can also send arguments with the key = value syntax.

This way the order of the arguments does not matter.

Example
```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

The phrase Keyword Arguments are often shortened to kwargs in Python documentations.

Arbitrary Keyword Arguments, **kwargs
If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.

This way the function will receive a dictionary of arguments, and can access the items accordingly:

Example
If the number of keyword arguments is unknown, add a double ** before the parameter name:

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

Arbitrary Kword Arguments are often shortened to **kwargs in Python documentations.

Default Parameter Value
The following example shows how to use a default parameter value.

If we call the function without argument, it uses the default value:

Example
```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

Passing a List as an Argument
You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

E.g. if you send a List as an argument, it will still be a List when it reaches the function:

Example
```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]
```

my_function(fruits)
Return Values
To let a function return a value, use the return statement:

Example
```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```
The pass Statement
function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

Example
```
def myfunction():
  pass
```
Recursion
Python also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

In this example, tri_recursion() is a function that we have defined to call itself ("recurse"). We use the k variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).

To a new developer it can take some time to work out how exactly this works, best way to find out is by testing and modifying it.

Example
Recursion Example

```
def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("\n\nRecursion Example Results")
tri_recursion(6)
```

Python Lambda
A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

Syntax
```
lambda arguments : expression
```
The expression is executed and the result is returned:

ExampleGet your own Python Server
Add 10 to argument a, and return the result:

```
x = lambda a : a + 10
print(x(5))
```

Lambda functions can take any number of arguments:

Example
Multiply argument a with argument b and return the result:

```
x = lambda a, b : a * b
print(x(5, 6))
```

Example
Summarize argument a, b, and c and return the result:

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```


Why Use Lambda Functions?
The power of lambda is better shown when you use them as an anonymous function inside another function.

Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

```
def myfunc(n):
  return lambda a : a * n
```

Use that function definition to make a function that always doubles the number you send in:

Example

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

Or, use the same function definition to make a function that always triples the number you send in:

Example

```
def myfunc(n):
  return lambda a : a * n

mytripler = myfunc(3)

print(mytripler(11))
```

Or, use the same function definition to make both functions, in the same program:

Example

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

Use lambda functions when an anonymous function is required for a short period of time.

Python Arrays
Note: Python does not have built-in support for Arrays, but Python Lists can be used instead.

Arrays
Note: This page shows you how to use LISTS as ARRAYS, however, to work with arrays in Python you will have to import a library, like the NumPy library.

Arrays are used to store multiple values in one single variable:

ExampleGet your own Python Server
Create an array containing car names:

```
cars = ["Ford", "Volvo", "BMW"]
```
What is an Array?
An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"
```
However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Access the Elements of an Array
You refer to an array element by referring to the index number.

Example
Get the value of the first array item:

```
x = cars[0]
```
Example
Modify the value of the first array item:

```
cars[0] = "Toyota"
```
The Length of an Array
Use the len() method to return the length of an array (the number of elements in an array).

Example
Return the number of elements in the cars array:

```
x = len(cars)
```
Note: The length of an array is always one more than the highest array index.

Looping Array Elements
You can use the for in loop to loop through all the elements of an array.

Example
Print each item in the cars array:

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

Adding Array Elements
You can use the append() method to add an element to an array.

Example
Add one more element to the cars array:

```
cars.append("Honda")
```

Removing Array Elements
You can use the pop() method to remove an element from the array.

Example
Delete the second element of the cars array:

```
cars.pop(1)
```

You can also use the remove() method to remove an element from the array.

Example
Delete the element that has the value "Volvo":

```
cars.remove("Volvo")
```

Note: The list's remove() method only removes the first occurrence of the specified value.

Array Methods
Python has a set of built-in methods that you can use on lists/arrays.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Note: Python does not have built-in support for Arrays, but Python Lists can be used instead.

Python Classes and Objects
Python Classes/Objects
Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class
To create a class, use the keyword class:

ExampleGet your own Python Server
Create a class named MyClass, with a property named x:

```
class MyClass:
  x = 5
```

Create Object
Now we can use the class named MyClass to create objects:

Example

Create an object named p1, and print the value of x:

```
p1 = MyClass()
print(p1.x)
```

The __init__() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in __init__() function.

All classes have a function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Example

Create a class named Person, use the __init__() function to assign values for name and age:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Note: The __init__() function is called automatically every time the class is being used to create a new object.

The __str__() Function

The __str__() function controls what should be returned when the class object is represented as a string.

If the __str__() function is not set, the string representation of the object is returned:

Example

The string representation of an object WITHOUT the __str__() function:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1)
```

Example

The string representation of an object WITH the __str__() function:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def __str__(self):
    return f"{self.name}({self.age})"
```

```
p1 = Person("John", 36)

print(p1)
```

Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object.

Let us create a method in the Person class:

Example

Insert a function that prints a greeting, and execute it on the p1 object:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

Note: The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

The self Parameter

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:

Example

Use the words mysillyobject and abc instead of self:

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

Modify Object Properties

You can modify properties on objects like this:

Example

Set the age of p1 to 40:

```
p1.age = 40
```

Delete Object Properties

You can delete properties on objects by using the del keyword:

Example

Delete the age property from the p1 object:

```
del p1.age
```

Delete Objects

You can delete objects by using the del keyword:

Example

Delete the p1 object:

del p1
The pass Statement
class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error.

Example
```
class Person:
  pass
```

Python Inheritance
Python Inheritance
Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

Create a Parent Class
Any class can be a parent class, so the syntax is the same as creating any other class:

ExampleGet your own Python Server
Create a class named Person, with firstname and lastname properties, and a printname method:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

#Use the Person class to create an object, and then execute the printname
method:

x = Person("John", "Doe")
x.printname()
```
Create a Child Class
To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:

Example
Create a class named Student, which will inherit the properties and methods from the Person class:

```
class Student(Person):
  pass
```
Note: Use the pass keyword when you do not want to add any other properties or methods to the class.

Now the Student class has the same properties and methods as the Person class.

Example
Use the Student class to create an object, and then execute the printname method:

```
x = Student("Mike", "Olsen")
x.printname()
```

ADVERTISEMENT

Add the __init__() Function
So far we have created a child class that inherits the properties and methods from its parent.

We want to add the __init__() function to the child class (instead of the pass keyword).

Note: The __init__() function is called automatically every time the class is being used to create a new object.

Example
Add the __init__() function to the Student class:

```
class Student(Person):
  def __init__(self, fname, lname):
    #add properties etc.
```

When you add the __init__() function, the child class will no longer inherit the parent's __init__() function.

Note: The child's __init__() function overrides the inheritance of the parent's __init__() function.

To keep the inheritance of the parent's __init__() function, add a call to the parent's __init__() function:

Example

```
class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)
```

Now we have successfully added the __init__() function, and kept the inheritance of the parent class, and we are ready to add functionality in the __init__() function.

Use the super() Function
Python also has a super() function that will make the child class inherit all the methods and properties from its parent:

Example

```
class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
```

By using the super() function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent.

Add Properties
Example
Add a property called graduationyear to the Student class:

```
class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
    self.graduationyear = 2019
```

In the example below, the year 2019 should be a variable, and passed into the Student class when creating student objects. To do so, add another parameter in the __init__() function:

Example
Add a year parameter, and pass the correct year when creating objects:

```
class Student(Person):
  def __init__(self, fname, lname, year):
```

```
    super().__init__(fname, lname)
    self.graduationyear = year

x = Student("Mike", "Olsen", 2019)
```
Add Methods
Example
Add a method called welcome to the Student class:

```
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of",
self.graduationyear)
```
If you add a method in the child class with the same name as a function in the parent class, the inheritance of the parent method will be overridden.

Python Iterators
Python Iterators
An iterator is an object that contains a countable number of values.

An iterator is an object that can be iterated upon, meaning that you can traverse through all the values.

Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods __iter__() and __next__().

Iterator vs Iterable
Lists, tuples, dictionaries, and sets are all iterable objects. They are iterable containers which you can get an iterator from.

All these objects have a iter() method which is used to get an iterator:

ExampleGet your own Python Server
Return an iterator from a tuple, and print each value:

```
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```
Even strings are iterable objects, and can return an iterator:

Example
Strings are also iterable objects, containing a sequence of characters:

```
mystr = "banana"
myit = iter(mystr)

print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
```
Looping Through an Iterator
We can also use a for loop to iterate through an iterable object:

Example

Iterate the values of a tuple:

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

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

Example
Iterate the characters of a string:

```
mystr = "banana"

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

The for loop actually creates an iterator object and executes the next() method for each loop.

ADVERTISEMENT

Create an Iterator
To create an object/class as an iterator you have to implement the methods __iter__() and __next__() to your object.

As you have learned in the Python Classes/Objects chapter, all classes have a function called __init__(), which allows you to do some initializing when the object is being created.

The __iter__() method acts similar, you can do operations (initializing etc.), but must always return the iterator object itself.

The __next__() method also allows you to do operations, and must return the next item in the sequence.

Example
Create an iterator that returns numbers, starting with 1, and each sequence will increase by one (returning 1,2,3,4,5 etc.):

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    x = self.a
    self.a += 1
    return x

myclass = MyNumbers()
myiter = iter(myclass)

print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
```

StopIteration
The example above would continue forever if you had enough next() statements, or if it was used in a for loop.

To prevent the iteration from going on forever, we can use the StopIteration statement.

In the __next__() method, we can add a terminating condition to raise an error if the iteration is done a specified number of times:

Example
Stop after 20 iterations:

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    if self.a <= 20:
      x = self.a
      self.a += 1
      return x
    else:
      raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)

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

Python Polymorphism
The word "polymorphism" means "many forms", and in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes.

Function Polymorphism
An example of a Python function that can be used on different objects is the len() function.

String
For strings len() returns the number of characters:

ExampleGet your own Python Server
```
x = "Hello World!"

print(len(x))
```
Tuple
For tuples len() returns the number of items in the tuple:

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

print(len(mytuple))
```
Dictionary
For dictionaries len() returns the number of key/value pairs in the dictionary:

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

print(len(thisdict))
```


Class Polymorphism
Polymorphism is often used in Class methods, where we can have multiple classes with the same method name.

For example, say we have three classes: Car, Boat, and Plane, and they all have a method called move():

Example
Different classes with the same method:

```
class Car:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Drive!")

class Boat:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Sail!")

class Plane:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Fly!")

car1 = Car("Ford", "Mustang")       #Create a Car class
boat1 = Boat("Ibiza", "Touring 20") #Create a Boat class
plane1 = Plane("Boeing", "747")     #Create a Plane class

for x in (car1, boat1, plane1):
  x.move()
```

Look at the for loop at the end. Because of polymorphism we can execute the same method for all three classes.

Inheritance Class Polymorphism
What about classes with child classes with the same name? Can we use polymorphism there?

Yes. If we use the example above and make a parent class called Vehicle, and make Car, Boat, Plane child classes of Vehicle, the child classes inherits the Vehicle methods, but can override them:

Example
Create a class called Vehicle and make Car, Boat, Plane child classes of Vehicle:

```
class Vehicle:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Move!")

class Car(Vehicle):
  pass

class Boat(Vehicle):
  def move(self):
```

```
    print("Sail!")

class Plane(Vehicle):
  def move(self):
    print("Fly!")

car1 = Car("Ford", "Mustang") #Create a Car object
boat1 = Boat("Ibiza", "Touring 20") #Create a Boat object
plane1 = Plane("Boeing", "747") #Create a Plane object

for x in (car1, boat1, plane1):
  print(x.brand)
  print(x.model)
  x.move()
```

Child classes inherits the properties and methods from the parent class.

In the example above you can see that the Car class is empty, but it inherits brand, model, and move() from Vehicle.

The Boat and Plane classes also inherit brand, model, and move() from Vehicle, but they both override the move() method.

Because of polymorphism we can execute the same method for all classes.

Python Scope
A variable is only available from inside the region it is created. This is called scope.

Local Scope
A variable created inside a function belongs to the local scope of that function, and can only be used inside that function.

ExampleGet your own Python Server
A variable created inside a function is available inside that function:

```
def myfunc():
  x = 300
  print(x)

myfunc()
```

Function Inside Function
As explained in the example above, the variable x is not available outside the function, but it is available for any function inside the function:

Example
The local variable can be accessed from a function within the function:

```
def myfunc():
  x = 300
  def myinnerfunc():
    print(x)
  myinnerfunc()

myfunc()
```


Global Scope
A variable created in the main body of the Python code is a global variable and belongs to the global scope.

Global variables are available from within any scope, global and local.

Example
A variable created outside of a function is global and can be used by anyone:

```
x = 300

def myfunc():
  print(x)

myfunc()

print(x)
```

Naming Variables
If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function):

Example
The function will print the local x, and then the code will print the global x:

```
x = 300

def myfunc():
  x = 200
  print(x)

myfunc()

print(x)
```

Global Keyword
If you need to create a global variable, but are stuck in the local scope, you can use the global keyword.

The global keyword makes the variable global.

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

```
def myfunc():
  global x
  x = 300

myfunc()

print(x)
```

Also, use the global keyword if you want to make a change to 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 = 300

def myfunc():
  global x
  x = 200

myfunc()

print(x)
```

Python Modules
What is a Module?
Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

Create a Module
To create a module just save the code you want in a file with the file extension .py:

ExampleGet your own Python Server
Save this code in a file named mymodule.py

```
def greeting(name):
  print("Hello, " + name)
```
Use a Module
Now we can use the module we just created, by using the import statement:

Example
Import the module named mymodule, and call the greeting function:

```
import mymodule

mymodule.greeting("Jonathan")
```
Note: When using a function from a module, use the syntax: module_name.function_name.

Variables in Module
The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):

Example
Save this code in the file mymodule.py

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```
Example
Import the module named mymodule, and access the person1 dictionary:

```
import mymodule

a = mymodule.person1["age"]
print(a)
```


Naming a Module
You can name the module file whatever you like, but it must have the file extension .py

Re-naming a Module
You can create an alias when you import a module, by using the as keyword:

Example
Create an alias for mymodule called mx:

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```
Built-in Modules
There are several built-in modules in Python, which you can import whenever you

like.

Example
Import and use the platform module:

```
import platform

x = platform.system()
print(x)
```

Using the dir() Function
There is a built-in function to list all the function names (or variable names) in a module. The dir() function:

Example
List all the defined names belonging to the platform module:

```
import platform

x = dir(platform)
print(x)
```

Note: The dir() function can be used on all modules, also the ones you create yourself.

Import From Module
You can choose to import only parts from a module, by using the from keyword.

Example
The module named mymodule has one function and one dictionary:

```
def greeting(name):
  print("Hello, " + name)

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Example
Import only the person1 dictionary from the module:

```
from mymodule import person1

print (person1["age"])
```

Note: When importing using the from keyword, do not use the module name when referring to elements in the module. Example: person1["age"], not mymodule.person1["age"]

Python Datetime
Python Dates
A date in Python is not a data type of its own, but we can import a module named datetime to work with dates as date objects.

ExampleGet your own Python Server
Import the datetime module and display the current date:

```
import datetime

x = datetime.datetime.now()
print(x)
```

Date Output
When we execute the code from the example above the result will be:

2023-11-27 09:33:35.709703
The date contains year, month, day, hour, minute, second, and microsecond.

The datetime module has many methods to return information about the date object.

Here are a few examples, you will learn more about them later in this chapter:

Example
Return the year and name of weekday:

```
import datetime

x = datetime.datetime.now()

print(x.year)
print(x.strftime("%A"))
```

Creating Date Objects
To create a date, we can use the datetime() class (constructor) of the datetime module.

The datetime() class requires three parameters to create a date: year, month, day.

Example
Create a date object:

```
import datetime

x = datetime.datetime(2020, 5, 17)

print(x)
```

The datetime() class also takes parameters for time and timezone (hour, minute, second, microsecond, tzone), but they are optional, and has a default value of 0, (None for timezone).

The strftime() Method
The datetime object has a method for formatting date objects into readable strings.

The method is called strftime(), and takes one parameter, format, to specify the format of the returned string:

Example
Display the name of the month:

```
import datetime

x = datetime.datetime(2018, 6, 1)

print(x.strftime("%B"))
```

A reference of all the legal format codes:

Directive	Description	Example	Try it
%a	Weekday, short version	Wed	
%A	Weekday, full version	Wednesday	
%w	Weekday as a number 0-6, 0 is Sunday	3	
%d	Day of month 01-31	31	
%b	Month name, short version	Dec	
%B	Month name, full version	December	
%m	Month as a number 01-12	12	
%y	Year, short version, without century	18	

%Y   Year, full version   2018
%H   Hour 00-23 17
%I   Hour 00-12 05
%p   AM/PM PM
%M   Minute 00-59   41
%S   Second 00-59   08
%f   Microsecond 000000-999999   548513
%z   UTC offset +0100
%Z   Timezone   CST
%j   Day number of year 001-366   365
%U   Week number of year, Sunday as the first day of week, 00-5352
%W   Week number of year, Monday as the first day of week, 00-5352
%c   Local version of date and time   Mon Dec 31 17:41:00 2018
%C   Century   20
%x   Local version of date   12/31/18
%X   Local version of time   17:41:00
%%   A % character   %
%G   ISO 8601 year   2018
%u   ISO 8601 weekday (1-7) 1
%V   ISO 8601 weeknumber (01-53)   01

Python Math
Python has a set of built-in math functions, including an extensive math module, that allows you to perform mathematical tasks on numbers.

Built-in Math Functions
The min() and max() functions can be used to find the lowest or highest value in an iterable:

ExampleGet your own Python Server

```
x = min(5, 10, 25)
y = max(5, 10, 25)

print(x)
print(y)
```

The abs() function returns the absolute (positive) value of the specified number:

Example

```
x = abs(-7.25)

print(x)
```

The pow(x, y) function returns the value of x to the power of y (xy).

Example
Return the value of 4 to the power of 3 (same as 4 * 4 * 4):

```
x = pow(4, 3)

print(x)
```


The Math Module
Python has also a built-in module called math, which extends the list of mathematical functions.

To use it, you must import the math module:

```
import math
```

When you have imported the math module, you can start using methods and constants of the module.

The math.sqrt() method for example, returns the square root of a number:

Example

```
import math

x = math.sqrt(64)

print(x)
```

The math.ceil() method rounds a number upwards to its nearest integer, and the math.floor() method rounds a number downwards to its nearest integer, and returns the result:

Example

```
import math

x = math.ceil(1.4)
y = math.floor(1.4)

print(x) # returns 2
print(y) # returns 1
```

The math.pi constant, returns the value of PI (3.14...):

Example

```
import math

x = math.pi

print(x)
```

Complete Math Module Reference

In our Math Module Reference you will find a complete reference of all methods and constants that belongs to the Math module.

Python JSON

JSON is a syntax for storing and exchanging data.

JSON is text, written with JavaScript object notation.

JSON in Python

Python has a built-in package called json, which can be used to work with JSON data.

ExampleGet your own Python Server

Import the json module:

```
import json
```

Parse JSON - Convert from JSON to Python

If you have a JSON string, you can parse it by using the json.loads() method.

The result will be a Python dictionary.

Example

Convert from JSON to Python:

```
import json

# some JSON:
x =  '{ "name":"John", "age":30, "city":"New York"}'

# parse x:
y = json.loads(x)

# the result is a Python dictionary:
```

print(y["age"])
Convert from Python to JSON
If you have a Python object, you can convert it into a JSON string by using the json.dumps() method.

Example
Convert from Python to JSON:

```
import json

# a Python object (dict):
x = {
  "name": "John",
  "age": 30,
  "city": "New York"
}

# convert into JSON:
y = json.dumps(x)

# the result is a JSON string:
print(y)
```
ADVERTISEMENT

You can convert Python objects of the following types, into JSON strings:

dict
list
tuple
string
int
float
True
False
None
Example
Convert Python objects into JSON strings, and print the values:

```
import json

print(json.dumps({"name": "John", "age": 30}))
print(json.dumps(["apple", "bananas"]))
print(json.dumps(("apple", "bananas")))
print(json.dumps("hello"))
print(json.dumps(42))
print(json.dumps(31.76))
print(json.dumps(True))
print(json.dumps(False))
print(json.dumps(None))
```
When you convert from Python to JSON, Python objects are converted into the JSON (JavaScript) equivalent:

Python	JSON
dict	Object
list	Array
tuple	Array
str	String
int	Number
float	Number
True	true
False	false
None	null

Example
Convert a Python object containing all the legal data types:

```
import json

x = {
  "name": "John",
  "age": 30,
  "married": True,
  "divorced": False,
  "children": ("Ann","Billy"),
  "pets": None,
  "cars": [
    {"model": "BMW 230", "mpg": 27.5},
    {"model": "Ford Edge", "mpg": 24.1}
  ]
}

print(json.dumps(x))
```
Format the Result
The example above prints a JSON string, but it is not very easy to read, with no indentations and line breaks.

The json.dumps() method has parameters to make it easier to read the result:

Example
Use the indent parameter to define the numbers of indents:

```
json.dumps(x, indent=4)
```
You can also define the separators, default value is (", ", ": "), which means using a comma and a space to separate each object, and a colon and a space to separate keys from values:

Example
Use the separators parameter to change the default separator:

```
json.dumps(x, indent=4, separators=(". ", " = "))
```
Order the Result
The json.dumps() method has parameters to order the keys in the result:

Example
Use the sort_keys parameter to specify if the result should be sorted or not:

```
json.dumps(x, indent=4, sort_keys=True)
```

Python RegEx
A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.

RegEx can be used to check if a string contains the specified search pattern.

RegEx Module
Python has a built-in package called re, which can be used to work with Regular Expressions.

Import the re module:

```
import re
```
RegEx in Python
When you have imported the re module, you can start using regular expressions:

ExampleGet your own Python Server
Search the string to see if it starts with "The" and ends with "Spain":

import re

txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
RegEx Functions
The re module offers a set of functions that allows us to search a string for a match:

Function	Description
findall	Returns a list containing all matches
search	Returns a Match object if there is a match anywhere in the string
split	Returns a list where the string has been split at each match
sub	Replaces one or many matches with a string

ADVERTISEMENT

Metacharacters
Metacharacters are characters with a special meaning:

Character	Description	Example	Try it
[]	A set of characters	"[a-m]"	
\	Signals a special sequence (can also be used to escape special characters)	"\d"	
.	Any character (except newline character)	"he..o"	
^	Starts with	"^hello"	
$	Ends with	"planet$"	
*	Zero or more occurrences	"he.*o"	
+	One or more occurrences	"he.+o"	
?	Zero or one occurrences	"he.?o"	
{}	Exactly the specified number of occurrences	"he.{2}o"	
|	Either or	"falls|stays"	
()	Capture and group		

Special Sequences
A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

Character	Description	Example	Try it
\A	Returns a match if the specified characters are at the beginning of the string	"\AThe"	
\b	Returns a match where the specified characters are at the beginning or at the end of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\bain" r"ain\b"	
\B	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\Bain" r"ain\B"	
\d	Returns a match where the string contains digits (numbers from 0-9)	"\d"	
\D	Returns a match where the string DOES NOT contain digits	"\D"	
\s	Returns a match where the string contains a white space character	"\s"	
\S	Returns a match where the string DOES NOT contain a white space character	"\S"	
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	"\w"	
\W	Returns a match where the string DOES NOT contain any word characters	"\W"	
\Z	Returns a match if the specified characters are at the end of the string		

"Spain\Z"
Sets
A set is a set of characters inside a pair of square brackets [] with a special meaning:

Set	Description	Try it
[arn]	Returns a match where one of the specified characters (a, r, or n) is present	
[a-n]	Returns a match for any lower case character, alphabetically between a and n	
[^arn]	Returns a match for any character EXCEPT a, r, and n	
[0123]	Returns a match where any of the specified digits (0, 1, 2, or 3) are present	
[0-9]	Returns a match for any digit between 0 and 9	
[0-5][0-9]	Returns a match for any two-digit numbers from 00 and 59	
[a-zA-Z]	Returns a match for any character alphabetically between a and z, lower case OR upper case	
[+]	In sets, +, *, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string	

The findall() Function
The findall() function returns a list containing all matches.

Example
Print a list of all matches:

```
import re

txt = "The rain in Spain"
x = re.findall("ai", txt)
print(x)
```

The list contains the matches in the order they are found.

If no matches are found, an empty list is returned:

Example
Return an empty list if no match was found:

```
import re

txt = "The rain in Spain"
x = re.findall("Portugal", txt)
print(x)
```

The search() Function
The search() function searches the string for a match, and returns a Match object if there is a match.

If there is more than one match, only the first occurrence of the match will be returned:

Example
Search for the first white-space character in the string:

```
import re

txt = "The rain in Spain"
x = re.search("\s", txt)

print("The first white-space character is located in position:", x.start())
```

If no matches are found, the value None is returned:

Example
Make a search that returns no match:

```
import re

txt = "The rain in Spain"
x = re.search("Portugal", txt)
print(x)
```

The split() Function
The split() function returns a list where the string has been split at each match:

Example
Split at each white-space character:

```
import re

txt = "The rain in Spain"
x = re.split("\s", txt)
print(x)
```

You can control the number of occurrences by specifying the maxsplit parameter:

Example
Split the string only at the first occurrence:

```
import re

txt = "The rain in Spain"
x = re.split("\s", txt, 1)
print(x)
```

The sub() Function
The sub() function replaces the matches with the text of your choice:

Example
Replace every white-space character with the number 9:

```
import re

txt = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)
```

You can control the number of replacements by specifying the count parameter:

Example
Replace the first 2 occurrences:

```
import re

txt = "The rain in Spain"
x = re.sub("\s", "9", txt, 2)
print(x)
```

Match Object
A Match Object is an object containing information about the search and the result.

Note: If there is no match, the value None will be returned, instead of the Match Object.

Example
Do a search that will return a Match Object:

```
import re
```

```
txt = "The rain in Spain"
x = re.search("ai", txt)
print(x) #this will print an object
```
The Match object has properties and methods used to retrieve information about the search, and the result:

.span() returns a tuple containing the start-, and end positions of the match.
.string returns the string passed into the function
.group() returns the part of the string where there was a match

Example
Print the position (start- and end-position) of the first match occurrence.

The regular expression looks for any words that starts with an upper case "S":

```
import re

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.span())
```
Example
Print the string passed into the function:

```
import re

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.string)
```
Example
Print the part of the string where there was a match.

The regular expression looks for any words that starts with an upper case "S":

```
import re

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.group())
```
Note: If there is no match, the value None will be returned, instead of the Match Object.

Python PIP
What is PIP?
PIP is a package manager for Python packages, or modules if you like.

Note: If you have Python version 3.4 or later, PIP is included by default.

What is a Package?
A package contains all the files you need for a module.

Modules are Python code libraries you can include in your project.

Check if PIP is Installed
Navigate your command line to the location of Python's script directory, and type the following:

ExampleGet your own Python Server
Check PIP version:

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip --
version
```
Install PIP

If you do not have PIP installed, you can download and install it from this page: https://pypi.org/project/pip/

Download a Package
Downloading a package is very easy.

Open the command line interface and tell PIP to download the package you want.

Navigate your command line to the location of Python's script directory, and type the following:

Example
Download a package named "camelcase":

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip install camelcase
```

Now you have downloaded and installed your first package!

Using a Package
Once the package is installed, it is ready to use.

Import the "camelcase" package into your project.

Example
Import and use "camelcase":

```
import camelcase

c = camelcase.CamelCase()

txt = "hello world"

print(c.hump(txt))
```

Find Packages
Find more packages at https://pypi.org/.

Remove a Package
Use the uninstall command to remove a package:

Example
Uninstall the package named "camelcase":

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip uninstall camelcase
```

The PIP Package Manager will ask you to confirm that you want to remove the camelcase package:

```
Uninstalling camelcase-02.1:
  Would remove:
    c:\users\Your Name\appdata\local\programs\python\python36-32\lib\site-packages\camelcase-0.2-py3.6.egg-info
    c:\users\Your Name\appdata\local\programs\python\python36-32\lib\site-packages\camelcase\*
Proceed (y/n)?
```

Press y and the package will be removed.

List Packages
Use the list command to list all the packages installed on your system:

Example
List installed packages:

C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip list
Result:

```
Package         Version
-----------------------
camelcase       0.2
mysql-connector 2.1.6
pip             18.1
pymongo         3.6.1
setuptools      39.0.1
```

Python Try Except
The try block lets you test a block of code for errors.

The except block lets you handle the error.

The else block lets you execute code when there is no error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

Exception Handling
When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the try statement:

ExampleGet your own Python Server
The try block will generate an exception, because x is not defined:

```
try:
  print(x)
except:
  print("An exception occurred")
```

Since the try block raises an error, the except block will be executed.

Without the try block, the program will crash and raise an error:

Example
This statement will raise an error, because x is not defined:

```
print(x)
```

Many Exceptions
You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:

Example
Print one message if the try block raises a NameError and another for other errors:

```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```


Else
You can use the else keyword to define a block of code to be executed if no errors were raised:

Example
In this example, the try block does not generate any error:

```
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```

Finally
The finally block, if specified, will be executed regardless if the try block raises an error or not.

Example

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

This can be useful to close objects and clean up resources:

Example
Try to open and write to a file that is not writable:

```
try:
  f = open("demofile.txt")
  try:
    f.write("Lorum Ipsum")
  except:
    print("Something went wrong when writing to the file")
  finally:
    f.close()
except:
  print("Something went wrong when opening the file")
```

The program can continue, without leaving the file object open.

Raise an exception
As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the raise keyword.

Example
Raise an error and stop the program if x is lower than 0:

```
x = -1

if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

The raise keyword is used to raise an exception.

You can define what kind of error to raise, and the text to print to the user.

Example
Raise a TypeError if x is not an integer:

```
x = "hello"

if not type(x) is int:
  raise TypeError("Only integers are allowed")
```

Python User Input
User Input
Python allows for user input.

That means we are able to ask the user for input.

The method is a bit different in Python 3.6 than Python 2.7.

Python 3.6 uses the input() method.

Python 2.7 uses the raw_input() method.

The following example asks for the username, and when you entered the username, it gets printed on the screen:

Python 3.6Get your own Python Server
```
username = input("Enter username:")
print("Username is: " + username)
```
Python 2.7
```
username = raw_input("Enter username:")
print("Username is: " + username)
```
Python stops executing when it comes to the input() function, and continues when the user has given some input.

Python String Formatting
To make sure a string will display as expected, we can format the result with the format() method.

String format()
The format() method allows you to format selected parts of a string.

Sometimes there are parts of a text that you do not control, maybe they come from a database, or user input?

To control such values, add placeholders (curly brackets {}) in the text, and run the values through the format() method:

ExampleGet your own Python Server
Add a placeholder where you want to display the price:

```
price = 49
txt = "The price is {} dollars"
print(txt.format(price))
```
You can add parameters inside the curly brackets to specify how to convert the value:

Example
Format the price to be displayed as a number with two decimals:

```
txt = "The price is {:.2f} dollars"
```
Check out all formatting types in our String format() Reference.

Multiple Values
If you want to use more values, just add more values to the format() method:

```
print(txt.format(price, itemno, count))
```
And add more placeholders:

Example
```
quantity = 3
itemno = 567
price = 49
```

myorder = "I want {} pieces of item number {} for {:.2f} dollars."
print(myorder.format(quantity, itemno, price))
ADVERTISEMENT

Index Numbers
You can use index numbers (a number inside the curly brackets {0}) to be sure the values are placed in the correct placeholders:

Example
quantity = 3
itemno = 567
price = 49
myorder = "I want {0} pieces of item number {1} for {2:.2f} dollars."
print(myorder.format(quantity, itemno, price))
Also, if you want to refer to the same value more than once, use the index number:

Example
age = 36
name = "John"
txt = "His name is {1}. {1} is {0} years old."
print(txt.format(age, name))
Named Indexes
You can also use named indexes by entering a name inside the curly brackets {carname}, but then you must use names when you pass the parameter values txt.format(carname = "Ford"):

Example
myorder = "I have a {carname}, it is a {model}."
print(myorder.format(carname = "Ford", model = "Mustang"))

Python File Open
File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

File Handling
The key function for working with files in Python is the open() function.

The open() function takes two parameters; filename, and mode.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

Syntax
To open a file for reading it is enough to specify the name of the file:

f = open("demofile.txt")
The code above is the same as:

f = open("demofile.txt", "rt")
Because "r" for read, and "t" for text are the default values, you do not need to specify them.

Note: Make sure the file exists, or else you will get an error.

Python File Open
Open a File on the Server
Assume we have the following file, located in the same folder as Python:

demofile.txt

Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
To open the file, use the built-in open() function.

The open() function returns a file object, which has a read() method for reading the content of the file:

ExampleGet your own Python Server
f = open("demofile.txt", "r")
print(f.read())
If the file is located in a different location, you will have to specify the file path, like this:

Example
Open a file on a different location:

f = open("D:\\myfiles\welcome.txt", "r")
print(f.read())
Read Only Parts of the File
By default the read() method returns the whole text, but you can also specify how many characters you want to return:

Example
Return the 5 first characters of the file:

f = open("demofile.txt", "r")
print(f.read(5))
ADVERTISEMENT

Read Lines
You can return one line by using the readline() method:

Example
Read one line of the file:

f = open("demofile.txt", "r")
print(f.readline())
By calling readline() two times, you can read the two first lines:

Example
Read two lines of the file:

f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
By looping through the lines of the file, you can read the whole file, line by

line:

Example
Loop through the file line by line:

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```
Close Files
It is a good practice to always close the file when you are done with it.

Example
Close the file when you are finish with it:

```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```
Note: You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.

Python File Write
Write to an Existing File
To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

ExampleGet your own Python Server
Open the file "demofile2.txt" and append content to the file:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```
Example
Open the file "demofile3.txt" and overwrite the content:

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the overwriting:
f = open("demofile3.txt", "r")
print(f.read())
```
Note: the "w" method will overwrite the entire file.

Create a New File
To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create - will create a file, returns an error if the file exist

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

Example
Create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

Result: a new empty file is created!

Example
Create a new file if it does not exist:

```
f = open("myfile.txt", "w")
```

Python Delete File
Delete a File
To delete a file, you must import the OS module, and run its os.remove() function:

ExampleGet your own Python Server
Remove the file "demofile.txt":

```
import os
os.remove("demofile.txt")
```

Check if File exist:
To avoid getting an error, you might want to check if the file exists before you try to delete it:

Example
Check if file exists, then delete it:

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

Delete Folder
To delete an entire folder, use the os.rmdir() method:

Example
Remove the folder "myfolder":

```
import os
os.rmdir("myfolder")
```

Note: You can only remove empty folders.

NumPy Tutorial
[+:
NumPy is a Python library.

NumPy is used for working with arrays.

NumPy is short for "Numerical Python".

Learning by Reading
We have created 43 tutorial pages for you to learn more about NumPy.

Starting with a basic introduction and ends up with creating and plotting random data sets, and working with NumPy functions:

Basic
Introduction
Getting Started
Creating Arrays
Array Indexing

Array Slicing
Data Types
Copy vs View
Array Shape
Array Reshape
Array Iterating
Array Join
Array Split
Array Search
Array Sort
Array Filter

Random
Random Intro
Data Distribution
Random Permutation
Seaborn Module
Normal Dist.
Binomial Dist.
Poison Dist.
Uniform Dist.
Logistic Dist.
Multinomial Dist.
Exponential Dis.
Chi Square Dist.
Rayleigh Dist.
Pareto Dist.
Zipf Dist.

ufunc
ufunc Intro
Create Function
Simple Arithmetic
Rounding Decimals
Logs
Summations
Products
Differences
Finding LCM
Finding GCD
Trigonometric
Hyperbolic
Set Operations

Learning by Quiz Test
Test your NumPy skills with a quiz test.

Learning by Exercises
NumPy Exercises
Exercise:
Insert the correct method for creating a NumPy array.

```
arr = np.([1, 2, 3, 4, 5])
```

Start the Exercise

Learning by Examples
In our "Try it Yourself" editor, you can use the NumPy module, and modify the code to see the result.

ExampleGet your own Python Server
Create a NumPy array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```
Click on the "Try it Yourself" button to see how it works.

NumPy Introduction
What is NumPy?
NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

NumPy stands for Numerical Python.

Why Use NumPy?
In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

Data Science: is a branch of computer science where we study how to store, use and analyze data for deriving information from it.

Why is NumPy Faster Than Lists?
NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

This behavior is called locality of reference in computer science.

This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

Which Language is NumPy written in?
NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

Where is the NumPy Codebase?
The source code for NumPy is located at this github repository
https://github.com/numpy/numpy

github: enables many people to work on the same codebase.

NumPy Getting Started
Installation of NumPy
If you have Python and PIP already installed on a system, then installation of

NumPy is very easy.

Install it using this command:

C:\Users\Your Name>pip install numpy
If this command fails, then use a python distribution that already has NumPy installed like, Anaconda, Spyder etc.

Import NumPy
Once NumPy is installed, import it in your applications by adding the import keyword:

import numpy
Now NumPy is imported and ready to use.

ExampleGet your own Python Server
```
import numpy

arr = numpy.array([1, 2, 3, 4, 5])

print(arr)
```
NumPy as np
NumPy is usually imported under the np alias.

alias: In Python alias are an alternate name for referring to the same thing.

Create an alias with the as keyword while importing:

import numpy as np
Now the NumPy package can be referred to as np instead of numpy.

Example
```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)
```
Checking NumPy Version
The version string is stored under __version__ attribute.

Example
```
import numpy as np

print(np.__version__)
```

NumPy Creating Arrays
Create a NumPy ndarray Object
NumPy is used to work with arrays. The array object in NumPy is called ndarray.

We can create a NumPy ndarray object by using the array() function.

ExampleGet your own Python Server
```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```
type(): This built-in Python function tells us the type of the object passed to it. Like in above code it shows that arr is numpy.ndarray type.

To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray:

Example
Use a tuple to create a NumPy array:

```
import numpy as np

arr = np.array((1, 2, 3, 4, 5))

print(arr)
```

Dimensions in Arrays
A dimension in arrays is one level of array depth (nested arrays).

nested array: are arrays that have arrays as their elements.

0-D Arrays
0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

Example
Create a 0-D array with value 42

```
import numpy as np

arr = np.array(42)

print(arr)
```

1-D Arrays
An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

Example
Create a 1-D array containing the values 1,2,3,4,5:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)
```

2-D Arrays
An array that has 1-D arrays as its elements is called a 2-D array.

These are often used to represent matrix or 2nd order tensors.

NumPy has a whole sub module dedicated towards matrix operations called numpy.mat

Example
Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr)
```

3-D arrays
An array that has 2-D arrays (matrices) as its elements is called 3-D array.

These are often used to represent a 3rd order tensor.

Example
Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)
```

Check Number of Dimensions?
NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

Example
Check how many dimensions the arrays have:

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

Higher Dimensional Arrays
An array can have any number of dimensions.

When the array is created, you can define the number of dimensions by using the ndmin argument.

Example
Create an array with 5 dimensions and verify that it has 5 dimensions:

```
import numpy as np

arr = np.array([1, 2, 3, 4], ndmin=5)

print(arr)
print('number of dimensions :', arr.ndim)
```

In this array the innermost dimension (5th dim) has 4 elements, the 4th dim has 1 element that is the vector, the 3rd dim has 1 element that is the matrix with the vector, the 2nd dim has 1 element that is 3D array and 1st dim has 1 element that is a 4D array.

NumPy Array Indexing
Access Array Elements
Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

ExampleGet your own Python Server
Get the first element from the following array:

```
import numpy as np
```

arr = np.array([1, 2, 3, 4])

print(arr[0])
Example
Get the second element from the following array.

import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[1])
Example
Get third and fourth elements from the following array and add them.

import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[2] + arr[3])

Access 2-D Arrays
To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Think of 2-D arrays like a table with rows and columns, where the dimension represents the row and the index represents the column.

Example
Access the element on the first row, second column:

import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st row: ', arr[0, 1])
Example
Access the element on the 2nd row, 5th column:

import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('5th element on 2nd row: ', arr[1, 4])
Access 3-D Arrays
To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

Example
Access the third element of the second array of the first array:

import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print(arr[0, 1, 2])
Example Explained
arr[0, 1, 2] prints the value 6.

And this is why:

The first number represents the first dimension, which contains two arrays:
[[1, 2, 3], [4, 5, 6]]
and:

[[7, 8, 9], [10, 11, 12]]
Since we selected 0, we are left with the first array:
[[1, 2, 3], [4, 5, 6]]

The second number represents the second dimension, which also contains two arrays:
[1, 2, 3]
and:
[4, 5, 6]
Since we selected 1, we are left with the second array:
[4, 5, 6]

The third number represents the third dimension, which contains three values:
4
5
6
Since we selected 2, we end up with the third value:
6

Negative Indexing
Use negative indexing to access an array from the end.

Example
Print the last element from the 2nd dim:

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('Last element from 2nd dim: ', arr[1, -1])
```

NumPy Array Slicing
Slicing arrays
Slicing in python means taking elements from one given index to another given index.

We pass slice instead of index like this: [start:end].

We can also define the step, like this: [start:end:step].

If we don't pass start its considered 0

If we don't pass end its considered length of array in that dimension

If we don't pass step its considered 1

ExampleGet your own Python Server
Slice elements from index 1 to index 5 from the following array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5])
```
Note: The result includes the start index, but excludes the end index.

Example
Slice elements from index 4 to the end of the array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])
```

print(arr[4:])
Example
Slice elements from the beginning to index 4 (not included):

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[:4])
```
ADVERTISEMENT

Negative Slicing
Use the minus operator to refer to an index from the end:

Example
Slice from the index 3 from the end to index 1 from the end:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[-3:-1])
```
STEP
Use the step value to determine the step of the slicing:

Example
Return every other element from index 1 to index 5:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5:2])
```
Example
Return every other element from the entire array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[::2])
```
Slicing 2-D Arrays
Example
From the second element, slice elements from index 1 to index 4 (not included):

```
import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4])
```
Note: Remember that second element has index 1.

Example
From both elements, return index 2:

```
import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[0:2, 2])
```
Example
From both elements, slice index 1 to index 4 (not included), this will return a 2-D array:

```
import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[0:2, 1:4])
```

NumPy Data Types
Data Types in Python
By default Python have these data types:

strings - used to represent text data, the text is given under quote marks. e.g. "ABCD"
integer - used to represent integer numbers. e.g. -1, -2, -3
float - used to represent real numbers. e.g. 1.2, 42.42
boolean - used to represent True or False.
complex - used to represent complex numbers. e.g. 1.0 + 2.0j, 1.5 + 2.5j
Data Types in NumPy
NumPy has some extra data types, and refer to data types with one character, like i for integers, u for unsigned integers etc.

Below is a list of all data types in NumPy and the characters used to represent them.

i - integer
b - boolean
u - unsigned integer
f - float
c - complex float
m - timedelta
M - datetime
O - object
S - string
U - unicode string
V - fixed chunk of memory for other type (void)
Checking the Data Type of an Array
The NumPy array object has a property called dtype that returns the data type of the array:

ExampleGet your own Python Server
Get the data type of an array object:

```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr.dtype)
```

Example
Get the data type of an array containing strings:

```
import numpy as np

arr = np.array(['apple', 'banana', 'cherry'])

print(arr.dtype)
```


Creating Arrays With a Defined Data Type
We use the array() function to create arrays, this function can take an optional argument: dtype that allows us to define the expected data type of the array elements:

Example
Create an array with data type string:

```
import numpy as np

arr = np.array([1, 2, 3, 4], dtype='S')

print(arr)
print(arr.dtype)
```
For i, u, f, S and U we can define size as well.

Example
Create an array with data type 4 bytes integer:

```
import numpy as np

arr = np.array([1, 2, 3, 4], dtype='i4')

print(arr)
print(arr.dtype)
```
What if a Value Can Not Be Converted?
If a type is given in which elements can't be casted then NumPy will raise a ValueError.

ValueError: In Python ValueError is raised when the type of passed argument to a function is unexpected/incorrect.

Example
A non integer string like 'a' can not be converted to integer (will raise an error):

```
import numpy as np

arr = np.array(['a', '2', '3'], dtype='i')
```
Converting Data Type on Existing Arrays
The best way to change the data type of an existing array, is to make a copy of the array with the astype() method.

The astype() function creates a copy of the array, and allows you to specify the data type as a parameter.

The data type can be specified using a string, like 'f' for float, 'i' for integer etc. or you can use the data type directly like float for float and int for integer.

Example
Change data type from float to integer by using 'i' as parameter value:

```
import numpy as np

arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype('i')

print(newarr)
print(newarr.dtype)
```
Example
Change data type from float to integer by using int as parameter value:

```
import numpy as np

arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype(int)
```

print(newarr)
print(newarr.dtype)
Example
Change data type from integer to boolean:

```
import numpy as np

arr = np.array([1, 0, 3])

newarr = arr.astype(bool)

print(newarr)
print(newarr.dtype)
```

NumPy Array Copy vs View
The Difference Between Copy and View
The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array.

The copy owns the data and any changes made to the copy will not affect original array, and any changes made to the original array will not affect the copy.

The view does not own the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the view.

COPY:
ExampleGet your own Python Server
Make a copy, change the original array, and display both arrays:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)
```

The copy SHOULD NOT be affected by the changes made to the original array.

VIEW:
Example
Make a view, change the original array, and display both arrays:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42

print(arr)
print(x)
```

The view SHOULD be affected by the changes made to the original array.

Make Changes in the VIEW:
Example
Make a view, change the view, and display both arrays:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
```

```
x[0] = 31

print(arr)
print(x)
```
The original array SHOULD be affected by the changes made to the view.

Check if Array Owns its Data
As mentioned above, copies owns the data, and views does not own the data, but how can we check this?

Every NumPy array has the attribute base that returns None if the array owns the data.

Otherwise, the base attribute refers to the original object.

Example
Print the value of the base attribute to check if an array owns it's data or not:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

x = arr.copy()
y = arr.view()

print(x.base)
print(y.base)
```
The copy returns None.
The view returns the original array.

NumPy Array Shape
Shape of an Array
The shape of an array is the number of elements in each dimension.

Get the Shape of an Array
NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

ExampleGet your own Python Server
Print the shape of a 2-D array:

```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(arr.shape)
```
The example above returns (2, 4), which means that the array has 2 dimensions, where the first dimension has 2 elements and the second has 4.

Example
Create an array with 5 dimensions using ndmin using a vector with values 1,2,3,4 and verify that last dimension has value 4:

```
import numpy as np

arr = np.array([1, 2, 3, 4], ndmin=5)

print(arr)
print('shape of array :', arr.shape)
```

What does the shape tuple represent?
Integers at every index tells about the number of elements the corresponding dimension has.

In the example above at index-4 we have value 4, so we can say that 5th (4 + 1 th) dimension has 4 elements.

NumPy Array Reshaping
Reshaping arrays
Reshaping means changing the shape of an array.

The shape of an array is the number of elements in each dimension.

By reshaping we can add or remove dimensions or change number of elements in each dimension.

Reshape From 1-D to 2-D
ExampleGet your own Python Server
Convert the following 1-D array with 12 elements into a 2-D array.

The outermost dimension will have 4 arrays, each with 3 elements:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)

print(newarr)
```

Reshape From 1-D to 3-D
Example
Convert the following 1-D array with 12 elements into a 3-D array.

The outermost dimension will have 2 arrays that contains 3 arrays, each with 2 elements:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(2, 3, 2)

print(newarr)
```


Can We Reshape Into any Shape?
Yes, as long as the elements required for reshaping are equal in both shapes.

We can reshape an 8 elements 1D array into 4 elements in 2 rows 2D array but we cannot reshape it into a 3 elements 3 rows 2D array as that would require 3x3 = 9 elements.

Example
Try converting 1D array with 8 elements to a 2D array with 3 elements in each dimension (will raise an error):

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

newarr = arr.reshape(3, 3)
```

```
print(newarr)
```

Returns Copy or View?

Example

Check if the returned array is a copy or a view:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

print(arr.reshape(2, 4).base)
```

The example above returns the original array, so it is a view.

Unknown Dimension

You are allowed to have one "unknown" dimension.

Meaning that you do not have to specify an exact number for one of the dimensions in the reshape method.

Pass -1 as the value, and NumPy will calculate this number for you.

Example

Convert 1D array with 8 elements to 3D array with 2x2 elements:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

newarr = arr.reshape(2, 2, -1)

print(newarr)
```

Note: We can not pass -1 to more than one dimension.

Flattening the arrays

Flattening array means converting a multidimensional array into a 1D array.

We can use reshape(-1) to do this.

Example

Convert the array into a 1D array:

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

newarr = arr.reshape(-1)

print(newarr)
```

Note: There are a lot of functions for changing the shapes of arrays in numpy flatten, ravel and also for rearranging the elements rot90, flip, fliplr, flipud etc. These fall under Intermediate to Advanced section of numpy.

NumPy Array Iterating

Iterating Arrays

Iterating means going through elements one by one.

As we deal with multi-dimensional arrays in numpy, we can do this using basic for loop of python.

If we iterate on a 1-D array it will go through each element one by one.

ExampleGet your own Python Server

Iterate on the elements of the following 1-D array:

```
import numpy as np

arr = np.array([1, 2, 3])

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

Iterating 2-D Arrays

In a 2-D array it will go through all the rows.

Example

Iterate on the elements of the following 2-D array:

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

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

If we iterate on a n-D array it will go through n-1th dimension one by one.

To return the actual values, the scalars, we have to iterate the arrays in each dimension.

Example

Iterate on each scalar element of the 2-D array:

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:
  for y in x:
    print(y)
```


Iterating 3-D Arrays

In a 3-D array it will go through all the 2-D arrays.

Example

Iterate on the elements of the following 3-D array:

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

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

To return the actual values, the scalars, we have to iterate the arrays in each dimension.

Example

Iterate down to the scalars:

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

for x in arr:
  for y in x:
    for z in y:
      print(z)
```

Iterating Arrays Using nditer()

The function nditer() is a helping function that can be used from very basic to very advanced iterations. It solves some basic issues which we face in iteration, lets go through it with examples.

Iterating on Each Scalar Element
In basic for loops, iterating through each scalar of an array we need to use n for loops which can be difficult to write for arrays with very high dimensionality.

Example
Iterate through the following 3-D array:

```
import numpy as np

arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

for x in np.nditer(arr):
  print(x)
```

Iterating Array With Different Data Types
We can use op_dtypes argument and pass it the expected datatype to change the datatype of elements while iterating.

NumPy does not change the data type of the element in-place (where the element is in array) so it needs some other space to perform this action, that extra space is called buffer, and in order to enable it in nditer() we pass flags=['buffered'].

Example
Iterate through the array as a string:

```
import numpy as np

arr = np.array([1, 2, 3])

for x in np.nditer(arr, flags=['buffered'], op_dtypes=['S']):
  print(x)
```

Iterating With Different Step Size
We can use filtering and followed by iteration.

Example
Iterate through every scalar element of the 2D array skipping 1 element:

```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

for x in np.nditer(arr[:, ::2]):
  print(x)
```

Enumerated Iteration Using ndenumerate()
Enumeration means mentioning sequence number of somethings one by one.

Sometimes we require corresponding index of the element while iterating, the ndenumerate() method can be used for those usecases.

Example
Enumerate on following 1D arrays elements:

```
import numpy as np

arr = np.array([1, 2, 3])

for idx, x in np.ndenumerate(arr):
  print(idx, x)
```

Example
Enumerate on following 2D array's elements:

```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

for idx, x in np.ndenumerate(arr):
  print(idx, x)
```

NumPy Joining Array
Joining NumPy Arrays
Joining means putting contents of two or more arrays in a single array.

In SQL we join tables based on a key, whereas in NumPy we join arrays by axes.

We pass a sequence of arrays that we want to join to the concatenate() function, along with the axis. If axis is not explicitly passed, it is taken as 0.

ExampleGet your own Python Server
Join two arrays

```
import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.concatenate((arr1, arr2))

print(arr)
```

Example
Join two 2-D arrays along rows (axis=1):

```
import numpy as np

arr1 = np.array([[1, 2], [3, 4]])

arr2 = np.array([[5, 6], [7, 8]])

arr = np.concatenate((arr1, arr2), axis=1)

print(arr)
```

Joining Arrays Using Stack Functions
Stacking is same as concatenation, the only difference is that stacking is done along a new axis.

We can concatenate two 1-D arrays along the second axis which would result in putting them one over the other, ie. stacking.

We pass a sequence of arrays that we want to join to the stack() method along with the axis. If axis is not explicitly passed it is taken as 0.

Example

```
import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.stack((arr1, arr2), axis=1)

print(arr)
```

Stacking Along Rows
NumPy provides a helper function: hstack() to stack along rows.

Example
```
import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.hstack((arr1, arr2))

print(arr)
```
Stacking Along Columns
NumPy provides a helper function: vstack() to stack along columns.

Example
```
import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.vstack((arr1, arr2))

print(arr)
```
Stacking Along Height (depth)
NumPy provides a helper function: dstack() to stack along height, which is the same as depth.

Example
```
import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.dstack((arr1, arr2))

print(arr)
```

NumPy Splitting Array
Splitting NumPy Arrays
Splitting is reverse operation of Joining.

Joining merges multiple arrays into one and Splitting breaks one array into multiple.

We use array_split() for splitting arrays, we pass it the array we want to split and the number of splits.

ExampleGet your own Python Server
Split the array in 3 parts:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array_split(arr, 3)

print(newarr)
```

Note: The return value is a list containing three arrays.

If the array has less elements than required, it will adjust from the end accordingly.

Example
Split the array in 4 parts:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array_split(arr, 4)

print(newarr)
```
Note: We also have the method split() available but it will not adjust the elements when elements are less in source array for splitting like in example above, array_split() worked properly but split() would fail.

Split Into Arrays
The return value of the array_split() method is an array containing each of the split as an array.

If you split an array into 3 arrays, you can access them from the result just like any array element:

Example
Access the splitted arrays:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array_split(arr, 3)

print(newarr[0])
print(newarr[1])
print(newarr[2])
```
Splitting 2-D Arrays
Use the same syntax when splitting 2-D arrays.

Use the array_split() method, pass in the array you want to split and the number of splits you want to do.

Example
Split the 2-D array into three 2-D arrays.

```
import numpy as np

arr = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])

newarr = np.array_split(arr, 3)

print(newarr)
```
The example above returns three 2-D arrays.

Let's look at another example, this time each element in the 2-D arrays contains 3 elements.

Example
Split the 2-D array into three 2-D arrays.

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15],
[16, 17, 18]])

newarr = np.array_split(arr, 3)

print(newarr)
```

The example above returns three 2-D arrays.

In addition, you can specify which axis you want to do the split around.

The example below also returns three 2-D arrays, but they are split along the row (axis=1).

Example
Split the 2-D array into three 2-D arrays along rows.

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15],
[16, 17, 18]])

newarr = np.array_split(arr, 3, axis=1)

print(newarr)
```

An alternate solution is using hsplit() opposite of hstack()

Example
Use the hsplit() method to split the 2-D array into three 2-D arrays along rows.

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15],
[16, 17, 18]])

newarr = np.hsplit(arr, 3)

print(newarr)
```

Note: Similar alternates to vstack() and dstack() are available as vsplit() and dsplit().

NumPy Searching Arrays
Searching Arrays
You can search an array for a certain value, and return the indexes that get a match.

To search an array, use the where() method.

ExampleGet your own Python Server
Find the indexes where the value is 4:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 4, 4])

x = np.where(arr == 4)

print(x)
```

The example above will return a tuple: (array([3, 5, 6],)

Which means that the value 4 is present at index 3, 5, and 6.

Example
Find the indexes where the values are even:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

x = np.where(arr%2 == 0)

print(x)
```

Example
Find the indexes where the values are odd:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

x = np.where(arr%2 == 1)

print(x)
```


Search Sorted
There is a method called searchsorted() which performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order.

The searchsorted() method is assumed to be used on sorted arrays.

Example
Find the indexes where the value 7 should be inserted:

```
import numpy as np

arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 7)

print(x)
```

Example explained: The number 7 should be inserted on index 1 to remain the sort order.

The method starts the search from the left and returns the first index where the number 7 is no longer larger than the next value.

Search From the Right Side
By default the left most index is returned, but we can give side='right' to return the right most index instead.

Example
Find the indexes where the value 7 should be inserted, starting from the right:

```
import numpy as np

arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 7, side='right')

print(x)
```

Example explained: The number 7 should be inserted on index 2 to remain the sort order.

The method starts the search from the right and returns the first index where the number 7 is no longer less than the next value.

Multiple Values
To search for more than one value, use an array with the specified values.

Example
Find the indexes where the values 2, 4, and 6 should be inserted:

```
import numpy as np

arr = np.array([1, 3, 5, 7])

x = np.searchsorted(arr, [2, 4, 6])

print(x)
```

The return value is an array: [1 2 3] containing the three indexes where 2, 4, 6 would be inserted in the original array to maintain the order.

NumPy Sorting Arrays
Sorting Arrays
Sorting means putting elements in an ordered sequence.

Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

The NumPy ndarray object has a function called sort(), that will sort a specified array.

ExampleGet your own Python Server
Sort the array:

```
import numpy as np

arr = np.array([3, 2, 0, 1])

print(np.sort(arr))
```

Note: This method returns a copy of the array, leaving the original array unchanged.

You can also sort arrays of strings, or any other data type:

Example
Sort the array alphabetically:

```
import numpy as np

arr = np.array(['banana', 'cherry', 'apple'])

print(np.sort(arr))
```

Example
Sort a boolean array:

```
import numpy as np

arr = np.array([True, False, True])

print(np.sort(arr))
```

Sorting a 2-D Array
If you use the sort() method on a 2-D array, both arrays will be sorted:

Example
Sort a 2-D array:

```
import numpy as np

arr = np.array([[3, 2, 4], [5, 0, 1]])

print(np.sort(arr))
```

NumPy Filter Array
Filtering Arrays
Getting some elements out of an existing array and creating a new array out of them is called filtering.

In NumPy, you filter an array using a boolean index list.

A boolean index list is a list of booleans corresponding to indexes in the array.

If the value at an index is True that element is contained in the filtered array, if the value at that index is False that element is excluded from the filtered array.

ExampleGet your own Python Server
Create an array from the elements on index 0 and 2:

```
import numpy as np

arr = np.array([41, 42, 43, 44])

x = [True, False, True, False]

newarr = arr[x]

print(newarr)
```
The example above will return [41, 43], why?

Because the new array contains only the values where the filter array had the value True, in this case, index 0 and 2.

Creating the Filter Array
In the example above we hard-coded the True and False values, but the common use is to create a filter array based on conditions.

Example
Create a filter array that will return only values higher than 42:

```
import numpy as np

arr = np.array([41, 42, 43, 44])

# Create an empty list
filter_arr = []

# go through each element in arr
for element in arr:
  # if the element is higher than 42, set the value to True, otherwise False:
  if element > 42:
    filter_arr.append(True)
  else:
    filter_arr.append(False)

newarr = arr[filter_arr]
```

```
print(filter_arr)
print(newarr)
```

ADVERTISEMENT

Example
Create a filter array that will return only even elements from the original array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

# Create an empty list
filter_arr = []

# go through each element in arr
for element in arr:
  # if the element is completely divisble by 2, set the value to True, otherwise False
  if element % 2 == 0:
    filter_arr.append(True)
  else:
    filter_arr.append(False)

newarr = arr[filter_arr]

print(filter_arr)
print(newarr)
```

Creating Filter Directly From Array
The above example is quite a common task in NumPy and NumPy provides a nice way to tackle it.

We can directly substitute the array instead of the iterable variable in our condition and it will work just as we expect it to.

Example
Create a filter array that will return only values higher than 42:

```
import numpy as np

arr = np.array([41, 42, 43, 44])

filter_arr = arr > 42

newarr = arr[filter_arr]

print(filter_arr)
print(newarr)
```

Example
Create a filter array that will return only even elements from the original array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

filter_arr = arr % 2 == 0

newarr = arr[filter_arr]

print(filter_arr)
print(newarr)
```

Random Numbers in NumPy
What is a Random Number?
Random number does NOT mean a different number every time. Random means something that can not be predicted logically.

Pseudo Random and True Random.
Computers work on programs, and programs are definitive set of instructions. So it means there must be some algorithm to generate a random number as well.

If there is a program to generate random number it can be predicted, thus it is not truly random.

Random numbers generated through a generation algorithm are called pseudo random.

Can we make truly random numbers?

Yes. In order to generate a truly random number on our computers we need to get the random data from some outside source. This outside source is generally our keystrokes, mouse movements, data on network etc.

We do not need truly random numbers, unless it is related to security (e.g. encryption keys) or the basis of application is the randomness (e.g. Digital roulette wheels).

In this tutorial we will be using pseudo random numbers.

Generate Random Number
NumPy offers the random module to work with random numbers.

ExampleGet your own Python Server
Generate a random integer from 0 to 100:

```
from numpy import random

x = random.randint(100)

print(x)
```
Generate Random Float
The random module's rand() method returns a random float between 0 and 1.

Example
Generate a random float from 0 to 1:

```
from numpy import random

x = random.rand()

print(x)
```


Generate Random Array
In NumPy we work with arrays, and you can use the two methods from the above examples to make random arrays.

Integers
The randint() method takes a size parameter where you can specify the shape of an array.

Example
Generate a 1-D array containing 5 random integers from 0 to 100:

```
from numpy import random
```

```
x=random.randint(100, size=(5))

print(x)
```
Example
Generate a 2-D array with 3 rows, each row containing 5 random integers from 0 to 100:

```
from numpy import random

x = random.randint(100, size=(3, 5))

print(x)
```
Floats
The rand() method also allows you to specify the shape of the array.

Example
Generate a 1-D array containing 5 random floats:

```
from numpy import random

x = random.rand(5)

print(x)
```
Example
Generate a 2-D array with 3 rows, each row containing 5 random numbers:

```
from numpy import random

x = random.rand(3, 5)

print(x)
```
Generate Random Number From Array
The choice() method allows you to generate a random value based on an array of values.

The choice() method takes an array as a parameter and randomly returns one of the values.

Example
Return one of the values in an array:

```
from numpy import random

x = random.choice([3, 5, 7, 9])

print(x)
```
The choice() method also allows you to return an array of values.

Add a size parameter to specify the shape of the array.

Example
Generate a 2-D array that consists of the values in the array parameter (3, 5, 7, and 9):

```
from numpy import random

x = random.choice([3, 5, 7, 9], size=(3, 5))

print(x)
```

Random Data Distribution

What is Data Distribution?
Data Distribution is a list of all possible values, and how often each value occurs.

Such lists are important when working with statistics and data science.

The random module offer methods that returns randomly generated data distributions.

Random Distribution
A random distribution is a set of random numbers that follow a certain probability density function.

Probability Density Function: A function that describes a continuous probability. i.e. probability of all values in an array.

We can generate random numbers based on defined probabilities using the choice() method of the random module.

The choice() method allows us to specify the probability for each value.

The probability is set by a number between 0 and 1, where 0 means that the value will never occur and 1 means that the value will always occur.

ExampleGet your own Python Server
Generate a 1-D array containing 100 values, where each value has to be 3, 5, 7 or 9.

The probability for the value to be 3 is set to be 0.1

The probability for the value to be 5 is set to be 0.3

The probability for the value to be 7 is set to be 0.6

The probability for the value to be 9 is set to be 0

```
from numpy import random

x = random.choice([3, 5, 7, 9], p=[0.1, 0.3, 0.6, 0.0], size=(100))

print(x)
```

The sum of all probability numbers should be 1.

Even if you run the example above 100 times, the value 9 will never occur.

You can return arrays of any shape and size by specifying the shape in the size parameter.

Example
Same example as above, but return a 2-D array with 3 rows, each containing 5 values.

```
from numpy import random

x = random.choice([3, 5, 7, 9], p=[0.1, 0.3, 0.6, 0.0], size=(3, 5))

print(x)
```

Random Permutations
Random Permutations of Elements
A permutation refers to an arrangement of elements. e.g. [3, 2, 1] is a permutation of [1, 2, 3] and vice-versa.

The NumPy Random module provides two methods for this: shuffle() and permutation().

Shuffling Arrays
Shuffle means changing arrangement of elements in-place. i.e. in the array itself.

ExampleGet your own Python Server
Randomly shuffle elements of following array:

```
from numpy import random
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

random.shuffle(arr)

print(arr)
```

The shuffle() method makes changes to the original array.

Generating Permutation of Arrays
Example
Generate a random permutation of elements of following array:

```
from numpy import random
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(random.permutation(arr))
```

The permutation() method returns a re-arranged array (and leaves the original array un-changed).

Seaborn
Visualize Distributions With Seaborn
Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.

Install Seaborn.
If you have Python and PIP already installed on a system, install it using this command:

```
C:\Users\Your Name>pip install seaborn
```

If you use Jupyter, install Seaborn using this command:

```
C:\Users\Your Name>!pip install seaborn
```

Distplots
Distplot stands for distribution plot, it takes as input an array and plots a curve corresponding to the distribution of points in the array.

Import Matplotlib
Import the pyplot object of the Matplotlib module in your code using the following statement:

```
import matplotlib.pyplot as plt
```

You can learn about the Matplotlib module in our Matplotlib Tutorial.

Import Seaborn
Import the Seaborn module in your code using the following statement:

```
import seaborn as sns
```

Plotting a Distplot

ExampleGet your own Python Server

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot([0, 1, 2, 3, 4, 5])

plt.show()
```

Plotting a Distplot Without the Histogram

Example

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot([0, 1, 2, 3, 4, 5], hist=False)

plt.show()
```

Note: We will be using: sns.distplot(arr, hist=False) to visualize random distributions in this tutorial.

Normal (Gaussian) Distribution

Normal Distribution

The Normal Distribution is one of the most important distributions.

It is also called the Gaussian Distribution after the German mathematician Carl Friedrich Gauss.

It fits the probability distribution of many events, eg. IQ Scores, Heartbeat etc.

Use the random.normal() method to get a Normal Data Distribution.

It has three parameters:

loc - (Mean) where the peak of the bell exists.

scale - (Standard Deviation) how flat the graph distribution should be.

size - The shape of the returned array.

ExampleGet your own Python Server

Generate a random normal distribution of size 2x3:

```
from numpy import random

x = random.normal(size=(2, 3))

print(x)
```

Example

Generate a random normal distribution of size 2x3 with mean at 1 and standard deviation of 2:

```
from numpy import random

x = random.normal(loc=1, scale=2, size=(2, 3))

print(x)
```

Visualization of Normal Distribution

Example

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.normal(size=1000), hist=False)
```

```
plt.show()
```
Result

Note: The curve of a Normal Distribution is also known as the Bell Curve because of the bell-shaped curve.

Binomial Distribution
Binomial Distribution
Binomial Distribution is a Discrete Distribution.

It describes the outcome of binary scenarios, e.g. toss of a coin, it will either be head or tails.

It has three parameters:

n - number of trials.

p - probability of occurence of each trial (e.g. for toss of a coin 0.5 each).

size - The shape of the returned array.

Discrete Distribution:The distribution is defined at separate set of events, e.g. a coin toss's result is discrete as it can be only head or tails whereas height of people is continuous as it can be 170, 170.1, 170.11 and so on.

ExampleGet your own Python Server
Given 10 trials for coin toss generate 10 data points:

```
from numpy import random

x = random.binomial(n=10, p=0.5, size=10)

print(x)
```
Visualization of Binomial Distribution
Example
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.binomial(n=10, p=0.5, size=1000), hist=True, kde=False)

plt.show()
```
Result

Difference Between Normal and Binomial Distribution
The main difference is that normal distribution is continous whereas binomial is discrete, but if there are enough data points it will be quite similar to normal distribution with certain loc and scale.

Example
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.normal(loc=50, scale=5, size=1000), hist=False,
label='normal')
sns.distplot(random.binomial(n=100, p=0.5, size=1000), hist=False,
label='binomial')
```

```
plt.show()
```
Result

Poisson Distribution
Poisson Distribution
Poisson Distribution is a Discrete Distribution.

It estimates how many times an event can happen in a specified time. e.g. If someone eats twice a day what is the probability he will eat thrice?

It has two parameters:

lam - rate or known number of occurrences e.g. 2 for above problem.

size - The shape of the returned array.

ExampleGet your own Python Server
Generate a random 1x10 distribution for occurrence 2:

```
from numpy import random

x = random.poisson(lam=2, size=10)

print(x)
```
Visualization of Poisson Distribution
Example
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.poisson(lam=2, size=1000), kde=False)

plt.show()
```
Result

Difference Between Normal and Poisson Distribution
Normal distribution is continuous whereas poisson is discrete.

But we can see that similar to binomial for a large enough poisson distribution it will become similar to normal distribution with certain std dev and mean.

Example
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.normal(loc=50, scale=7, size=1000), hist=False, 
label='normal')
sns.distplot(random.poisson(lam=50, size=1000), hist=False, label='poisson')

plt.show()
```
Result

Difference Between Binomial and Poisson Distribution
Binomial distribution only has two possible outcomes, whereas poisson distribution can have unlimited possible outcomes.

But for very large n and near-zero p binomial distribution is near identical to poisson distribution such that n * p is nearly equal to lam.

Example

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.binomial(n=1000, p=0.01, size=1000), hist=False,
label='binomial')
sns.distplot(random.poisson(lam=10, size=1000), hist=False, label='poisson')

plt.show()
```

Result

Uniform Distribution

Uniform Distribution

Used to describe probability where every event has equal chances of occuring.

E.g. Generation of random numbers.

It has three parameters:

a - lower bound - default 0 .0.

b - upper bound - default 1.0.

size - The shape of the returned array.

ExampleGet your own Python Server

Create a 2x3 uniform distribution sample:

```
from numpy import random

x = random.uniform(size=(2, 3))

print(x)
```

Visualization of Uniform Distribution

Example

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.uniform(size=1000), hist=False)

plt.show()
```

Result

Logistic Distribution

Logistic Distribution

Logistic Distribution is used to describe growth.

Used extensively in machine learning in logistic regression, neural networks etc.

It has three parameters:

loc - mean, where the peak is. Default 0.

scale - standard deviation, the flatness of distribution. Default 1.

size - The shape of the returned array.

ExampleGet your own Python Server
Draw 2x3 samples from a logistic distribution with mean at 1 and stddev 2.0:

```
from numpy import random

x = random.logistic(loc=1, scale=2, size=(2, 3))

print(x)
```

Visualization of Logistic Distribution
Example

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.logistic(size=1000), hist=False)

plt.show()
```

Result

Difference Between Logistic and Normal Distribution
Both distributions are near identical, but logistic distribution has more area under the tails, meaning it represents more possibility of occurrence of an event further away from mean.

For higher value of scale (standard deviation) the normal and logistic distributions are near identical apart from the peak.

Example

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.normal(scale=2, size=1000), hist=False, label='normal')
sns.distplot(random.logistic(size=1000), hist=False, label='logistic')

plt.show()
```

Result

Multinomial Distribution
Multinomial Distribution
Multinomial distribution is a generalization of binomial distribution.

It describes outcomes of multi-nomial scenarios unlike binomial where scenarios must be only one of two. e.g. Blood type of a population, dice roll outcome.

It has three parameters:

n - number of possible outcomes (e.g. 6 for dice roll).

pvals - list of probabilties of outcomes (e.g. [1/6, 1/6, 1/6, 1/6, 1/6, 1/6] for dice roll).

size - The shape of the returned array.

ExampleGet your own Python Server
Draw out a sample for dice roll:

```
from numpy import random
```

x = random.multinomial(n=6, pvals=[1/6, 1/6, 1/6, 1/6, 1/6, 1/6])

print(x)
Note: Multinomial samples will NOT produce a single value! They will produce one value for each pval.

Note: As they are generalization of binomial distribution their visual representation and similarity of normal distribution is same as that of multiple binomial distributions.

Exponential Distribution
Exponential Distribution
Exponential distribution is used for describing time till next event e.g. failure/success etc.

It has two parameters:

scale - inverse of rate (see lam in poisson distribution) defaults to 1.0.

size - The shape of the returned array.

ExampleGet your own Python Server
Draw out a sample for exponential distribution with 2.0 scale with 2x3 size:

```
from numpy import random

x = random.exponential(scale=2, size=(2, 3))

print(x)
```
Visualization of Exponential Distribution
Example
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.exponential(size=1000), hist=False)

plt.show()
```
Result

Relation Between Poisson and Exponential Distribution
Poisson distribution deals with number of occurences of an event in a time period whereas exponential distribution deals with the time between these events.

Chi Square Distribution
Chi Square Distribution
Chi Square distribution is used as a basis to verify the hypothesis.

It has two parameters:

df - (degree of freedom).

size - The shape of the returned array.

ExampleGet your own Python Server
Draw out a sample for chi squared distribution with degree of freedom 2 with size 2x3:

```
from numpy import random

x = random.chisquare(df=2, size=(2, 3))

print(x)
```
Visualization of Chi Square Distribution
Example
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.chisquare(df=1, size=1000), hist=False)

plt.show()
```
Result

Rayleigh Distribution
Rayleigh Distribution
Rayleigh distribution is used in signal processing.

It has two parameters:

scale - (standard deviation) decides how flat the distribution will be default 1.0).

size - The shape of the returned array.

ExampleGet your own Python Server
Draw out a sample for rayleigh distribution with scale of 2 with size 2x3:

```
from numpy import random

x = random.rayleigh(scale=2, size=(2, 3))

print(x)
```
Visualization of Rayleigh Distribution
Example
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.rayleigh(size=1000), hist=False)

plt.show()
```
Result

Similarity Between Rayleigh and Chi Square Distribution
At unit stddev and 2 degrees of freedom rayleigh and chi square represent the same distributions.

Pareto Distribution
Pareto Distribution
A distribution following Pareto's law i.e. 80-20 distribution (20% factors cause 80% outcome).

It has two parameter:

a - shape parameter.

size - The shape of the returned array.

ExampleGet your own Python Server
Draw out a sample for pareto distribution with shape of 2 with size 2x3:

```
from numpy import random

x = random.pareto(a=2, size=(2, 3))

print(x)
```

Visualization of Pareto Distribution
Example

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.pareto(a=2, size=1000), kde=False)

plt.show()
```

Result

Zipf Distribution
Zipf distributions are used to sample data based on zipf's law.

Zipf's Law: In a collection, the nth common term is 1/n times of the most common term. E.g. the 5th most common word in English occurs nearly 1/5 times as often as the most common word.

It has two parameters:

a - distribution parameter.

size - The shape of the returned array.

ExampleGet your own Python Server
Draw out a sample for zipf distribution with distribution parameter 2 with size 2x3:

```
from numpy import random

x = random.zipf(a=2, size=(2, 3))

print(x)
```

Visualization of Zipf Distribution
Sample 1000 points but plotting only ones with value < 10 for more meaningful chart.

Example

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

x = random.zipf(a=2, size=1000)
sns.distplot(x[x<10], kde=False)

plt.show()
```

Result

NumPy ufuncs
What are ufuncs?

ufuncs stands for "Universal Functions" and they are NumPy functions that operate on the ndarray object.

Why use ufuncs?
ufuncs are used to implement vectorization in NumPy which is way faster than iterating over elements.

They also provide broadcasting and additional methods like reduce, accumulate etc. that are very helpful for computation.

ufuncs also take additional arguments, like:

where boolean array or condition defining where the operations should take place.

dtype defining the return type of elements.

out output array where the return value should be copied.

What is Vectorization?
Converting iterative statements into a vector based operation is called vectorization.

It is faster as modern CPUs are optimized for such operations.

Add the Elements of Two Lists
list 1: [1, 2, 3, 4]

list 2: [4, 5, 6, 7]

One way of doing it is to iterate over both of the lists and then sum each elements.

ExampleGet your own Python Server
Without ufunc, we can use Python's built-in zip() method:

```
x = [1, 2, 3, 4]
y = [4, 5, 6, 7]
z = []

for i, j in zip(x, y):
  z.append(i + j)
print(z)
```

NumPy has a ufunc for this, called add(x, y) that will produce the same result.

Example
With ufunc, we can use the add() function:

```
import numpy as np

x = [1, 2, 3, 4]
y = [4, 5, 6, 7]
z = np.add(x, y)

print(z)
```

Create Your Own ufunc
How To Create Your Own ufunc
To create your own ufunc, you have to define a function, like you do with normal functions in Python, then you add it to your NumPy ufunc library with the frompyfunc() method.

The frompyfunc() method takes the following arguments:

function - the name of the function.
inputs - the number of input arguments (arrays).
outputs - the number of output arrays.
ExampleGet your own Python Server
Create your own ufunc for addition:

```
import numpy as np

def myadd(x, y):
  return x+y

myadd = np.frompyfunc(myadd, 2, 1)

print(myadd([1, 2, 3, 4], [5, 6, 7, 8]))
```

Check if a Function is a ufunc
Check the type of a function to check if it is a ufunc or not.

A ufunc should return <class 'numpy.ufunc'>.

Example
Check if a function is a ufunc:

```
import numpy as np

print(type(np.add))
```

If it is not a ufunc, it will return another type, like this built-in NumPy function for joining two or more arrays:

Example
Check the type of another function: concatenate():

```
import numpy as np

print(type(np.concatenate))
```

If the function is not recognized at all, it will return an error:

Example
Check the type of something that does not exist. This will produce an error:

```
import numpy as np

print(type(np.blahblah))
```

To test if the function is a ufunc in an if statement, use the numpy.ufunc value (or np.ufunc if you use np as an alias for numpy):

Example
Use an if statement to check if the function is a ufunc or not:

```
import numpy as np

if type(np.add) == np.ufunc:
  print('add is ufunc')
else:
  print('add is not ufunc')
```

Simple Arithmetic
Simple Arithmetic
You could use arithmetic operators + - * / directly between NumPy arrays, but this section discusses an extension of the same where we have functions that can take any array-like objects e.g. lists, tuples etc. and perform arithmetic

conditionally.

Arithmetic Conditionally: means that we can define conditions where the arithmetic operation should happen.

All of the discussed arithmetic functions take a where parameter in which we can specify that condition.

Addition
The add() function sums the content of two arrays, and return the results in a new array.

ExampleGet your own Python Server
Add the values in arr1 to the values in arr2:

```
import numpy as np

arr1 = np.array([10, 11, 12, 13, 14, 15])
arr2 = np.array([20, 21, 22, 23, 24, 25])

newarr = np.add(arr1, arr2)

print(newarr)
```

The example above will return [30 32 34 36 38 40] which is the sums of 10+20, 11+21, 12+22 etc.

Subtraction
The subtract() function subtracts the values from one array with the values from another array, and return the results in a new array.

Example
Subtract the values in arr2 from the values in arr1:

```
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([20, 21, 22, 23, 24, 25])

newarr = np.subtract(arr1, arr2)

print(newarr)
```

The example above will return [-10 -1 8 17 26 35] which is the result of 10-20, 20-21, 30-22 etc.

Multiplication
The multiply() function multiplies the values from one array with the values from another array, and return the results in a new array.

Example
Multiply the values in arr1 with the values in arr2:

```
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([20, 21, 22, 23, 24, 25])

newarr = np.multiply(arr1, arr2)

print(newarr)
```

The example above will return [200 420 660 920 1200 1500] which is the result of 10*20, 20*21, 30*22 etc.

Division
The divide() function divides the values from one array with the values from another array, and return the results in a new array.

Example
Divide the values in arr1 with the values in arr2:

```
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([3, 5, 10, 8, 2, 33])

newarr = np.divide(arr1, arr2)

print(newarr)
```

The example above will return [3.33333333 4. 3. 5. 25. 1.81818182] which is the result of 10/3, 20/5, 30/10 etc.

Power
The power() function rises the values from the first array to the power of the values of the second array, and return the results in a new array.

Example
Raise the valules in arr1 to the power of values in arr2:

```
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([3, 5, 6, 8, 2, 33])

newarr = np.power(arr1, arr2)

print(newarr)
```

The example above will return [1000 3200000 729000000 6553600000000 2500 0] which is the result of 10*10*10, 20*20*20*20*20, 30*30*30*30*30*30 etc.

Remainder
Both the mod() and the remainder() functions return the remainder of the values in the first array corresponding to the values in the second array, and return the results in a new array.

Example
Return the remainders:

```
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([3, 7, 9, 8, 2, 33])

newarr = np.mod(arr1, arr2)

print(newarr)
```

The example above will return [1 6 3 0 0 27] which is the remainders when you divide 10 with 3 (10%3), 20 with 7 (20%7) 30 with 9 (30%9) etc.

You get the same result when using the remainder() function:

Example
Return the remainders:

```
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([3, 7, 9, 8, 2, 33])
```

newarr = np.remainder(arr1, arr2)

print(newarr)
Quotient and Mod
The divmod() function return both the quotient and the the mod. The return value is two arrays, the first array contains the quotient and second array contains the mod.

Example
Return the quotient and mod:

```
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([3, 7, 9, 8, 2, 33])

newarr = np.divmod(arr1, arr2)

print(newarr)
```
The example above will return:
(array([3, 2, 3, 5, 25, 1]), array([1, 6, 3, 0, 0, 27]))
The first array represents the quotients, (the integer value when you divide 10 with 3, 20 with 7, 30 with 9 etc.
The second array represents the remainders of the same divisions.

Absolute Values
Both the absolute() and the abs() functions do the same absolute operation element-wise but we should use absolute() to avoid confusion with python's inbuilt math.abs()

Example
Return the quotient and mod:

```
import numpy as np

arr = np.array([-1, -2, 1, 2, 3, -4])

newarr = np.absolute(arr)

print(newarr)
```
The example above will return [1 2 1 2 3 4].

Rounding Decimals
Rounding Decimals
There are primarily five ways of rounding off decimals in NumPy:

truncation
fix
rounding
floor
ceil
Truncation
Remove the decimals, and return the float number closest to zero. Use the trunc() and fix() functions.

ExampleGet your own Python Server
Truncate elements of following array:

```
import numpy as np

arr = np.trunc([-3.1666, 3.6667])
```

print(arr)
Example
Same example, using fix():

```
import numpy as np

arr = np.fix([-3.1666, 3.6667])

print(arr)
```

Rounding
The around() function increments preceding digit or decimal by 1 if >=5 else do nothing.

E.g. round off to 1 decimal point, 3.16666 is 3.2

Example
Round off 3.1666 to 2 decimal places:

```
import numpy as np

arr = np.around(3.1666, 2)

print(arr)
```


Floor
The floor() function rounds off decimal to nearest lower integer.

E.g. floor of 3.166 is 3.

Example
Floor the elements of following array:

```
import numpy as np

arr = np.floor([-3.1666, 3.6667])

print(arr)
```

Ceil
The ceil() function rounds off decimal to nearest upper integer.

E.g. ceil of 3.166 is 4.

Example
Ceil the elements of following array:

```
import numpy as np

arr = np.ceil([-3.1666, 3.6667])

print(arr)
```

NumPy Logs
Logs
NumPy provides functions to perform log at the base 2, e and 10.

We will also explore how we can take log for any base by creating a custom ufunc.

All of the log functions will place -inf or inf in the elements if the log can not be computed.

Log at Base 2
Use the log2() function to perform log at the base 2.

ExampleGet your own Python Server
Find log at base 2 of all elements of following array:

```
import numpy as np

arr = np.arange(1, 10)

print(np.log2(arr))
```

Note: The arange(1, 10) function returns an array with integers starting from 1 (included) to 10 (not included).

Log at Base 10
Use the log10() function to perform log at the base 10.

Example
Find log at base 10 of all elements of following array:

```
import numpy as np

arr = np.arange(1, 10)

print(np.log10(arr))
```

Natural Log, or Log at Base e
Use the log() function to perform log at the base e.

Example
Find log at base e of all elements of following array:

```
import numpy as np

arr = np.arange(1, 10)

print(np.log(arr))
```

Log at Any Base
NumPy does not provide any function to take log at any base, so we can use the frompyfunc() function along with inbuilt function math.log() with two input parameters and one output parameter:

Example

```
from math import log
import numpy as np

nplog = np.frompyfunc(log, 2, 1)

print(nplog(100, 15))
```

NumPy Summations
Summations
What is the difference between summation and addition?

Addition is done between two arguments whereas summation happens over n elements.

ExampleGet your own Python Server
Add the values in arr1 to the values in arr2:

```
import numpy as np
```

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([1, 2, 3])

newarr = np.add(arr1, arr2)

print(newarr)
```
Returns: [2 4 6]

Example
Sum the values in arr1 and the values in arr2:

```
import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([1, 2, 3])

newarr = np.sum([arr1, arr2])

print(newarr)
```
Returns: 12

Summation Over an Axis
If you specify axis=1, NumPy will sum the numbers in each array.

Example
Perform summation in the following array over 1st axis:

```
import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([1, 2, 3])

newarr = np.sum([arr1, arr2], axis=1)

print(newarr)
```
Returns: [6 6]

Cummulative Sum
Cummulative sum means partially adding the elements in array.

E.g. The partial sum of [1, 2, 3, 4] would be [1, 1+2, 1+2+3, 1+2+3+4] = [1, 3, 6, 10].

Perfom partial sum with the cumsum() function.

Example
Perform cummulative summation in the following array:

```
import numpy as np

arr = np.array([1, 2, 3])

newarr = np.cumsum(arr)

print(newarr)
```
Returns: [1 3 6]

NumPy Products
Products
To find the product of the elements in an array, use the prod() function.

ExampleGet your own Python Server

Find the product of the elements of this array:

```
import numpy as np

arr = np.array([1, 2, 3, 4])

x = np.prod(arr)

print(x)
```

Returns: 24 because 1*2*3*4 = 24

Example
Find the product of the elements of two arrays:

```
import numpy as np

arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([5, 6, 7, 8])

x = np.prod([arr1, arr2])

print(x)
```

Returns: 40320 because 1*2*3*4*5*6*7*8 = 40320

Product Over an Axis
If you specify axis=1, NumPy will return the product of each array.

Example
Perform summation in the following array over 1st axis:

```
import numpy as np

arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([5, 6, 7, 8])

newarr = np.prod([arr1, arr2], axis=1)

print(newarr)
```

Returns: [24 1680]

Cummulative Product
Cummulative product means taking the product partially.

E.g. The partial product of [1, 2, 3, 4] is [1, 1*2, 1*2*3, 1*2*3*4] = [1, 2, 6, 24]

Perfom partial sum with the cumprod() function.

Example
Take cummulative product of all elements for following array:

```
import numpy as np

arr = np.array([5, 6, 7, 8])

newarr = np.cumprod(arr)

print(newarr)
```

Returns: [5 30 210 1680]

NumPy Differences
Differences

A discrete difference means subtracting two successive elements.

E.g. for [1, 2, 3, 4], the discrete difference would be [2-1, 3-2, 4-3] = [1, 1, 1]

To find the discrete difference, use the diff() function.

ExampleGet your own Python Server
Compute discrete difference of the following array:

```
import numpy as np

arr = np.array([10, 15, 25, 5])

newarr = np.diff(arr)

print(newarr)
```
Returns: [5 10 -20] because 15-10=5, 25-15=10, and 5-25=-20

We can perform this operation repeatedly by giving parameter n.

E.g. for [1, 2, 3, 4], the discrete difference with n = 2 would be [2-1, 3-2, 4-3] = [1, 1, 1] , then, since n=2, we will do it once more, with the new result: [1-1, 1-1] = [0, 0]
Example
Compute discrete difference of the following array twice:

```
import numpy as np

arr = np.array([10, 15, 25, 5])

newarr = np.diff(arr, n=2)

print(newarr)
```
Returns: [5 -30] because: 15-10=5, 25-15=10, and 5-25=-20 AND 10-5=5 and -20-10=-30

NumPy LCM Lowest Common Multiple
Finding LCM (Lowest Common Multiple)
The Lowest Common Multiple is the smallest number that is a common multiple of two numbers.

ExampleGet your own Python Server
Find the LCM of the following two numbers:

```
import numpy as np

num1 = 4
num2 = 6

x = np.lcm(num1, num2)

print(x)
```
Returns: 12 because that is the lowest common multiple of both numbers (4*3=12 and 6*2=12).

Finding LCM in Arrays
To find the Lowest Common Multiple of all values in an array, you can use the reduce() method.

The reduce() method will use the ufunc, in this case the lcm() function, on each element, and reduce the array by one dimension.

Example
Find the LCM of the values of the following array:

```
import numpy as np

arr = np.array([3, 6, 9])

x = np.lcm.reduce(arr)

print(x)
```

Returns: 18 because that is the lowest common multiple of all three numbers (3*6=18, 6*3=18 and 9*2=18).

Example
Find the LCM of all values of an array where the array contains all integers from 1 to 10:

```
import numpy as np

arr = np.arange(1, 11)

x = np.lcm.reduce(arr)

print(x)
```

NumPy GCD Greatest Common Denominator
Finding GCD (Greatest Common Denominator)
The GCD (Greatest Common Denominator), also known as HCF (Highest Common Factor) is the biggest number that is a common factor of both of the numbers.

ExampleGet your own Python Server
Find the HCF of the following two numbers:

```
import numpy as np

num1 = 6
num2 = 9

x = np.gcd(num1, num2)

print(x)
```

Returns: 3 because that is the highest number both numbers can be divided by (6/3=2 and 9/3=3).

Finding GCD in Arrays
To find the Highest Common Factor of all values in an array, you can use the reduce() method.

The reduce() method will use the ufunc, in this case the gcd() function, on each element, and reduce the array by one dimension.

Example
Find the GCD for all of the numbers in the following array:

```
import numpy as np

arr = np.array([20, 8, 32, 36, 16])

x = np.gcd.reduce(arr)

print(x)
```

Returns: 4 because that is the highest number all values can be divided by.

NumPy Trigonometric Functions
Trigonometric Functions
NumPy provides the ufuncs sin(), cos() and tan() that take values in radians and produce the corresponding sin, cos and tan values.

ExampleGet your own Python Server
Find sine value of PI/2:

```
import numpy as np

x = np.sin(np.pi/2)

print(x)
```
Example
Find sine values for all of the values in arr:

```
import numpy as np

arr = np.array([np.pi/2, np.pi/3, np.pi/4, np.pi/5])

x = np.sin(arr)

print(x)
```
Convert Degrees Into Radians
By default all of the trigonometric functions take radians as parameters but we can convert radians to degrees and vice versa as well in NumPy.

Note: radians values are pi/180 * degree_values.

Example
Convert all of the values in following array arr to radians:

```
import numpy as np

arr = np.array([90, 180, 270, 360])

x = np.deg2rad(arr)

print(x)
```


Radians to Degrees
Example
Convert all of the values in following array arr to degrees:

```
import numpy as np

arr = np.array([np.pi/2, np.pi, 1.5*np.pi, 2*np.pi])

x = np.rad2deg(arr)

print(x)
```
Finding Angles
Finding angles from values of sine, cos, tan. E.g. sin, cos and tan inverse (arcsin, arccos, arctan).

NumPy provides ufuncs arcsin(), arccos() and arctan() that produce radian values for corresponding sin, cos and tan values given.

Example

Find the angle of 1.0:

```
import numpy as np

x = np.arcsin(1.0)

print(x)
```

Angles of Each Value in Arrays
Example
Find the angle for all of the sine values in the array

```
import numpy as np

arr = np.array([1, -1, 0.1])

x = np.arcsin(arr)

print(x)
```

Hypotenues
Finding hypotenues using pythagoras theorem in NumPy.

NumPy provides the hypot() function that takes the base and perpendicular values and produces hypotenues based on pythagoras theorem.

Example
Find the hypotenues for 4 base and 3 perpendicular:

```
import numpy as np

base = 3
perp = 4

x = np.hypot(base, perp)

print(x)
```

NumPy Hyperbolic Functions
Hyperbolic Functions
NumPy provides the ufuncs sinh(), cosh() and tanh() that take values in radians and produce the corresponding sinh, cosh and tanh values..

ExampleGet your own Python Server
Find sinh value of PI/2:

```
import numpy as np

x = np.sinh(np.pi/2)

print(x)
```

Example
Find cosh values for all of the values in arr:

```
import numpy as np

arr = np.array([np.pi/2, np.pi/3, np.pi/4, np.pi/5])

x = np.cosh(arr)

print(x)
```

Finding Angles
Finding angles from values of hyperbolic sine, cos, tan. E.g. sinh, cosh and tanh inverse (arcsinh, arccosh, arctanh).

Numpy provides ufuncs arcsinh(), arccosh() and arctanh() that produce radian values for corresponding sinh, cosh and tanh values given.

Example
Find the angle of 1.0:

```
import numpy as np

x = np.arcsinh(1.0)

print(x)
```

Angles of Each Value in Arrays
Example
Find the angle for all of the tanh values in array:

```
import numpy as np

arr = np.array([0.1, 0.2, 0.5])

x = np.arctanh(arr)

print(x)
```

NumPy Set Operations
What is a Set
A set in mathematics is a collection of unique elements.

Sets are used for operations involving frequent intersection, union and difference operations.

Create Sets in NumPy
We can use NumPy's unique() method to find unique elements from any array. E.g. create a set array, but remember that the set arrays should only be 1-D arrays.

ExampleGet your own Python Server
Convert following array with repeated elements to a set:

```
import numpy as np

arr = np.array([1, 1, 1, 2, 3, 4, 5, 5, 6, 7])

x = np.unique(arr)

print(x)
```

Finding Union
To find the unique values of two arrays, use the union1d() method.

Example
Find union of the following two set arrays:

```
import numpy as np

arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([3, 4, 5, 6])

newarr = np.union1d(arr1, arr2)

print(newarr)
```

Finding Intersection
To find only the values that are present in both arrays, use the intersect1d() method.

Example
Find intersection of the following two set arrays:

```
import numpy as np

arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([3, 4, 5, 6])

newarr = np.intersect1d(arr1, arr2, assume_unique=True)

print(newarr)
```

Note: the intersect1d() method takes an optional argument assume_unique, which if set to True can speed up computation. It should always be set to True when dealing with sets.

Finding Difference
To find only the values in the first set that is NOT present in the seconds set, use the setdiff1d() method.

Example
Find the difference of the set1 from set2:

```
import numpy as np

set1 = np.array([1, 2, 3, 4])
set2 = np.array([3, 4, 5, 6])

newarr = np.setdiff1d(set1, set2, assume_unique=True)

print(newarr)
```

Note: the setdiff1d() method takes an optional argument assume_unique, which if set to True can speed up computation. It should always be set to True when dealing with sets.

Finding Symmetric Difference
To find only the values that are NOT present in BOTH sets, use the setxor1d() method.

Example
Find the symmetric difference of the set1 and set2:

```
import numpy as np

set1 = np.array([1, 2, 3, 4])
set2 = np.array([3, 4, 5, 6])

newarr = np.setxor1d(set1, set2, assume_unique=True)

print(newarr)
```

Note: the setxor1d() method takes an optional argument assume_unique, which if set to True can speed up computation. It should always be set to True when dealing with sets.

Pandas Tutorial
[+:
Pandas is a Python library.

Pandas is used to analyze data.

Learning by Reading
We have created 14 tutorial pages for you to learn more about Pandas.

Starting with a basic introduction and ends up with cleaning and plotting data:

Basic
Introduction
Getting Started
Pandas Series
Data Frames
Read CSV
Read JSON
Analyze Data
Cleaning Data
Clean Data
Clean Empty Cells
Clean Wrong Form
Clean Wrong Data
Remove Duplicate

Advanced
Correlations
Plotting

Learning by Quiz Test
Test your Pandas skills with a quiz test.

Learning by Exercises
Pandas Exercises
Exercise:
Insert the correct Pandas method to create a Series.

```
pd.(mylist)
```

Start the Exercise

Learning by Examples
In our "Try it Yourself" editor, you can use the Pandas module, and modify the code to see the result.

ExampleGet your own Python Server
Load a CSV file into a Pandas DataFrame:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.to_string())
```

Click on the "Try it Yourself" button to see how it works.

Pandas Introduction
What is Pandas?
Pandas is a Python library used for working with data sets.

It has functions for analyzing, cleaning, exploring, and manipulating data.

The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why Use Pandas?
Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.

:}
Data Science: is a branch of computer science where we study how to store, use and analyze data for deriving information from it.

What Can Pandas Do?
Pandas gives you answers about the data. Like:

Is there a correlation between two or more columns?
What is average value?
Max value?
Min value?
Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

Where is the Pandas Codebase?
The source code for Pandas is located at this github repository https://github.com/pandas-dev/pandas

{:
github: enables many people to work on the same codebase.

Pandas Getting Started
Installation of Pandas
If you have Python and PIP already installed on a system, then installation of Pandas is very easy.

Install it using this command:

```
C:\Users\Your Name>pip install pandas
```

If this command fails, then use a python distribution that already has Pandas installed like, Anaconda, Spyder etc.

Import Pandas
Once Pandas is installed, import it in your applications by adding the import keyword:

```
import pandas
```

Now Pandas is imported and ready to use.

ExampleGet your own Python Server

```
import pandas

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar = pandas.DataFrame(mydataset)

print(myvar)
```

w
3
s
c
h
o
o

l
s
C
E
R
T
I
F
I
E
D
.
2
0
2
2
Get Certified!
Complete the Pandas modules, do the exercises, take the exam, and you will become w3schools certified!

Pandas as pd
Pandas is usually imported under the pd alias.

alias: In Python alias are an alternate name for referring to the same thing.

Create an alias with the as keyword while importing:

```
import pandas as pd
```
Now the Pandas package can be referred to as pd instead of pandas.

Example
```
import pandas as pd

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar = pd.DataFrame(mydataset)

print(myvar)
```
Checking Pandas Version
The version string is stored under __version__ attribute.

Example
```
import pandas as pd

print(pd.__version__)
```

Pandas Series
What is a Series?
A Pandas Series is like a column in a table.

It is a one-dimensional array holding data of any type.

ExampleGet your own Python Server
Create a simple Pandas Series from a list:

```
import pandas as pd

a = [1, 7, 2]
```

```
myvar = pd.Series(a)

print(myvar)
```

Labels
If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc.

This label can be used to access a specified value.

Example
Return the first value of the Series:

```
print(myvar[0])
```

Create Labels
With the index argument, you can name your own labels.

Example
Create your own labels:

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
```

When you have created labels, you can access an item by referring to the label.

Example
Return the value of "y":

```
print(myvar["y"])
```

w
3
s
c
h
o
o
l
s
C
E
R
T
I
F
I
E
D
.
2
0
2
2
Get Certified!
Complete the Pandas modules, do the exercises, take the exam, and you will become w3schools certified!

Key/Value Objects as Series
You can also use a key/value object, like a dictionary, when creating a Series.

Example
Create a simple Pandas Series from a dictionary:

```python
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories)

print(myvar)
```
Note: The keys of the dictionary become the labels.

To select only some of the items in the dictionary, use the index argument and specify only the items you want to include in the Series.

Example
Create a Series using only data from "day1" and "day2":

```python
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories, index = ["day1", "day2"])

print(myvar)
```
DataFrames
Data sets in Pandas are usually multi-dimensional tables, called DataFrames.

Series is like a column, a DataFrame is the whole table.

Example
Create a DataFrame from two Series:

```python
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

myvar = pd.DataFrame(data)

print(myvar)
```
You will learn about DataFrames in the next chapter.

Pandas DataFrames
What is a DataFrame?
A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

ExampleGet your own Python Server
Create a simple Pandas DataFrame:

```python
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)
```

Result

```
     calories  duration
  0       420        50
  1       380        40
  2       390        45
```

Locate Row
As you can see from the result above, the DataFrame is like a table with rows and columns.

Pandas use the loc attribute to return one or more specified row(s)

Example
Return row 0:

```
#refer to the row index:
print(df.loc[0])
```

Result

```
  calories    420
  duration     50
  Name: 0, dtype: int64
```

Note: This example returns a Pandas Series.

Example
Return row 0 and 1:

```
#use a list of indexes:
print(df.loc[[0, 1]])
```

Result

```
     calories  duration
  0       420        50
  1       380        40
```

Note: When using [], the result is a Pandas DataFrame.

become w3schools certified!

Named Indexes
With the index argument, you can name your own indexes.

Example
Add a list of names to give each row a name:

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index = ["day1", "day2", "day3"])

print(df)
```

Result

```
        calories  duration
  day1       420        50
  day2       380        40
  day3       390        45
```

Locate Named Indexes
Use the named index in the loc attribute to return the specified row(s).

Example
Return "day2":

```
#refer to the named index:
print(df.loc["day2"])
```

Result

```
  calories    380
  duration     40
  Name: day2, dtype: int64
```

Load Files Into a DataFrame
If your data sets are stored in a file, Pandas can load them into a DataFrame.

Example
Load a comma separated file (CSV file) into a DataFrame:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df)
```

You will learn more about importing files in the next chapters.

Pandas Read CSV
Read CSV Files
A simple way to store big data sets is to use CSV files (comma separated files).

CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

In our examples we will be using a CSV file called 'data.csv'.

Download data.csv. or Open data.csv

ExampleGet your own Python Server
Load the CSV into a DataFrame:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.to_string())
```
Tip: use to_string() to print the entire DataFrame.

If you have a large DataFrame with many rows, Pandas will only return the first 5 rows, and the last 5 rows:

Example
Print the DataFrame without the to_string() method:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df)
```
max_rows
The number of rows returned is defined in Pandas option settings.

You can check your system's maximum rows with the pd.options.display.max_rows statement.

Example
Check the number of maximum returned rows:

```
import pandas as pd

print(pd.options.display.max_rows)
```
In my system the number is 60, which means that if the DataFrame contains more than 60 rows, the print(df) statement will return only the headers and the first and last 5 rows.

You can change the maximum rows number with the same statement.

Example
Increase the maximum number of rows to display the entire DataFrame:

```
import pandas as pd

pd.options.display.max_rows = 9999

df = pd.read_csv('data.csv')

print(df)
```

Pandas Read JSON
Read JSON
Big data sets are often stored, or extracted as JSON.

JSON is plain text, but has the format of an object, and is well known in the world of programming, including Pandas.

In our examples we will be using a JSON file called 'data.json'.

Open data.json.

ExampleGet your own Python Server
Load the JSON file into a DataFrame:

```
import pandas as pd

df = pd.read_json('data.json')

print(df.to_string())
```

Tip: use to_string() to print the entire DataFrame.

Dictionary as JSON
JSON = Python Dictionary

JSON objects have the same format as Python dictionaries.

If your JSON code is not in a file, but in a Python Dictionary, you can load it into a DataFrame directly:

Example
Load a Python Dictionary into a DataFrame:

```
import pandas as pd

data = {
  "Duration":{
    "0":60,
    "1":60,
    "2":60,
    "3":45,
    "4":45,
    "5":60
  },
  "Pulse":{
    "0":110,
    "1":117,
    "2":103,
    "3":109,
    "4":117,
    "5":102
  },
  "Maxpulse":{
    "0":130,
    "1":145,
    "2":135,
    "3":175,
    "4":148,
    "5":127
  },
  "Calories":{
    "0":409,
    "1":479,
    "2":340,
    "3":282,
    "4":406,
    "5":300
  }
}

df = pd.DataFrame(data)

print(df)
```

Pandas - Analyzing DataFrames
Viewing the Data
One of the most used method for getting a quick overview of the DataFrame, is the head() method.

The head() method returns the headers and a specified number of rows, starting from the top.

ExampleGet your own Python Server
Get a quick overview by printing the first 10 rows of the DataFrame:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.head(10))
```
In our examples we will be using a CSV file called 'data.csv'.

Download data.csv, or open data.csv in your browser.

Note: if the number of rows is not specified, the head() method will return the top 5 rows.

Example
Print the first 5 rows of the DataFrame:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.head())
```
There is also a tail() method for viewing the last rows of the DataFrame.

The tail() method returns the headers and a specified number of rows, starting from the bottom.

Example
Print the last 5 rows of the DataFrame:

```
print(df.tail())
```

Complete the Pandas modules, do the exercises, take the exam, and you will become w3schools certified!

Info About the Data
The DataFrames object has a method called info(), that gives you more information about the data set.

Example
Print information about the data:

```
print(df.info())
```

Result

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None
```

Result Explained
The result tells us there are 169 rows and 4 columns:

```
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
```

And the name of each column, with the data type:

```
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  164 non-null    float64
```

Null Values
The info() method also tells us how many Non-Null values there are present in each column, and in our data set it seems like there are 164 of 169 Non-Null values in the "Calories" column.

Which means that there are 5 rows with no value at all, in the "Calories" column, for whatever reason.

Empty values, or Null values, can be bad when analyzing data, and you should consider removing rows with empty values. This is a step towards what is called cleaning data, and you will learn more about that in the next chapters.

Pandas - Cleaning Data
Data Cleaning
Data cleaning means fixing bad data in your data set.

Bad data could be:

Empty cells

Data in wrong format
Wrong data
Duplicates
In this tutorial you will learn how to deal with all of them.

Our Data Set
In the next chapters we will use this data set:

```
      Duration          Date  Pulse  Maxpulse  Calories
  0         60  '2020/12/01'    110       130     409.1
  1         60  '2020/12/02'    117       145     479.0
  2         60  '2020/12/03'    103       135     340.0
  3         45  '2020/12/04'    109       175     282.4
  4         45  '2020/12/05'    117       148     406.0
  5         60  '2020/12/06'    102       127     300.0
  6         60  '2020/12/07'    110       136     374.0
  7        450  '2020/12/08'    104       134     253.3
  8         30  '2020/12/09'    109       133     195.1
  9         60  '2020/12/10'     98       124     269.0
  10        60  '2020/12/11'    103       147     329.3
  11        60  '2020/12/12'    100       120     250.7
  12        60  '2020/12/12'    100       120     250.7
  13        60  '2020/12/13'    106       128     345.3
  14        60  '2020/12/14'    104       132     379.3
  15        60  '2020/12/15'     98       123     275.0
  16        60  '2020/12/16'     98       120     215.2
  17        60  '2020/12/17'    100       120     300.0
  18        45  '2020/12/18'     90       112       NaN
  19        60  '2020/12/19'    103       123     323.0
  20        45  '2020/12/20'     97       125     243.0
  21        60  '2020/12/21'    108       131     364.2
  22        45           NaN    100       119     282.0
  23        60  '2020/12/23'    130       101     300.0
  24        45  '2020/12/24'    105       132     246.0
  25        60  '2020/12/25'    102       126     334.5
  26        60    2020/12/26    100       120     250.0
  27        60  '2020/12/27'     92       118     241.0
  28        60  '2020/12/28'    103       132       NaN
  29        60  '2020/12/29'    100       132     280.0
  30        60  '2020/12/30'    102       129     380.3
  31        60  '2020/12/31'     92       115     243.0
```

The data set contains some empty cells ("Date" in row 22, and "Calories" in row 18 and 28).

The data set contains wrong format ("Date" in row 26).

The data set contains wrong data ("Duration" in row 7).

The data set contains duplicates (row 11 and 12).

Pandas - Cleaning Empty Cells
Empty Cells
Empty cells can potentially give you a wrong result when you analyze data.

Remove Rows
One way to deal with empty cells is to remove rows that contain empty cells.

This is usually OK, since data sets can be very big, and removing a few rows will not have a big impact on the result.

ExampleGet your own Python Server
Return a new Data Frame with no empty cells:

```
import pandas as pd

df = pd.read_csv('data.csv')

new_df = df.dropna()

print(new_df.to_string())
```

Note: By default, the dropna() method returns a new DataFrame, and will not change the original.

If you want to change the original DataFrame, use the inplace = True argument:

Example
Remove all rows with NULL values:

```
import pandas as pd

df = pd.read_csv('data.csv')

df.dropna(inplace = True)

print(df.to_string())
```

Note: Now, the dropna(inplace = True) will NOT return a new DataFrame, but it will remove all rows containing NULL values from the original DataFrame.

Replace Empty Values
Another way of dealing with empty cells is to insert a new value instead.

This way you do not have to delete entire rows just because of some empty cells.

The fillna() method allows us to replace empty cells with a value:

Example
Replace NULL values with the number 130:

```
import pandas as pd

df = pd.read_csv('data.csv')

df.fillna(130, inplace = True)
```

Replace Only For Specified Columns
The example above replaces all empty cells in the whole Data Frame.

To only replace empty values for one column, specify the column name for the DataFrame:

Example
Replace NULL values in the "Calories" columns with the number 130:

```
import pandas as pd

df = pd.read_csv('data.csv')

df["Calories"].fillna(130, inplace = True)
```

w
3
s
c
h
o
o

l
s
C
E
R
T
I
F
I
E
D
.
2
0
2
2
Get Certified!
Complete the Pandas modules, do the exercises, take the exam, and you will become w3schools certified!

Replace Using Mean, Median, or Mode
A common way to replace empty cells, is to calculate the mean, median or mode value of the column.

Pandas uses the mean() median() and mode() methods to calculate the respective values for a specified column:

Example
Calculate the MEAN, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].mean()

df["Calories"].fillna(x, inplace = True)
```
Mean = the average value (the sum of all values divided by number of values).

Example
Calculate the MEDIAN, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].median()

df["Calories"].fillna(x, inplace = True)
```
Median = the value in the middle, after you have sorted all values ascending.

Example
Calculate the MODE, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].mode()[0]

df["Calories"].fillna(x, inplace = True)
```
Mode = the value that appears most frequently.

Pandas - Cleaning Data of Wrong Format
Data of Wrong Format
Cells with data of wrong format can make it difficult, or even impossible, to analyze data.

To fix it, you have two options: remove the rows, or convert all cells in the columns into the same format.

Convert Into a Correct Format
In our Data Frame, we have two cells with the wrong format. Check out row 22 and 26, the 'Date' column should be a string that represents a date:

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	100	120	250.7
13	60	'2020/12/13'	106	128	345.3
14	60	'2020/12/14'	104	132	379.3
15	60	'2020/12/15'	98	123	275.0
16	60	'2020/12/16'	98	120	215.2
17	60	'2020/12/17'	100	120	300.0
18	45	'2020/12/18'	90	112	NaN
19	60	'2020/12/19'	103	123	323.0
20	45	'2020/12/20'	97	125	243.0
21	60	'2020/12/21'	108	131	364.2
22	45	NaN	100	119	282.0
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	20201226	100	120	250.0
27	60	'2020/12/27'	92	118	241.0
28	60	'2020/12/28'	103	132	NaN
29	60	'2020/12/29'	100	132	280.0
30	60	'2020/12/30'	102	129	380.3
31	60	'2020/12/31'	92	115	243.0

Let's try to convert all cells in the 'Date' column into dates.

Pandas has a to_datetime() method for this:

ExampleGet your own Python Server
Convert to date:

```
import pandas as pd

df = pd.read_csv('data.csv')

df['Date'] = pd.to_datetime(df['Date'])

print(df.to_string())
```
Result:

```
    Duration          Date  Pulse  Maxpulse  Calories
  0       60  '2020/12/01'    110       130     409.1
  1       60  '2020/12/02'    117       145     479.0
  2       60  '2020/12/03'    103       135     340.0
  3       45  '2020/12/04'    109       175     282.4
  4       45  '2020/12/05'    117       148     406.0
  5       60  '2020/12/06'    102       127     300.0
  6       60  '2020/12/07'    110       136     374.0
  7      450  '2020/12/08'    104       134     253.3
  8       30  '2020/12/09'    109       133     195.1
  9       60  '2020/12/10'     98       124     269.0
  10      60  '2020/12/11'    103       147     329.3
  11      60  '2020/12/12'    100       120     250.7
  12      60  '2020/12/12'    100       120     250.7
  13      60  '2020/12/13'    106       128     345.3
  14      60  '2020/12/14'    104       132     379.3
  15      60  '2020/12/15'     98       123     275.0
  16      60  '2020/12/16'     98       120     215.2
  17      60  '2020/12/17'    100       120     300.0
  18      45  '2020/12/18'     90       112       NaN
  19      60  '2020/12/19'    103       123     323.0
  20      45  '2020/12/20'     97       125     243.0
  21      60  '2020/12/21'    108       131     364.2
  22      45           NaT    100       119     282.0
  23      60  '2020/12/23'    130       101     300.0
  24      45  '2020/12/24'    105       132     246.0
  25      60  '2020/12/25'    102       126     334.5
  26      60  '2020/12/26'    100       120     250.0
  27      60  '2020/12/27'     92       118     241.0
  28      60  '2020/12/28'    103       132       NaN
  29      60  '2020/12/29'    100       132     280.0
  30      60  '2020/12/30'    102       129     380.3
  31      60  '2020/12/31'     92       115     243.0
```

As you can see from the result, the date in row 26 was fixed, but the empty date in row 22 got a NaT (Not a Time) value, in other words an empty value. One way to deal with empty values is simply removing the entire row.

become w3schools certified!

Removing Rows
The result from the converting in the example above gave us a NaT value, which can be handled as a NULL value, and we can remove the row by using the dropna() method.

Example
Remove rows with a NULL value in the "Date" column:

```
df.dropna(subset=['Date'], inplace = True)
```

Pandas - Fixing Wrong Data
Wrong Data
"Wrong data" does not have to be "empty cells" or "wrong format", it can just be wrong, like if someone registered "199" instead of "1.99".

Sometimes you can spot wrong data by looking at the data set, because you have an expectation of what it should be.

If you take a look at our data set, you can see that in row 7, the duration is 450, but for all the other rows the duration is between 30 and 60.

It doesn't have to be wrong, but taking in consideration that this is the data set of someone's workout sessions, we conclude with the fact that this person did not work out in 450 minutes.

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	100	120	250.7
13	60	'2020/12/13'	106	128	345.3
14	60	'2020/12/14'	104	132	379.3
15	60	'2020/12/15'	98	123	275.0
16	60	'2020/12/16'	98	120	215.2
17	60	'2020/12/17'	100	120	300.0
18	45	'2020/12/18'	90	112	NaN
19	60	'2020/12/19'	103	123	323.0
20	45	'2020/12/20'	97	125	243.0
21	60	'2020/12/21'	108	131	364.2
22	45	NaN	100	119	282.0
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	20201226	100	120	250.0
27	60	'2020/12/27'	92	118	241.0
28	60	'2020/12/28'	103	132	NaN
29	60	'2020/12/29'	100	132	280.0
30	60	'2020/12/30'	102	129	380.3
31	60	'2020/12/31'	92	115	243.0

How can we fix wrong values, like the one for "Duration" in row 7?

w
3
s
c
h
o
o
l
s
C
E
R
T
I
F
I
E
D
.
2
0
2
2
Get Certified!
Complete the Pandas modules, do the exercises, take the exam, and you will become w3schools certified!

Replacing Values
One way to fix wrong values is to replace them with something else.

In our example, it is most likely a typo, and the value should be "45" instead of "450", and we could just insert "45" in row 7:

ExampleGet your own Python Server
Set "Duration" = 45 in row 7:

```
df.loc[7, 'Duration'] = 45
```

For small data sets you might be able to replace the wrong data one by one, but not for big data sets.

To replace wrong data for larger data sets you can create some rules, e.g. set some boundaries for legal values, and replace any values that are outside of the boundaries.

Example
Loop through all values in the "Duration" column.

If the value is higher than 120, set it to 120:

```
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.loc[x, "Duration"] = 120
```

Removing Rows
Another way of handling wrong data is to remove the rows that contains wrong data.

This way you do not have to find out what to replace them with, and there is a good chance you do not need them to do your analyses.

Example
Delete rows where "Duration" is higher than 120:

```
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.drop(x, inplace = True)
```

Pandas - Removing Duplicates
Discovering Duplicates
Duplicate rows are rows that have been registered more than one time.

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	100	120	250.7
13	60	'2020/12/13'	106	128	345.3
14	60	'2020/12/14'	104	132	379.3
15	60	'2020/12/15'	98	123	275.0
16	60	'2020/12/16'	98	120	215.2
17	60	'2020/12/17'	100	120	300.0
18	45	'2020/12/18'	90	112	NaN
19	60	'2020/12/19'	103	123	323.0
20	45	'2020/12/20'	97	125	243.0
21	60	'2020/12/21'	108	131	364.2
22	45	NaN	100	119	282.0
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	20201226	100	120	250.0
27	60	'2020/12/27'	92	118	241.0
28	60	'2020/12/28'	103	132	NaN
29	60	'2020/12/29'	100	132	280.0
30	60	'2020/12/30'	102	129	380.3
31	60	'2020/12/31'	92	115	243.0

By taking a look at our test data set, we can assume that row 11 and 12 are duplicates.

To discover duplicates, we can use the duplicated() method.

The duplicated() method returns a Boolean values for each row:

ExampleGet your own Python Server
Returns True for every row that is a duplicate, otherwise False:

```
print(df.duplicated())
```

w
3
s
c
h
o
o
l

s
C
E
R
T
I
F
I
E
D
.
2
0
2
2
Get Certified!
Complete the Pandas modules, do the exercises, take the exam, and you will become w3schools certified!

Removing Duplicates
To remove duplicates, use the drop_duplicates() method.

Example
Remove all duplicates:

```
df.drop_duplicates(inplace = True)
```

Remember: The (inplace = True) will make sure that the method does NOT return a new DataFrame, but it will remove all duplicates from the original DataFrame.

Pandas - Data Correlations
Finding Relationships
A great aspect of the Pandas module is the corr() method.

The corr() method calculates the relationship between each column in your data set.

The examples in this page uses a CSV file called: 'data.csv'.

Download data.csv. or Open data.csv

ExampleGet your own Python Server
Show the relationship between the columns:

```
df.corr()
```

Result

```
            Duration     Pulse  Maxpulse  Calories
  Duration  1.000000 -0.155408  0.009403  0.922721
  Pulse    -0.155408  1.000000  0.786535  0.025120
  Maxpulse  0.009403  0.786535  1.000000  0.203814
  Calories  0.922721  0.025120  0.203814  1.000000
```

Note: The corr() method ignores "not numeric" columns.

Result Explained
The Result of the corr() method is a table with a lot of numbers that represents how well the relationship is between two columns.

The number varies from -1 to 1.

1 means that there is a 1 to 1 relationship (a perfect correlation), and for this data set, each time a value went up in the first column, the other one went

up as well.

0.9 is also a good relationship, and if you increase one value, the other will probably increase as well.

-0.9 would be just as good relationship as 0.9, but if you increase one value, the other will probably go down.

0.2 means NOT a good relationship, meaning that if one value goes up does not mean that the other will.

What is a good correlation? It depends on the use, but I think it is safe to say you have to have at least 0.6 (or -0.6) to call it a good correlation.

Perfect Correlation:
We can see that "Duration" and "Duration" got the number 1.000000, which makes sense, each column always has a perfect relationship with itself.

Good Correlation:
"Duration" and "Calories" got a 0.922721 correlation, which is a very good correlation, and we can predict that the longer you work out, the more calories you burn, and the other way around: if you burned a lot of calories, you probably had a long work out.

Bad Correlation:
"Duration" and "Maxpulse" got a 0.009403 correlation, which is a very bad correlation, meaning that we can not predict the max pulse by just looking at the duration of the work out, and vice versa.

Pandas - Plotting

Plotting
Pandas uses the plot() method to create diagrams.

We can use Pyplot, a submodule of the Matplotlib library to visualize the diagram on the screen.

Read more about Matplotlib in our Matplotlib Tutorial.

ExampleGet your own Python Server
Import pyplot from Matplotlib and visualize our DataFrame:

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('data.csv')

df.plot()

plt.show()
```

The examples in this page uses a CSV file called: 'data.csv'.

Download data.csv or Open data.csv

Scatter Plot
Specify that you want a scatter plot with the kind argument:

kind = 'scatter'

A scatter plot needs an x- and a y-axis.

In the example below we will use "Duration" for the x-axis and "Calories" for the y-axis.

Include the x and y arguments like this:

x = 'Duration', y = 'Calories'

Example

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('data.csv')

df.plot(kind = 'scatter', x = 'Duration', y = 'Calories')

plt.show()
```

Result

Remember: In the previous example, we learned that the correlation between "Duration" and "Calories" was 0.922721, and we concluded with the fact that higher duration means more calories burned.

By looking at the scatterplot, I will agree.

Let's create another scatterplot, where there is a bad relationship between the columns, like "Duration" and "Maxpulse", with the correlation 0.009403:

Example

A scatterplot where there are no relationship between the columns:

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('data.csv')

df.plot(kind = 'scatter', x = 'Duration', y = 'Maxpulse')

plt.show()
```

Result

Get Certified!
Complete the Pandas modules, do the exercises, take the exam, and you will become w3schools certified!

Histogram
Use the kind argument to specify that you want a histogram:

kind = 'hist'

A histogram needs only one column.

A histogram shows us the frequency of each interval, e.g. how many workouts lasted between 50 and 60 minutes?

In the example below we will use the "Duration" column to create the histogram:

Example
```
df["Duration"].plot(kind = 'hist')
```
Result

Note: The histogram tells us that there were over 100 workouts that lasted between 50 and 60 minutes.

Pandas - DataFrame Reference
All properties and methods of the DataFrame object, with explanations and examples:

Property/Method	Description
abs() Return a DataFrame with the absolute value of each value
add() Adds the values of a DataFrame with the specified value(s)
add_prefix()	Prefix all labels
add_suffix()	Suffix all labels
agg() Apply a function or a function name to one of the axis of the DataFrame
aggregate() Apply a function or a function name to one of the axis of the DataFrame
align()	Aligns two DataFrames with a specified join method
all() Return True if all values in the DataFrame are True, otherwise False
any() Returns True if any of the values in the DataFrame are True, otherwise False
append()	Append new columns
applymap()	Execute a function for each element in the DataFrame
apply()	Apply a function to one of the axis of the DataFrame
assign()	Assign new columns
astype()	Convert the DataFrame into a specified dtype
at	Get or set the value of the item with the specified label
axes	Returns the labels of the rows and the columns of the DataFrame
bfill()	Replaces NULL values with the value from the next row
bool()	Returns the Boolean value of the DataFrame
columns	Returns the column labels of the DataFrame
combine()	Compare the values in two DataFrames, and let a function decide which values to keep
combine_first()	Compare two DataFrames, and if the first DataFrame has a NULL value, it will be filled with the respective value from the second DataFrame
compare()	Compare two DataFrames and return the differences
convert_dtypes()	Converts the columns in the DataFrame into new dtypes
corr()	Find the correlation (relationship) between each column
count()	Returns the number of not empty cells for each column/row
cov() Find the covariance of the columns
copy()	Returns a copy of the DataFrame
cummax()	Calculate the cumulative maximum values of the DataFrame
cummin()	Calculate the cumulative minmum values of the DataFrame

cumprod()   Calculate the cumulative product over the DataFrame
cumsum()   Calculate the cumulative sum over the DataFrame
describe() Returns a description summary for each column in the DataFrame
diff()   Calculate the difference between a value and the value of the same column in the previous row
div() Divides the values of a DataFrame with the specified value(s)
dot() Multiplies the values of a DataFrame with values from another array-like object, and add the result
drop()   Drops the specified rows/columns from the DataFrame
drop_duplicates() Drops duplicate values from the DataFrame
droplevel() Drops the specified index/column(s)
dropna()   Drops all rows that contains NULL values
dtypes   Returns the dtypes of the columns of the DataFrame
duplicated()   Returns True for duplicated rows, otherwise False
empty Returns True if the DataFrame is empty, otherwise False
eq() Returns True for values that are equal to the specified value(s), otherwise False
equals()   Returns True if two DataFrames are equal, otherwise False
eval Evaluate a specified string
explode()   Converts each element into a row
ffill()   Replaces NULL values with the value from the previous row
fillna()   Replaces NULL values with the specified value
filter()   Filter the DataFrame according to the specified filter
first()   Returns the first rows of a specified date selection
floordiv() Divides the values of a DataFrame with the specified value(s), and floor the values
ge() Returns True for values greater than, or equal to the specified value(s), otherwise False
get() Returns the item of the specified key
groupby()   Groups the rows/columns into specified groups
gt() Returns True for values greater than the specified value(s), otherwise False
head()   Returns the header row and the first 10 rows, or the specified number of rows
iat   Get or set the value of the item in the specified position
idxmax()   Returns the label of the max value in the specified axis
idxmin()   Returns the label of the min value in the specified axis
iloc Get or set the values of a group of elements in the specified positions
index Returns the row labels of the DataFrame
infer_objects()   Change the dtype of the columns in the DataFrame
info()   Prints information about the DataFrame
insert()   Insert a column in the DataFrame
interpolate()   Replaces not-a-number values with the interpolated method
isin()   Returns True if each elements in the DataFrame is in the specified value
isna()   Finds not-a-number values
isnull()   Finds NULL values
items()   Iterate over the columns of the DataFrame
iteritems() Iterate over the columns of the DataFrame
iterrows() Iterate over the rows of the DataFrame
itertuples()   Iterate over the rows as named tuples
join()   Join columns of another DataFrame
last()   Returns the last rows of a specified date selection
le() Returns True for values less than, or equal to the specified value(s), otherwise False
loc   Get or set the value of a group of elements specified using their labels
lt() Returns True for values less than the specified value(s), otherwise False
keys()   Returns the keys of the info axis
kurtosis() Returns the kurtosis of the values in the specified axis
mask()   Replace all values where the specified condition is True
max() Return the max of the values in the specified axis
mean()   Return the mean of the values in the specified axis
median()   Return the median of the values in the specified axis
melt()   Reshape the DataFrame from a wide table to a long table

memory_usage()   Returns the memory usage of each column
merge()   Merge DataFrame objects
min() Returns the min of the values in the specified axis
mod() Modules (find the remainder) of the values of a DataFrame
mode()   Returns the mode of the values in the specified axis
mul() Multiplies the values of a DataFrame with the specified value(s)
ndim Returns the number of dimensions of the DataFrame
ne() Returns True for values that are not equal to the specified value(s), otherwise False
nlargest() Sort the DataFrame by the specified columns, descending, and return the specified number of rows
notna()   Finds values that are not not-a-number
notnull()   Finds values that are not NULL
nsmallest() Sort the DataFrame by the specified columns, ascending, and return the specified number of rows
nunique()   Returns the number of unique values in the specified axis
pct_change()   Returns the percentage change between the previous and the current value
pipe()   Apply a function to the DataFrame
pivot()   Re-shape the DataFrame
pivot_table()   Create a spreadsheet pivot table as a DataFrame
pop() Removes an element from the DataFrame
pow() Raise the values of one DataFrame to the values of another DataFrame
prod()   Returns the product of all values in the specified axis
product()   Returns the product of the values in the specified axis
quantile() Returns the values at the specified quantile of the specified axis
query()   Query the DataFrame
radd()   Reverse-adds the values of one DataFrame with the values of another DataFrame
rdiv()   Reverse-divides the values of one DataFrame with the values of another DataFrame
reindex()   Change the labels of the DataFrame
reindex_like()   ??
rename()   Change the labels of the axes
rename_axis()   Change the name of the axis
reorder_levels() Re-order the index levels
replace()   Replace the specified values
reset_index()   Reset the index
rfloordiv() Reverse-divides the values of one DataFrame with the values of another DataFrame
rmod()   Reverse-modules the values of one DataFrame to the values of another DataFrame
rmul()   Reverse-multiplies the values of one DataFrame with the values of another DataFrame
round()   Returns a DataFrame with all values rounded into the specified format
rpow()   Reverse-raises the values of one DataFrame up to the values of another DataFrame
rsub()   Reverse-subtracts the values of one DataFrame to the values of another DataFrame
rtruediv() Reverse-divides the values of one DataFrame with the values of another DataFrame
sample()   Returns a random selection elements
sem() Returns the standard error of the mean in the specified axis
select_dtypes()   Returns a DataFrame with columns of selected data types
shape Returns the number of rows and columns of the DataFrame
set_axis() Sets the index of the specified axis
set_flags() Returns a new DataFrame with the specified flags
set_index() Set the Index of the DataFrame
size Returns the number of elements in the DataFrame
skew()   Returns the skew of the values in the specified axis
sort_index()   Sorts the DataFrame according to the labels
sort_values()   Sorts the DataFrame according to the values
squeeze()   Converts a single column DataFrame into a Series

stack()	Reshape the DataFrame from a wide table to a long table
std()	Returns the standard deviation of the values in the specified axis
sum()	Returns the sum of the values in the specified axis
sub()	Subtracts the values of a DataFrame with the specified value(s)
swaplevel()	Swaps the two specified levels
T	Turns rows into columns and columns into rows
tail()	Returns the headers and the last rows
take()	Returns the specified elements
to_xarray()	Returns an xarray object
transform()	Execute a function for each value in the DataFrame
transpose()	Turns rows into columns and columns into rows
truediv()	Divides the values of a DataFrame with the specified value(s)
truncate()	Removes elements outside of a specified set of values
update()	Update one DataFrame with the values from another DataFrame
value_counts()	Returns the number of unique rows
values	Returns the DataFrame as a NumPy array
var()	Returns the variance of the values in the specified axis
where()	Replace all values where the specified condition is False
xs()	Returns the cross-section of the DataFrame
__iter__()	Returns an iterator of the info axes

SciPy Tutorial
[+:
SciPy is a scientific computation library that uses NumPy underneath.

SciPy stands for Scientific Python.

Learning by Reading
We have created 10 tutorial pages for you to learn the fundamentals of SciPy:

Basic SciPy
Introduction
Getting Started
Constants
Optimizers
Sparse Data
Graphs
Spatial Data
Matlab Arrays
Interpolation
Significance Tests

Learning by Quiz Test
Test your SciPy skills with a quiz test.

Learning by Exercises
SciPy Exercises
Exercise:
Insert the correct syntax for printing the kilometer unit (in meters):

```
print(constants.);
```

Start the Exercise

Learning by Examples
In our "Try it Yourself" editor, you can use the SciPy module, and modify the code to see the result.

Example
How many cubic meters are in one liter:

```
from scipy import constants

print(constants.liter)
```
Click on the "Try it Yourself" button to see how it works.

SciPy Introduction
What is SciPy?
SciPy is a scientific computation library that uses NumPy underneath.

SciPy stands for Scientific Python.

It provides more utility functions for optimization, stats and signal processing.

Like NumPy, SciPy is open source so we can use it freely.

SciPy was created by NumPy's creator Travis Olliphant.

Why Use SciPy?
If SciPy uses NumPy underneath, why can we not just use NumPy?

SciPy has optimized and added functions that are frequently used in NumPy and Data Science.

Which Language is SciPy Written in?
SciPy is predominantly written in Python, but a few segments are written in C.

Where is the SciPy Codebase?
The source code for SciPy is located at this github repository https://github.com/scipy/scipy

github: enables many people to work on the same codebase.

SciPy Getting Started
Installation of SciPy
If you have Python and PIP already installed on a system, then installation of SciPy is very easy.

Install it using this command:

```
C:\Users\Your Name>pip install scipy
```
If this command fails, then use a Python distribution that already has SciPy installed like, Anaconda, Spyder etc.

Import SciPy
Once SciPy is installed, import the SciPy module(s) you want to use in your applications by adding the from scipy import module statement:

```
from scipy import constants
```
Now we have imported the constants module from SciPy, and the application is ready to use it:

Example
How many cubic meters are in one liter:

```
from scipy import constants

print(constants.liter)
```
constants: SciPy offers a set of mathematical constants, one of them is liter

which returns 1 liter as cubic meters.

You will learn more about constants in the next chapter.

Checking SciPy Version
The version string is stored under the __version__ attribute.

Example
```
import scipy

print(scipy.__version__)
```
Note: two underscore characters are used in __version__.

SciPy Constants
Constants in SciPy
As SciPy is more focused on scientific implementations, it provides many built-in scientific constants.

These constants can be helpful when you are working with Data Science.

PI is an example of a scientific constant.

Example
Print the constant value of PI:

```
from scipy import constants

print(constants.pi)
```
Constant Units
A list of all units under the constants module can be seen using the dir() function.

Example
List all constants:

```
from scipy import constants

print(dir(constants))
```
Unit Categories
The units are placed under these categories:

Metric
Binary
Mass
Angle
Time
Length
Pressure
Volume
Speed
Temperature
Energy
Power
Force

Metric (SI) Prefixes:
Return the specified unit in meter (e.g. centi returns 0.01)

Example
```
from scipy import constants
```

```
print(constants.yotta)    #1e+24
print(constants.zetta)    #1e+21
print(constants.exa)      #1e+18
print(constants.peta)     #1000000000000000.0
print(constants.tera)     #1000000000000.0
print(constants.giga)     #1000000000.0
print(constants.mega)     #1000000.0
print(constants.kilo)     #1000.0
print(constants.hecto)    #100.0
print(constants.deka)     #10.0
print(constants.deci)     #0.1
print(constants.centi)    #0.01
print(constants.milli)    #0.001
print(constants.micro)    #1e-06
print(constants.nano)     #1e-09
print(constants.pico)     #1e-12
print(constants.femto)    #1e-15
print(constants.atto)     #1e-18
print(constants.zepto)    #1e-21
```

Binary Prefixes:
Return the specified unit in bytes (e.g. kibi returns 1024)

Example

```
from scipy import constants

print(constants.kibi)    #1024
print(constants.mebi)    #1048576
print(constants.gibi)    #1073741824
print(constants.tebi)    #1099511627776
print(constants.pebi)    #1125899906842624
print(constants.exbi)    #1152921504606846976
print(constants.zebi)    #1180591620717411303424
print(constants.yobi)    #1208925819614629174706176
```

Mass:
Return the specified unit in kg (e.g. gram returns 0.001)

Example

```
from scipy import constants

print(constants.gram)        #0.001
print(constants.metric_ton)  #1000.0
print(constants.grain)       #6.479891e-05
print(constants.lb)          #0.45359236999999997
print(constants.pound)       #0.45359236999999997
print(constants.oz)          #0.028349523124999998
print(constants.ounce)       #0.028349523124999998
print(constants.stone)       #6.3502931799999995
print(constants.long_ton)    #1016.0469088
print(constants.short_ton)   #907.1847399999999
print(constants.troy_ounce)  #0.031103476799999998
print(constants.troy_pound)  #0.37324172159999996
print(constants.carat)       #0.0002
print(constants.atomic_mass) #1.66053904e-27
print(constants.m_u)         #1.66053904e-27
print(constants.u)           #1.66053904e-27
```

Angle:
Return the specified unit in radians (e.g. degree returns 0.017453292519943295)

Example

```
from scipy import constants

print(constants.degree)     #0.017453292519943295
print(constants.arcmin)     #0.0002908882086657216
print(constants.arcminute)  #0.0002908882086657216
```

```
print(constants.arcsec)     #4.84813681109536e-06
print(constants.arcsecond)  #4.84813681109536e-06
```

Time:
Return the specified unit in seconds (e.g. hour returns 3600.0)

Example
```
from scipy import constants

print(constants.minute)      #60.0
print(constants.hour)        #3600.0
print(constants.day)         #86400.0
print(constants.week)        #604800.0
print(constants.year)        #31536000.0
print(constants.Julian_year) #31557600.0
```

Length:
Return the specified unit in meters (e.g. nautical_mile returns 1852.0)

Example
```
from scipy import constants

print(constants.inch)              #0.0254
print(constants.foot)              #0.30479999999999996
print(constants.yard)              #0.9143999999999999
print(constants.mile)              #1609.3439999999998
print(constants.mil)               #2.5399999999999997e-05
print(constants.pt)                #0.00035277777777777776
print(constants.point)             #0.00035277777777777776
print(constants.survey_foot)       #0.3048006096012192
print(constants.survey_mile)       #1609.3472186944373
print(constants.nautical_mile)     #1852.0
print(constants.fermi)             #1e-15
print(constants.angstrom)          #1e-10
print(constants.micron)            #1e-06
print(constants.au)                #149597870691.0
print(constants.astronomical_unit) #149597870691.0
print(constants.light_year)        #9460730472580800.0
print(constants.parsec)            #3.0856775813057292e+16
```

Pressure:
Return the specified unit in pascals (e.g. psi returns 6894.757293168361)

Example
```
from scipy import constants

print(constants.atm)         #101325.0
print(constants.atmosphere)  #101325.0
print(constants.bar)         #100000.0
print(constants.torr)        #133.32236842105263
print(constants.mmHg)        #133.32236842105263
print(constants.psi)         #6894.757293168361
```

Area:
Return the specified unit in square meters(e.g. hectare returns 10000.0)

Example
```
from scipy import constants

print(constants.hectare) #10000.0
print(constants.acre)    #4046.8564223999992
```

Volume:
Return the specified unit in cubic meters (e.g. liter returns 0.001)

Example
```
from scipy import constants

print(constants.liter)            #0.001
```

```
print(constants.litre)            #0.001
print(constants.gallon)           #0.0037854117839999997
print(constants.gallon_US)        #0.0037854117839999997
print(constants.gallon_imp)       #0.00454609
print(constants.fluid_ounce)      #2.9573529562499998e-05
print(constants.fluid_ounce_US)   #2.9573529562499998e-05
print(constants.fluid_ounce_imp)  #2.84130625e-05
print(constants.barrel)           #0.15898729492799998
print(constants.bbl)              #0.15898729492799998
```

Speed:
Return the specified unit in meters per second (e.g. speed_of_sound returns 340.5)

Example

```
from scipy import constants

print(constants.kmh)            #0.2777777777777778
print(constants.mph)            #0.44703999999999994
print(constants.mach)           #340.5
print(constants.speed_of_sound) #340.5
print(constants.knot)           #0.5144444444444445
```

Temperature:
Return the specified unit in Kelvin (e.g. zero_Celsius returns 273.15)

Example

```
from scipy import constants

print(constants.zero_Celsius)      #273.15
print(constants.degree_Fahrenheit) #0.5555555555555556
```

Energy:
Return the specified unit in joules (e.g. calorie returns 4.184)

Example

```
from scipy import constants

print(constants.eV)            #1.6021766208e-19
print(constants.electron_volt) #1.6021766208e-19
print(constants.calorie)       #4.184
print(constants.calorie_th)    #4.184
print(constants.calorie_IT)    #4.1868
print(constants.erg)           #1e-07
print(constants.Btu)           #1055.05585262
print(constants.Btu_IT)        #1055.05585262
print(constants.Btu_th)        #1054.3502644888888
print(constants.ton_TNT)       #4184000000.0
```

Power:
Return the specified unit in watts (e.g. horsepower returns 745.6998715822701)

Example

```
from scipy import constants

print(constants.hp)         #745.6998715822701
print(constants.horsepower) #745.6998715822701
```

Force:
Return the specified unit in newton (e.g. kilogram_force returns 9.80665)

Example

```
from scipy import constants

print(constants.dyn)             #1e-05
print(constants.dyne)            #1e-05
print(constants.lbf)             #4.4482216152605
print(constants.pound_force)     #4.4482216152605
print(constants.kgf)             #9.80665
```

```
print(constants.kilogram_force)  #9.80665
```

SciPy Optimizers
Optimizers in SciPy
Optimizers are a set of procedures defined in SciPy that either find the minimum value of a function, or the root of an equation.

Optimizing Functions
Essentially, all of the algorithms in Machine Learning are nothing more than a complex equation that needs to be minimized with the help of given data.

Roots of an Equation
NumPy is capable of finding roots for polynomials and linear equations, but it can not find roots for non linear equations, like this one:

x + cos(x)

For that you can use SciPy's optimze.root function.

This function takes two required arguments:

fun - a function representing an equation.

x0 - an initial guess for the root.

The function returns an object with information regarding the solution.

The actual solution is given under attribute x of the returned object:

Example
Find root of the equation x + cos(x):

```
from scipy.optimize import root
from math import cos

def eqn(x):
  return x + cos(x)

myroot = root(eqn, 0)

print(myroot.x)
```

Note: The returned object has much more information about the solution.

Example
Print all information about the solution (not just x which is the root)

```
print(myroot)
```


Minimizing a Function
A function, in this context, represents a curve, curves have high points and low points.

High points are called maxima.

Low points are called minima.

The highest point in the whole curve is called global maxima, whereas the rest of them are called local maxima.

The lowest point in whole curve is called global minima, whereas the rest of them are called local minima.

Finding Minima
We can use scipy.optimize.minimize() function to minimize the function.

The minimize() function takes the following arguments:

fun - a function representing an equation.

x0 - an initial guess for the root.

method - name of the method to use. Legal values:
    'CG'
    'BFGS'
    'Newton-CG'
    'L-BFGS-B'
    'TNC'
    'COBYLA'
    'SLSQP'

callback - function called after each iteration of optimization.

options - a dictionary defining extra params:

```
{
     "disp": boolean - print detailed description
     "gtol": number - the tolerance of the error
  }
```

Example
Minimize the function x^2 + x + 2 with BFGS:

```
from scipy.optimize import minimize

def eqn(x):
  return x**2 + x + 2

mymin = minimize(eqn, 0, method='BFGS')

print(mymin)
```

SciPy Sparse Data
What is Sparse Data
Sparse data is data that has mostly unused elements (elements that don't carry any information).

It can be an array like this one:

[1, 0, 2, 0, 0, 3, 0, 0, 0, 0, 0, 0]

Sparse Data: is a data set where most of the item values are zero.

Dense Array: is the opposite of a sparse array: most of the values are not zero.

In scientific computing, when we are dealing with partial derivatives in linear algebra we will come across sparse data.

How to Work With Sparse Data
SciPy has a module, scipy.sparse that provides functions to deal with sparse data.

There are primarily two types of sparse matrices that we use:

CSC - Compressed Sparse Column. For efficient arithmetic, fast column slicing.

CSR - Compressed Sparse Row. For fast row slicing, faster matrix vector products

We will use the CSR matrix in this tutorial.

CSR Matrix
We can create CSR matrix by passing an arrray into function scipy.sparse.csr_matrix().

Example
Create a CSR matrix from an array:

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([0, 0, 0, 0, 0, 1, 1, 0, 2])

print(csr_matrix(arr))
```

The example above returns:

```
  (0, 5)    1
  (0, 6)    1
  (0, 8)    2
```

From the result we can see that there are 3 items with value.

The 1. item is in row 0 position 5 and has the value 1.

The 2. item is in row 0 position 6 and has the value 1.

The 3. item is in row 0 position 8 and has the value 2.

Sparse Matrix Methods
Viewing stored data (not the zero items) with the data property:

Example

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])

print(csr_matrix(arr).data)
```

Counting nonzeros with the count_nonzero() method:

Example

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])

print(csr_matrix(arr).count_nonzero())
```

Removing zero-entries from the matrix with the eliminate_zeros() method:

Example

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])

mat = csr_matrix(arr)
mat.eliminate_zeros()
```

```
print(mat)
```

Eliminating duplicate entries with the sum_duplicates() method:

Example

Eliminating duplicates by adding them:

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])

mat = csr_matrix(arr)
mat.sum_duplicates()

print(mat)
```

Converting from csr to csc with the tocsc() method:

Example

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])

newarr = csr_matrix(arr).tocsc()

print(newarr)
```

Note: Apart from the mentioned sparse specific operations, sparse matrices support all of the operations that normal matrices support e.g. reshaping, summing, arithemetic, broadcasting etc.

SciPy Graphs

Working with Graphs

Graphs are an essential data structure.

SciPy provides us with the module scipy.sparse.csgraph for working with such data structures.

Adjacency Matrix

Adjacency matrix is a nxn matrix where n is the number of elements in a graph.

And the values represents the connection between the elements.

Example:

For a graph like this, with elements A, B and C, the connections are:

A & B are connected with weight 1.

A & C are connected with weight 2.

C & B is not connected.

The Adjency Matrix would look like this:

```
      A B C
   A:[0 1 2]
   B:[1 0 0]
   C:[2 0 0]
```

Below follows some of the most used methods for working with adjacency matrices.

Connected Components
Find all of the connected components with the connected_components() method.
Example
```
import numpy as np
from scipy.sparse.csgraph import connected_components
from scipy.sparse import csr_matrix

arr = np.array([
  [0, 1, 2],
  [1, 0, 0],
  [2, 0, 0]
])

newarr = csr_matrix(arr)

print(connected_components(newarr))
```


Dijkstra
Use the dijkstra method to find the shortest path in a graph from one element to another.

It takes following arguments:

return_predecessors: boolean (True to return whole path of traversal otherwise False).
indices: index of the element to return all paths from that element only.
limit: max weight of path.
Example
Find the shortest path from element 1 to 2:

```
import numpy as np
from scipy.sparse.csgraph import dijkstra
from scipy.sparse import csr_matrix

arr = np.array([
  [0, 1, 2],
  [1, 0, 0],
  [2, 0, 0]
])

newarr = csr_matrix(arr)

print(dijkstra(newarr, return_predecessors=True, indices=0))
```
Floyd Warshall
Use the floyd_warshall() method to find shortest path between all pairs of elements.

Example
Find the shortest path between all pairs of elements:

```
import numpy as np
from scipy.sparse.csgraph import floyd_warshall
from scipy.sparse import csr_matrix

arr = np.array([
  [0, 1, 2],
  [1, 0, 0],
  [2, 0, 0]
])

newarr = csr_matrix(arr)
```

```
print(floyd_warshall(newarr, return_predecessors=True))
```

Bellman Ford

The bellman_ford() method can also find the shortest path between all pairs of elements, but this method can handle negative weights as well.

Example

Find shortest path from element 1 to 2 with given graph with a negative weight:

```
import numpy as np
from scipy.sparse.csgraph import bellman_ford
from scipy.sparse import csr_matrix

arr = np.array([
  [0, -1, 2],
  [1, 0, 0],
  [2, 0, 0]
])

newarr = csr_matrix(arr)

print(bellman_ford(newarr, return_predecessors=True, indices=0))
```

Depth First Order

The depth_first_order() method returns a depth first traversal from a node.

This function takes following arguments:

the graph.
the starting element to traverse graph from.

Example

Traverse the graph depth first for given adjacency matrix:

```
import numpy as np
from scipy.sparse.csgraph import depth_first_order
from scipy.sparse import csr_matrix

arr = np.array([
  [0, 1, 0, 1],
  [1, 1, 1, 1],
  [2, 1, 1, 0],
  [0, 1, 0, 1]
])

newarr = csr_matrix(arr)

print(depth_first_order(newarr, 1))
```

Breadth First Order

The breadth_first_order() method returns a breadth first traversal from a node.

This function takes following arguments:

the graph.
the starting element to traverse graph from.

Example

Traverse the graph breadth first for given adjacency matrix:

```
import numpy as np
from scipy.sparse.csgraph import breadth_first_order
from scipy.sparse import csr_matrix

arr = np.array([
  [0, 1, 0, 1],
  [1, 1, 1, 1],
  [2, 1, 1, 0],
```

```
  [0, 1, 0, 1]
])

newarr = csr_matrix(arr)

print(breadth_first_order(newarr, 1))
```

SciPy Spatial Data
Working with Spatial Data
Spatial data refers to data that is represented in a geometric space.

E.g. points on a coordinate system.

We deal with spatial data problems on many tasks.

E.g. finding if a point is inside a boundary or not.

SciPy provides us with the module scipy.spatial, which has functions for working with spatial data.

Triangulation
A Triangulation of a polygon is to divide the polygon into multiple triangles with which we can compute an area of the polygon.

A Triangulation with points means creating surface composed triangles in which all of the given points are on at least one vertex of any triangle in the surface.

One method to generate these triangulations through points is the Delaunay() Triangulation.

Example
Create a triangulation from following points:

```
import numpy as np
from scipy.spatial import Delaunay
import matplotlib.pyplot as plt

points = np.array([
  [2, 4],
  [3, 4],
  [3, 0],
  [2, 2],
  [4, 1]
])

simplices = Delaunay(points).simplices

plt.triplot(points[:, 0], points[:, 1], simplices)
plt.scatter(points[:, 0], points[:, 1], color='r')

plt.show()
```

Result:

Note: The simplices property creates a generalization of the triangle notation.

Convex Hull
A convex hull is the smallest polygon that covers all of the given points.

Use the ConvexHull() method to create a Convex Hull.

Example
Create a convex hull for following points:

```
import numpy as np
from scipy.spatial import ConvexHull
import matplotlib.pyplot as plt

points = np.array([
  [2, 4],
  [3, 4],
  [3, 0],
  [2, 2],
  [4, 1],
  [1, 2],
  [5, 0],
  [3, 1],
  [1, 2],
  [0, 2]
])

hull = ConvexHull(points)
hull_points = hull.simplices

plt.scatter(points[:,0], points[:,1])
for simplex in hull_points:
  plt.plot(points[simplex,0], points[simplex,1], 'k-')

plt.show()
```

Result:

KDTrees
KDTrees are a datastructure optimized for nearest neighbor queries.

E.g. in a set of points using KDTrees we can efficiently ask which points are nearest to a certain given point.

The KDTree() method returns a KDTree object.

The query() method returns the distance to the nearest neighbor and the location of the neighbors.

Example
Find the nearest neighbor to point (1,1):

```
from scipy.spatial import KDTree

points = [(1, -1), (2, 3), (-2, 3), (2, -3)]

kdtree = KDTree(points)

res = kdtree.query((1, 1))

print(res)
```

Result:

```
 (2.0, 0)
```

Distance Matrix
There are many Distance Metrics used to find various types of distances between two points in data science, Euclidean distsance, cosine distsance etc.

The distance between two vectors may not only be the length of straight line between them, it can also be the angle between them from origin, or number of unit steps required etc.

Many of the Machine Learning algorithm's performance depends greatly on distance metrices. E.g. "K Nearest Neighbors", or "K Means" etc.

Let us look at some of the Distance Metrices:

Euclidean Distance
Find the euclidean distance between given points.

Example
```
from scipy.spatial.distance import euclidean

p1 = (1, 0)
p2 = (10, 2)

res = euclidean(p1, p2)

print(res)
```
Result:

```
 9.21954445729
```

Cityblock Distance (Manhattan Distance)
Is the distance computed using 4 degrees of movement.

E.g. we can only move: up, down, right, or left, not diagonally.

Example
Find the cityblock distance between given points:

```
from scipy.spatial.distance import cityblock

p1 = (1, 0)
p2 = (10, 2)

res = cityblock(p1, p2)

print(res)
```
Result:

```
 11
```

Cosine Distance
Is the value of cosine angle between the two points A and B.

Example
Find the cosine distsance between given points:

```
from scipy.spatial.distance import cosine

p1 = (1, 0)
p2 = (10, 2)

res = cosine(p1, p2)

print(res)
```
Result:

```
 0.019419324309079777
```

Hamming Distance

Is the proportion of bits where two bits are different.

It's a way to measure distance for binary sequences.

Example
Find the hamming distance between given points:

```
from scipy.spatial.distance import hamming

p1 = (True, False, True)
p2 = (False, True, True)

res = hamming(p1, p2)

print(res)
```

Result:

```
 0.666666666667
```

SciPy Matlab Arrays
Working With Matlab Arrays
We know that NumPy provides us with methods to persist the data in readable formats for Python. But SciPy provides us with interoperability with Matlab as well.

SciPy provides us with the module scipy.io, which has functions for working with Matlab arrays.

Exporting Data in Matlab Format
The savemat() function allows us to export data in Matlab format.

The method takes the following parameters:

filename - the file name for saving data.
mdict - a dictionary containing the data.
do_compression - a boolean value that specifies whether to compress the result or not. Default False.
Example
Export the following array as variable name "vec" to a mat file:

```
from scipy import io
import numpy as np

arr = np.arange(10)

io.savemat('arr.mat', {"vec": arr})
```

Note: The example above saves a file name "arr.mat" on your computer.

To open the file, check out the "Import Data from Matlab Format" example below:

Import Data from Matlab Format
The loadmat() function allows us to import data from a Matlab file.

The function takes one required parameter:

filename - the file name of the saved data.

It will return a structured array whose keys are the variable names, and the corresponding values are the variable values.

Example
Import the array from following mat file.:

```
from scipy import io
import numpy as np

arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9,])

# Export:
io.savemat('arr.mat', {"vec": arr})

# Import:
mydata = io.loadmat('arr.mat')

print(mydata)
```

Result:

```
 {
   '__header__': b'MATLAB 5.0 MAT-file Platform: nt, Created on: Tue Sep 22
13:12:32 2020',
   '__version__': '1.0',
   '__globals__': [],
   'vec': array([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])
 }
```

Use the variable name "vec" to display only the array from the matlab data:

Example

```
...

print(mydata['vec'])
```

Result:

```
 [[0 1 2 3 4 5 6 7 8 9]]
```

Note: We can see that the array originally was 1D, but on extraction it has increased one dimension.

In order to resolve this we can pass an additional argument squeeze_me=True:

Example

```
# Import:
mydata = io.loadmat('arr.mat', squeeze_me=True)

print(mydata['vec'])
```

Result:

```
 [0 1 2 3 4 5 6 7 8 9]
```

SciPy Interpolation
What is Interpolation?
Interpolation is a method for generating points between given points.

For example: for points 1 and 2, we may interpolate and find points 1.33 and 1.66.

Interpolation has many usage, in Machine Learning we often deal with missing data in a dataset, interpolation is often used to substitute those values.

This method of filling values is called imputation.

Apart from imputation, interpolation is often used where we need to smooth the discrete points in a dataset.

How to Implement it in SciPy?
SciPy provides us with a module called scipy.interpolate which has many functions to deal with interpolation:

1D Interpolation
The function interp1d() is used to interpolate a distribution with 1 variable.

It takes x and y points and returns a callable function that can be called with new x and returns corresponding y.

Example
For given xs and ys interpolate values from 2.1, 2.2... to 2.9:

```
from scipy.interpolate import interp1d
import numpy as np

xs = np.arange(10)
ys = 2*xs + 1

interp_func = interp1d(xs, ys)

newarr = interp_func(np.arange(2.1, 3, 0.1))

print(newarr)
```

Result:

```
  [5.2  5.4  5.6  5.8  6.   6.2  6.4  6.6  6.8]
```

Note: that new xs should be in same range as of the old xs, meaning that we can't call interp_func() with values higher than 10, or less than 0.

Spline Interpolation
In 1D interpolation the points are fitted for a single curve whereas in Spline interpolation the points are fitted against a piecewise function defined with polynomials called splines.

The UnivariateSpline() function takes xs and ys and produce a callable funciton that can be called with new xs.

Piecewise function: A function that has different definition for different ranges.

Example
Find univariate spline interpolation for 2.1, 2.2... 2.9 for the following non linear points:

```
from scipy.interpolate import UnivariateSpline
import numpy as np

xs = np.arange(10)
ys = xs**2 + np.sin(xs) + 1

interp_func = UnivariateSpline(xs, ys)

newarr = interp_func(np.arange(2.1, 3, 0.1))

print(newarr)
```

Result:

```
  [5.62826474 6.03987348 6.47131994 6.92265019 7.3939103  7.88514634
```

```
   8.39640439 8.92773053 9.47917082]
```

Interpolation with Radial Basis Function
Radial basis function is a function that is defined corresponding to a fixed reference point.

The Rbf() function also takes xs and ys as arguments and produces a callable function that can be called with new xs.

Example
Interpolate following xs and ys using rbf and find values for 2.1, 2.2 ... 2.9:

```
from scipy.interpolate import Rbf
import numpy as np

xs = np.arange(10)
ys = xs**2 + np.sin(xs) + 1

interp_func = Rbf(xs, ys)

newarr = interp_func(np.arange(2.1, 3, 0.1))

print(newarr)
```

Result:

```
  [6.25748981  6.62190817  7.00310702  7.40121814  7.8161443   8.24773402
   8.69590519  9.16070828  9.64233874]
```

SciPy Statistical Significance Tests
What is Statistical Significance Test?
In statistics, statistical significance means that the result that was produced has a reason behind it, it was not produced randomly, or by chance.

SciPy provides us with a module called scipy.stats, which has functions for performing statistical significance tests.

Here are some techniques and keywords that are important when performing such tests:

Hypothesis in Statistics
Hypothesis is an assumption about a parameter in population.

Null Hypothesis
It assumes that the observation is not statistically significant.

Alternate Hypothesis
It assumes that the observations are due to some reason.

It's alternate to Null Hypothesis.

Example:

For an assessment of a student we would take:

"student is worse than average" - as a null hypothesis, and:

"student is better than average" - as an alternate hypothesis.

One tailed test
When our hypothesis is testing for one side of the value only, it is called "one tailed test".

Example:

For the null hypothesis:

"the mean is equal to k", we can have alternate hypothesis:

"the mean is less than k", or:

"the mean is greater than k"

Two tailed test
When our hypothesis is testing for both side of the values.

Example:

For the null hypothesis:

"the mean is equal to k", we can have alternate hypothesis:

"the mean is not equal to k"

In this case the mean is less than, or greater than k, and both sides are to be checked.

Alpha value
Alpha value is the level of significance.

Example:

How close to extremes the data must be for null hypothesis to be rejected.

It is usually taken as 0.01, 0.05, or 0.1.

P value
P value tells how close to extreme the data actually is.

P value and alpha values are compared to establish the statistical significance.

If p value <= alpha we reject the null hypothesis and say that the data is statistically significant. otherwise we accept the null hypothesis.

T-Test
T-tests are used to determine if there is significant deference between means of two variables and lets us know if they belong to the same distribution.

It is a two tailed test.

The function ttest_ind() takes two samples of same size and produces a tuple of t-statistic and p-value.

Example
Find if the given values v1 and v2 are from same distribution:

```
import numpy as np
from scipy.stats import ttest_ind

v1 = np.random.normal(size=100)
v2 = np.random.normal(size=100)

res = ttest_ind(v1, v2)

print(res)
```

Result:

```
  Ttest_indResult(statistic=0.40833510339674095, pvalue=0.68346891833752133)
```

If you want to return only the p-value, use the pvalue property:

Example

```
...
res = ttest_ind(v1, v2).pvalue

print(res)
```

Result:

```
  0.68346891833752133
```

KS-Test

KS test is used to check if given values follow a distribution.

The function takes the value to be tested, and the CDF as two parameters.

A CDF can be either a string or a callable function that returns the probability.

It can be used as a one tailed or two tailed test.

By default it is two tailed. We can pass parameter alternative as a string of one of two-sided, less, or greater.

Example

Find if the given value follows the normal distribution:

```
import numpy as np
from scipy.stats import kstest

v = np.random.normal(size=100)

res = kstest(v, 'norm')

print(res)
```

Result:

```
  KstestResult(statistic=0.047798701221956841, pvalue=0.97630967161777515)
```

Statistical Description of Data

In order to see a summary of values in an array, we can use the describe() function.

It returns the following description:

number of observations (nobs)
minimum and maximum values = minmax
mean
variance
skewness
kurtosis

Example

Show statistical description of the values in an array:

```
import numpy as np
from scipy.stats import describe

v = np.random.normal(size=100)
res = describe(v)
```

```
print(res)
```

Result:

```
  DescribeResult(
    nobs=100,
    minmax=(-2.0991855456740121, 2.1304142707414964),
    mean=0.11503747689121079,
    variance=0.99418092655064605,
    skewness=0.013953400984243667,
    kurtosis=-0.671060517912661
  )
```

Normality Tests (Skewness and Kurtosis)
Normality tests are based on the skewness and kurtosis.

The normaltest() function returns p value for the null hypothesis:

"x comes from a normal distribution".

Skewness:
A measure of symmetry in data.

For normal distributions it is 0.

If it is negative, it means the data is skewed left.

If it is positive it means the data is skewed right.

Kurtosis:
A measure of whether the data is heavy or lightly tailed to a normal distribution.

Positive kurtosis means heavy tailed.

Negative kurtosis means lightly tailed.

Example
Find skewness and kurtosis of values in an array:

```
import numpy as np
from scipy.stats import skew, kurtosis

v = np.random.normal(size=100)

print(skew(v))
print(kurtosis(v))
```

Result:

```
  0.11168446328610283
  -0.1879320563260931
```

Example
Find if the data comes from a normal distribution:

```
import numpy as np
from scipy.stats import normaltest

v = np.random.normal(size=100)

print(normaltest(v))
```

Result:

```
  NormaltestResult(statistic=4.4783745697002848, pvalue=0.10654505998635538)
```

Django Tutorial
[+:
Django is a back-end server side web framework.

Django is free, open source and written in Python.

Django makes it easier to build web pages using Python.

Learning by Doing
In this tutorial you get a step by step guide on how to install and create a Django project. You will learn how to create a project where you can add, read, update or delete data.

You will learn how to make HTML Templates and use Django Template Tags to insert data within a HTML document.

You will learn how to work with QuerySets to extract, filter, and sort data from the database.

You will also learn how to set up a PostgreSQL database and how to deploy your Django project to the world.

QuerySets
QuerySet
Get Data
Filter
Order By

Static Files
Add Static
WhiteNoise
Collect
Global Static
Add to Poject

PostgreSQL
Intro
AWS Account
RDS
Connect
Add Members

Deploy
Elastic Beanstal
Requirements
django.config
Create .zip
Deploy
Update

Django Exercises
Test Yourself With Exercises
Exercise:
Insert the missing parts to write a Django variable in a template:

<h1>Hello firstname , how are you?</h1>

Start the Exercise

Django Quiz
Learn by taking a quiz! The quiz will give you a signal of how much you know about Django.

Learning by Examples
In the tutorial we will use examples to better explain the various concepts.

ExampleGet your own Django Server
```
<ul>
  {% for x in mymembers %}
    <li>{{ x.firstname }}</li>
  {% endfor %}
</ul>
```
My Learning
Track your progress with the free "My Learning" program here at W3Schools.

Log in to your account, and start earning points!

This is an optional feature. You can study W3Schools without using My Learning.

Learning by References
You will also find references over the most common Django features:

Template Tags QuerySet Filters Field Lookups

Django Introduction
What is Django?
Django is a Python framework that makes it easier to create web sites using Python.

Django takes care of the difficult stuff so that you can concentrate on building your web applications.

Django emphasizes reusability of components, also referred to as DRY (Don't Repeat Yourself), and comes with ready-to-use features like login system, database connection and CRUD operations (Create Read Update Delete).

Django is especially helpful for database driven websites.

How does Django Work?
Django follows the MVT design pattern (Model View Template).

Model - The data you want to present, usually data from a database.
View - A request handler that returns the relevant template and content - based on the request from the user.
Template - A text file (like an HTML file) containing the layout of the web page, with logic on how to display the data.
Model
The model provides data from the database.

In Django, the data is delivered as an Object Relational Mapping (ORM), which is a technique designed to make it easier to work with databases.

The most common way to extract data from a database is SQL. One problem with SQL is that you have to have a pretty good understanding of the database structure to be able to work with it.

Django, with ORM, makes it easier to communicate with the database, without having to write complex SQL statements.

The models are usually located in a file called models.py.

View
A view is a function or method that takes http requests as arguments, imports the relevant model(s), and finds out what data to send to the template, and returns the final result.

The views are usually located in a file called views.py.

Template
A template is a file where you describe how the result should be represented.

Templates are often .html files, with HTML code describing the layout of a web page, but it can also be in other file formats to present other results, but we will concentrate on .html files.

Django uses standard HTML to describe the layout, but uses Django tags to add logic:

<h1>My Homepage</h1>

<p>My name is {{ firstname }}.</p>
The templates of an application is located in a folder named templates.

URLs

Django also provides a way to navigate around the different pages in a website.

When a user requests a URL, Django decides which view it will send it to.

This is done in a file called urls.py.

So, What is Going On?
When you have installed Django and created your first Django web application, and the browser requests the URL, this is basically what happens:

Django receives the URL, checks the urls.py file, and calls the view that matches the URL.
The view, located in views.py, checks for relevant models.
The models are imported from the models.py file.
The view then sends the data to a specified template in the template folder.
The template contains HTML and Django tags, and with the data it returns finished HTML content back to the browser.
Django can do a lot more than this, but this is basically what you will learn in this tutorial, and are the basic steps in a simple web application made with Django.

Django History
Django was invented by Lawrence Journal-World in 2003, to meet the short deadlines in the newspaper and at the same time meeting the demands of experienced web developers.

Initial release to the public was in July 2005.

Latest version of Django is 4.0.3 (March 2022).

Django Getting Started
To install Django, you must have Python installed, and a package manager like PIP.

PIP is included in Python from version 3.4.

Django Requires Python
To check if your system has Python installed, run this command in the command prompt:

```
python --version
```
If Python is installed, you will get a result with the version number, like this

```
Python 3.9.2
```
If you find that you do not have Python installed on your computer, then you can download it for free from the following website: https://www.python.org/

PIP
To install Django, you must use a package manager like PIP, which is included in Python from version 3.4.

To check if your system has PIP installed, run this command in the command prompt:

```
pip --version
```
If PIP is installed, you will get a result with the version number.

For me, on a windows machine, the result looks like this:

```
pip 20.2.3 from c:\python39\lib\site-packages\pip (python 3.9)
```
If you do not have PIP installed, you can download and install it from this page: https://pypi.org/project/pip/

Virtual Environment
It is suggested to have a dedicated virtual environment for each Django project, and in the next chapter you will learn how to create a virtual environment, and then install Django in it.

Django - Create Virtual Environment
Virtual Environment
It is suggested to have a dedicated virtual environment for each Django project, and one way to manage a virtual environment is venv, which is included in Python.

The name of the virtual environment is your choice, in this tutorial we will call it myworld.

Type the following in the command prompt, remember to navigate to where you want to create your project:

Windows:

```
py -m venv myworld
```
Unix/MacOS:

```
python -m venv myworld
```
This will set up a virtual environment, and create a folder named "myworld" with subfolders and files, like this:

```
myworld
  Include
  Lib
  Scripts
  pyvenv.cfg
```
Then you have to activate the environment, by typing this command:

Windows:

```
myworld\Scripts\activate.bat
```
Unix/MacOS:

```
source myworld/bin/activate
```
Once the environment is activated, you will see this result in the command prompt:

Windows:

```
(myworld) C:\Users\Your Name>
```
Unix/MacOS:

```
(myworld) ... $
```
Note: You must activate the virtual environment every time you open the command prompt to work on your project.

Install Django
In the next chapter you will finally learn how to install Django!

Install Django
Install Django
Now, that we have created a virtual environment, we are ready to install Django.

Note: Remember to install Django while you are in the virtual environment!

```
Django is installed using pip, with this command:

Windows:

(myworld) C:\Users\Your Name>py -m pip install Django
Unix/MacOS:

(myworld) ... $ python -m pip install Django
Which will give a result that looks like this (at least on my Windows machine):

Collecting Django
  Downloading Django-4.0.3-py3-none-any.whl (8.0 MB)
      |ââââââââââââââââââââââââââââââââ| 8.0 MB 2.2 MB/s
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.2-py3-none-any.whl (42 kB)
Collecting asgiref<4,>=3.4.1
  Downloading asgiref-3.5.0-py3-none-any.whl (22 kB)
Collecting tzdata; sys_platform == "win32"
  Downloading tzdata-2021.5-py2.py3-none-any.whl (339 kB)
      |ââââââââââââââââââââââââââââââââ| 339 kB 6.4 MB/s
Installing collected packages: sqlparse, asgiref, tzdata, Django
Successfully installed Django-4.0.3 asgiref-3.5.0 sqlparse-0.4.2 tzdata-2021.5
WARNING: You are using pip version 20.2.3; however, version 22.3 is available.
You should consider upgrading via the 'C:\Users\Your Name\myworld\Scripts\
python.exe -m pip install --upgrade pip' command.
That's it! Now you have installed Django in your new project, running in a
virtual environment!

Windows, Mac, or Unix?
You can run this project on either one. There are some small differences, like
when writing commands in the command prompt, Windows uses py as the first word
in the command line, while Unix and MacOS use python:

Windows:

py --version
Unix/MacOS:

python --version
In the rest of this tutorial, we will be using the Windows command.

Check Django Version
You can check if Django is installed by asking for its version number like this:

(myworld) C:\Users\Your Name>django-admin --version
If Django is installed, you will get a result with the version number:

4.1.2
What's Next?
Now you are ready to create a Django project in a virtual environment on your
computer.

In the next chapters of this tutorial we will create a Django project and look
at the various features of Django and hopefully make you a Django developer.

-----

Django Create Project
My First Project
Once you have come up with a suitable name for your Django project, like mine:
my_tennis_club, navigate to where in the file system you want to store the code
(in the virtual environment), I will navigate to the myworld folder, and run
this command in the command prompt:
```

```
django-admin startproject my_tennis_club
```

Django creates a my_tennis_club folder on my computer, with this content:

```
my_tennis_club
    manage.py
    my_tennis_club/
        __init__.py
        asgi.py
        settings.py
        urls.py
        wsgi.py
```

These are all files and folders with a specific meaning, you will learn about some of them later in this tutorial, but for now, it is more important to know that this is the location of your project, and that you can start building applications in it.

Run the Django Project

Now that you have a Django project, you can run it, and see what it looks like in a browser.

Navigate to the /my_tennis_club folder and execute this command in the command prompt:

```
py manage.py runserver
```

Which will produce this result:

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you
apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
October 27, 2022 - 13:03:14
Django version 4.1.2, using settings 'my_tennis_club.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Open a new browser window and type 127.0.0.1:8000 in the address bar.

The result:

What's Next?

We have a Django project!

The next step is to make an app in your project.

You cannot have a web page created with Django without an app.

Django Create App

What is an App?

An app is a web application that has a specific meaning in your project, like a home page, a contact form, or a members database.

In this tutorial we will create an app that allows us to list and register members in a database.

But first, let's just create a simple Django app that displays "Hello World!".

Create App

I will name my app members.

Start by navigating to the selected location where you want to store the app, in my case the my_tennis_club folder, and run the command below.

If the server is still running, and you are not able to write commands, press [CTRL] [BREAK], or [CTRL] [C] to stop the server and you should be back in the virtual environment.

```
py manage.py startapp members
```

Django creates a folder named members in my project, with this content:

```
my_tennis_club
    manage.py
    my_tennis_club/
    members/
        migrations/
            __init__.py
        __init__.py
        admin.py
        apps.py
        models.py
        tests.py
        views.py
```

These are all files and folders with a specific meaning. You will learn about most of them later in this tutorial.

First, take a look at the file called views.py.

This is where we gather the information we need to send back a proper response.

You will learn more about views in the next chapter.

Django Views

Views

Django views are Python functions that takes http requests and returns http response, like HTML documents.

A web page that uses Django is full of views with different tasks and missions.

Views are usually put in a file called views.py located on your app's folder.

There is a views.py in your members folder that looks like this:

my_tennis_club/members/views.py:

```
from django.shortcuts import render

# Create your views here.
```

Find it and open it, and replace the content with this:

my_tennis_club/members/views.py:

```
from django.shortcuts import render
from django.http import HttpResponse

def members(request):
    return HttpResponse("Hello world!")
```

Note: The name of the view does not have to be the same as the application.

I call it members because I think it fits well in this context.

This is a simple example on how to send a response back to the browser.

But how can we execute the view? Well, we must call the view via a URL.

You will learn about URLs in the next chapter.

Django URLs
URLs
Create a file named urls.py in the same folder as the views.py file, and type this code in it:

my_tennis_club/members/urls.py:

```
from django.urls import path
from . import views

urlpatterns = [
    path('members/', views.members, name='members'),
]
```
The urls.py file you just created is specific for the members application. We have to do some routing in the root directory my_tennis_club as well. This may seem complicated, but for now, just follow the instructions below.

There is a file called urls.py on the my_tennis_club folder, open that file and add the include module in the import statement, and also add a path() function in the urlpatterns[] list, with arguments that will route users that comes in via 127.0.0.1:8000/.

Then your file will look like this:

my_tennis_club/my_tennis_club/urls.py:

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('', include('members.urls')),
    path('admin/', admin.site.urls),
]
```

If the server is not running, navigate to the /my_tennis_club folder and execute this command in the command prompt:

```
py manage.py runserver
```
In the browser window, type 127.0.0.1:8000/members/ in the address bar.

Django Templates
Templates
In the Django Intro page, we learned that the result should be in HTML, and it should be created in a template, so let's do that.

Create a templates folder inside the members folder, and create a HTML file named myfirst.html.

The file structure should be like this:

```
my_tennis_club
    manage.py
    my_tennis_club/
```

```
    members/
        templates/
            myfirst.html
```

Open the HTML file and insert the following:

my_tennis_club/members/templates/myfirst.html:

```
<!DOCTYPE html>
<html>
<body>

<h1>Hello World!</h1>
<p>Welcome to my first Django project!</p>

</body>
</html>
```

Modify the View

Open the views.py file and replace the members view with this:

my_tennis_club/members/views.py:

```
from django.http import HttpResponse
from django.template import loader

def members(request):
  template = loader.get_template('myfirst.html')
  return HttpResponse(template.render())
```

Change Settings

To be able to work with more complicated stuff than "Hello World!", We have to tell Django that a new app is created.

This is done in the settings.py file in the my_tennis_club folder.

Look up the INSTALLED_APPS[] list and add the members app like this:

my_tennis_club/my_tennis_club/settings.py:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'members'
]
```

Then run this command:

```
py manage.py migrate
```

Which will produce this output:

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
```

```
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK

(myworld) C:\Users\Your Name\myworld\my_tennis_club>
```

Start the server by navigating to the /my_tennis_club folder and execute this command:

```
py manage.py runserver
```

In the browser window, type 127.0.0.1:8000/members/ in the address bar.

The result should look like this:

Django Models
A Django model is a table in your database.

Django Models
Up until now in this tutorial, output has been static data from Python or HTML templates.

Now we will see how Django allows us to work with data, without having to change or upload files in the prosess.

In Django, data is created in objects, called Models, and is actually tables in a database.

Create Table (Model)
To create a model, navigate to the models.py file in the /members/ folder.

Open it, and add a Member table by creating a Member class, and describe the table fields in it:

my_tennis_club/members/models.py:

```
from django.db import models

class Member(models.Model):
  firstname = models.CharField(max_length=255)
  lastname = models.CharField(max_length=255)
```

The first field, firstname, is a Text field, and will contain the first name of the members.

The second field, lastname, is also a Text field, with the member's last name.

Both firstname and lastname is set up to have a maximum of 255 characters.

SQLite Database
When we created the Django project, we got an empty SQLite database.

It was created in the my_tennis_club root folder, and has the filename db.sqlite3.

By default, all Models created in the Django project will be created as tables in this database.

Migrate
Now when we have described a Model in the models.py file, we must run a command

to actually create the table in the database.

Navigate to the /my_tennis_club/ folder and run this command:

```
py manage.py makemigrations members
```

Which will result in this output:

```
Migrations for 'members':
  members\migrations\0001_initial.py
    - Create model Member

(myworld) C:\Users\Your Name\myworld\my_tennis_club>
```

Django creates a file describing the changes and stores the file in the /migrations/ folder:

my_tennis_club/members/migrations/0001_initial.py:

```
# Generated by Django 4.1.2 on 2022-10-27 11:14

from django.db import migrations, models


class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Member',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True,
serialize=False, verbose_name='ID')),
                ('firstname', models.CharField(max_length=255)),
                ('lastname', models.CharField(max_length=255)),
            ],
        ),
    ]
```

Note that Django inserts an id field for your tables, which is an auto increment number (first record gets the value 1, the second record 2 etc.), this is the default behavior of Django, you can override it by describing your own id field.

The table is not created yet, you will have to run one more command, then Django will create and execute an SQL statement, based on the content of the new file in the /migrations/ folder.

Run the migrate command:

```
py manage.py migrate
```

Which will result in this output:

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, members, sessions
Running migrations:
  Applying members.0001_initial... OK

(myworld) C:\Users\Your Name\myworld\my_tennis_club>
```

Now you have a Member table in you database!

View SQL

As a side-note: you can view the SQL statement that were executed from the migration above. All you have to do is to run this command, with the migration number:

```
py manage.py sqlmigrate members 0001
```

Which will result in this output:

```
BEGIN;
--
-- Create model Member
--
CREATE TABLE "members_member" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
"firstname" varchar(255) NOT NULL, "lastname" varchar(255) NOT NULL); COMMIT;
```

Django Insert Data
Add Records
The Members table created in the previous chapter is empty.

We will use the Python interpreter (Python shell) to add some members to it.

To open a Python shell, type this command:

```
py manage.py shell
```

Now we are in the shell, the result should be something like this:

```
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

At the bottom, after the three >>> write the following:

```
>>> from members.models import Member
```

Hit [enter] and write this to look at the empty Member table:

```
>>> Member.objects.all()
```

This should give you an empty QuerySet object, like this:

```
<QuerySet []>
```

A QuerySet is a collection of data from a database.

Read more about QuerySets in the Django QuerySet chapter.

Add a record to the table, by executing these two lines:

```
>>> member = Member(firstname='Emil', lastname='Refsnes')
>>> member.save()
```

Execute this command to see if the Member table got a member:

```
>>> Member.objects.all().values()
```

Hopefully, the result will look like this:

```
<QuerySet [{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'}]>
```

Add Multiple Records
You can add multiple records by making a list of Member objects, and
execute .save() on each entry:

```
>>> member1 = Member(firstname='Tobias', lastname='Refsnes')
>>> member2 = Member(firstname='Linus', lastname='Refsnes')
>>> member3 = Member(firstname='Lene', lastname='Refsnes')
>>> member4 = Member(firstname='Stale', lastname='Refsnes')
>>> member5 = Member(firstname='Jane', lastname='Doe')
>>> members_list = [member1, member2, member3, member4, member5]
>>> for x in members_list:
```

```
>>>   x.save()
```

Now there are 6 members in the Member table:

```
>>> Member.objects.all().values()
<QuerySet [{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'},
{'id': 2, 'firstname': 'Tobias', 'lastname': 'Refsnes'},
{'id': 3, 'firstname': 'Linus', 'lastname': 'Refsnes'},
{'id': 4, 'firstname': 'Lene', 'lastname': 'Refsnes'},
{'id': 5, 'firstname': 'Stale', 'lastname': 'Refsnes'},
{'id': 6, 'firstname': 'Jane', 'lastname': 'Doe'}]>
```

Django Update Data

Update Records

To update records that are already in the database, we first have to get the record we want to update:

```
>>> from members.models import Member
>>> x = Member.objects.all()[4]
```

x will now represent the member at index 4, which is "Stale Refsnes", but to make sure, let us see if that is correct:

```
>>> x.firstname
```

This should give you this result:

```
'Stale'
```

Now we can change the values of this record:

```
>>> x.firstname = "Stalikken"
>>> x.save()
```

Execute this command to see if the Member table got updated:

```
>>> Member.objects.all().values()
```

Hopefully, the result will look like this:

```
<QuerySet [{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'},
{'id': 2, 'firstname': 'Tobias', 'lastname': 'Refsnes'},
{'id': 3, 'firstname': 'Linus', 'lastname': 'Refsnes'},
{'id': 4, 'firstname': 'Lene', 'lastname': 'Refsnes'},
{'id': 5, 'firstname': 'Stalikken', 'lastname': 'Refsnes'},
{'id': 6, 'firstname': 'Jane', 'lastname': 'Doe'}]>
```

Django Delete Data

Delete Records

To delete a record in a table, start by getting the record you want to delete:

```
>>> from members.models import Member
>>> x = Member.objects.all()[5]
```

x will now represent the member at index 5, which is "Jane Doe", but to make sure, let us see if that is correct:

```
>>> x.firstname
```

This should give you this result:

```
'Jane'
```

Now we can delete the record:

```
>>> x.delete()
```

The result will be:

```
(1, {'members.Member': 1})
```

Which tells us how many items were deleted, and from which Model.

If we look at the Member Model, we can see that 'Jane Doe' is removed from the Model:

```
>>> Member.objects.all().values()
<QuerySet [{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'},
{'id': 2, 'firstname': 'Tobias', 'lastname': 'Refsnes'},
{'id': 3, 'firstname': 'Linus', 'lastname': 'Refsnes'},
{'id': 4, 'firstname': 'Lene', 'lastname': 'Refsnes'},
{'id': 5, 'firstname': 'Stalikken', 'lastname': 'Refsnes'}]>
```

Django Update Model
Add Fields in the Model
To add a field to a table after it is created, open the models.py file, and make your changes:

my_tennis_club/members/models.py:

```
from django.db import models

class Member(models.Model):
  firstname = models.CharField(max_length=255)
  lastname = models.CharField(max_length=255)
  phone = models.IntegerField()
  joined_date = models.DateField()
```

As you can see, we want to add phone and joined_date to our Member Model.

This is a change in the Model's structure, and therefor we have to make a migration to tell Django that it has to update the database:

```
py manage.py makemigrations members
```

Which, in my case, will result in a prompt, because I try to add fields that are not allowed to be null, to a table that already contains records.

As you can see, Django asks if I want to provide the fields with a specific value, or if I want to stop the migration and fix it in the model:

```
py manage.py makemigrations members
You are trying to add a non-nullable field 'joined_date' to members without a default; we can't do that (the database needs something to populate existing rows).
Please select a fix:
 1) Provide a one-off default now (will be set on all existing rows with a null value for this column)
 2) Quit, and let me add a default in models.py
Select an option:
```

I will select option 2, and open the models.py file again and allow NULL values for the two new fields:

my_tennis_club/members/models.py:

```
from django.db import models

class Member(models.Model):
  firstname = models.CharField(max_length=255)
  lastname = models.CharField(max_length=255)
  phone = models.IntegerField(null=True)
  joined_date = models.DateField(null=True)
```

And make the migration once again:

```
py manage.py makemigrations members
```

Which will result in this:

```
Migrations for 'members':
  members\migrations\0002_member_joined_date_member_phone.py
    - Add field joined_date to member
    - Add field phone to member
```

Run the migrate command:

```
py manage.py migrate
```

Which will result in this output:

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, members, sessions
Running migrations:
  Applying members.0002_member_joined_date_member_phone... OK

(myworld) C:\Users\Your Name\myworld\my_tennis_club>
```

Insert Data

We can insert data to the two new fields with the same approach as we did in the Update Data chapter:

First we enter the Python Shell:

```
py manage.py shell
```

Now we are in the shell, the result should be something like this:

```
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

At the bottom, after the three >>> write the following (and hit [enter] for each line):

```
>>> from members.models import Member
>>> x = Member.objects.all()[0]
>>> x.phone = 5551234
>>> x.joined_date = '2022-01-05'
>>> x.save()
```

This will insert a phone number and a date in the Member Model, at least for the first record, the four remaining records will for now be left empty. We will deal with them later in the tutorial.

Execute this command to see if the Member table got updated:

```
>>> Member.objects.all().values()
```

The result should look like this:

```
<QuerySet [
{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes', 'phone': 5551234,
'joined_date': datetime.date(2022, 1, 5)},
{'id': 2, 'firstname': 'Tobias', 'lastname': 'Refsnes', 'phone': None,
'joined_date': None},
{'id': 3, 'firstname': 'Linus', 'lastname': 'Refsnes', 'phone': None,
'joined_date': None},
{'id': 4, 'firstname': 'Lene', 'lastname': 'Refsnes', 'phone': None,
'joined_date': None},
{'id': 5, 'firstname': 'Stalikken', 'lastname': 'Refsnes', 'phone': None,
'joined_date': None}]>
```

Django Prepare Template

Create Template

After creating Models, with the fields and data we want in them, it is time to display the data in a web page.

Start by creating an HTML file named all_members.html and place it in the /templates/ folder:

my_tennis_club/members/templates/all_members.html:

```
<!DOCTYPE html>
<html>
<body>

<h1>Members</h1>

<ul>
  {% for x in mymembers %}
    <li>{{ x.firstname }} {{ x.lastname }}</li>
  {% endfor %}
</ul>

</body>
</html>
```

Do you see the {% %} brackets inside the HTML document?

They are Django Tags, telling Django to perform some programming logic inside these brackets.

You will learn more about Django Tags in our Django Tags chapter.

Modify View

Next we need to make the model data available in the template. This is done in the view.

In the view we have to import the Member model, and send it to the template like this:

my_tennis_club/members/views.py:

```
from django.http import HttpResponse
from django.template import loader
from .models import Member

def members(request):
  mymembers = Member.objects.all().values()
  template = loader.get_template('all_members.html')
  context = {
    'mymembers': mymembers,
  }
  return HttpResponse(template.render(context, request))
```

The members view does the following:

Creates a mymembers object with all the values of the Member model.
Loads the all_members.html template.
Creates an object containing the mymembers object.
Sends the object to the template.
Outputs the HTML that is rendered by the template.

The Result

We have created an example so that you can see the result:

If you have followed all the steps on your own computer, you can see the result in your own browser:

Start the server by navigating to the /my_tennis_club/ folder and execute this command:

```
py manage.py runserver
```

In the browser window, type 127.0.0.1:8000/members/ in the address bar.

Django Add Link to Details

Details Template

The next step in our web page will be to add a Details page, where we can list more details about a specific member.

Start by creating a new template called details.html:

my_tennis_club/members/templates/details.html:

```
<!DOCTYPE html>
<html>

<body>

<h1>{{ mymember.firstname }} {{ mymember.lastname }}</h1>

<p>Phone: {{ mymember.phone }}</p>
<p>Member since: {{ mymember.joined_date }}</p>

<p>Back to <a href="/members">Members</a></p>

</body>
</html>
```

Add Link in all-members Template

The list in all_members.html should be clickable, and take you to the details page with the ID of the member you clicked on:

my_tennis_club/members/templates/all_members.html:

```
<!DOCTYPE html>
<html>
<body>

<h1>Members</h1>

<ul>
  {% for x in mymembers %}
    <li><a href="details/{{ x.id }}">{{ x.firstname }} {{ x.lastname }}</a></li>
  {% endfor %}
</ul>

</body>
</html>
```

Create new View

Then create a new view in the views.py file, that will deal with incoming requests to the /details/ url:

my_tennis_club/members/views.py:

```
from django.http import HttpResponse
from django.template import loader
from .models import Member

def members(request):
  mymembers = Member.objects.all().values()
  template = loader.get_template('all_members.html')
  context = {
    'mymembers': mymembers,
```

```
  }
  return HttpResponse(template.render(context, request))

def details(request, id):
  mymember = Member.objects.get(id=id)
  template = loader.get_template('details.html')
  context = {
    'mymember': mymember,
  }
  return HttpResponse(template.render(context, request))
```

The details view does the following:

Gets the id as an argument.
Uses the id to locate the correct record in the Member table.
loads the details.html template.
Creates an object containing the member.
Sends the object to the template.
Outputs the HTML that is rendered by the template.
Add URLs
Now we need to make sure that the /details/ url points to the correct view, with id as a parameter.

Open the urls.py file and add the details view to the urlpatterns list:

my_tennis_club/members/urls.py:

```
from django.urls import path
from . import views

urlpatterns = [
    path('members/', views.members, name='members'),
    path('members/details/<int:id>', views.details, name='details'),
]
```

If you have followed all the steps on your own computer, you can see the result in your own browser: 127.0.0.1:8000/members/.

If the server is down, you have to start it again with the runserver command:

```
py manage.py runserver
```

Django Add Master Template
The extends Tag
In the previous pages we created two templates, one for listing all members, and one for details about a member.

The templates have a set of HTML code that are the same for both templates.

Django provides a way of making a "parent template" that you can include in all pages to do the stuff that is the same in all pages.

Start by creating a template called master.html, with all the necessary HTML elements:

MasterGet your own Django Server
my_tennis_club/members/templates/master.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>{% block title %}{% endblock %}</title>
</head>
<body>

{% block content %}
{% endblock %}

</body>
</html>
```

Do you see Django block Tag inside the <title> element, and the <body> element?

They are placeholders, telling Django to replace this block with content from other sources.

Modify Templates

Now the two templates (all_members.html and details.html) can use this master.html template.

This is done by including the master template with the {% extends %} tag, and inserting a title block and a content block:

Members

my_tennis_club/members/templates/all_members.html:

```
{% extends "master.html" %}

{% block title %}
  My Tennis Club - List of all members
{% endblock %}


{% block content %}
  <h1>Members</h1>

  <ul>
    {% for x in mymembers %}
      <li><a href="details/{{ x.id }}">{{ x.firstname }} {{ x.lastname
}}</a></li>
    {% endfor %}
  </ul>
{% endblock %}
```

Details

my_tennis_club/members/templates/details.html:

```
{% extends "master.html" %}

{% block title %}
  Details about {{ mymember.firstname }} {{ mymember.lastname }}
{% endblock %}


{% block content %}
  <h1>{{ mymember.firstname }} {{ mymember.lastname }}</h1>

  <p>Phone {{ mymember.phone }}</p>
  <p>Member since: {{ mymember.joined_date }}</p>

  <p>Back to <a href="/members">Members</a></p>

{% endblock %}
```

If you have followed all the steps on your own computer, you can see the result in your own browser: 127.0.0.1:8000/members/.

If the server is down, you have to start it again with the runserver command:

```
py manage.py runserver
```

Django Add Main Index Page
Main Index Page
Our project needs a main page.

The main page will be the landing page when someone visits the root folder of the project.

Now, you get an error when visiting the root folder of your project:

127.0.0.1:8000/.

Start by creating a template called main.html:

MainGet your own Django Server
my_tennis_club/members/templates/main.html:

```
{% extends "master.html" %}

{% block title %}
  My Tennis Club
{% endblock %}


{% block content %}
  <h1>My Tennis Club</h1>

  <h3>Members</h3>

  <p>Check out all our <a href="members/">members</a></p>

{% endblock %}
```

Create new View
Then create a new view called main, that will deal with incoming requests to root of the project:

my_tennis_club/members/views.py:

```
from django.http import HttpResponse
from django.template import loader
from .models import Member

def members(request):
  mymembers = Member.objects.all().values()
  template = loader.get_template('all_members.html')
  context = {
    'mymembers': mymembers,
  }
  return HttpResponse(template.render(context, request))

def details(request, id):
  mymember = Member.objects.get(id=id)
  template = loader.get_template('details.html')
  context = {
    'mymember': mymember,
  }
```

```
  return HttpResponse(template.render(context, request))

def main(request):
  template = loader.get_template('main.html')
  return HttpResponse(template.render())
```

The main view does the following:

loads the main.html template.
Outputs the HTML that is rendered by the template.
Add URL
Now we need to make sure that the root url points to the correct view.

Open the urls.py file and add the main view to the urlpatterns list:

my_tennis_club/members/urls.py:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.main, name='main'),
    path('members/', views.members, name='members'),
    path('members/details/<int:id>', views.details, name='details'),
]
```

Add Link Back to Main
The members page is missing a link back to the main page, so let us add that in the all_members.html template, in the content block:

Example
my_tennis_club/members/templates/all_members.html:

```
{% extends "master.html" %}

{% block title %}
  My Tennis Club - List of all members
{% endblock %}


{% block content %}

  <p><a href="/">HOME</a></p>

  <h1>Members</h1>

  <ul>
    {% for x in mymembers %}
      <li><a href="details/{{ x.id }}">{{ x.firstname }} {{ x.lastname
}}</a></li>
    {% endfor %}
  </ul>
{% endblock %}
```

If you have followed all the steps on your own computer, you can see the result in your own browser: 127.0.0.1:8000/.

If the server is down, you have to start it again with the runserver command:

```
py manage.py runserver
```

Django 404 (page not found)
Page Not Found
If you try to access a page that does not exist (a 404 error), Django directs you to a built-in view that handles 404 errors.

You will learn how to customize this 404 view later in this chapter, but first, just try to request a page that does not exist.

In the browser window, type 127.0.0.1:8000/masfdfg/ in the address bar.

You will get one of two results:

1:

2:

If you got the first result, you got directed to the built-in Django 404 template.

If you got the second result, then DEBUG is set to True in your settings, and you must set it to False to get directed to the 404 template.

This is done in the settings.py file, which is located in the project folder, in our case the my_tennis_club folder, where you also have to specify the host name from where your project runs from:

ExampleGet your own Django Server
Set the debug property to False, and allow the project to run from your local host:

my_tennis_club/my_tennis_club/settings.py:

```
.
.
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = False

ALLOWED_HOSTS = ['*']
.
.
```
Important: When DEBUG = False, Django requires you to specify the hosts you will allow this Django project to run from.

In production, this should be replaced with a proper domain name:

```
ALLOWED_HOSTS = ['yourdomain.com']
```

In the browser window, type 127.0.0.1:8000/masfdfg/ in the address bar, and you will get the built-in 404 template:

Customize the 404 Template
Django will look for a file named 404.html in the templates folder, and display it when there is a 404 error.

If no such file exists, Django shows the "Not Found" that you saw in the example above.

To customize this message, all you have to do is to create a file in the templates folder and name it 404.html, and fill it with whatever you want:

my_tennis_club/members/templates/404.html:

```
<!DOCTYPE html>
<html>
<title>Wrong address</title>
<body>

<h1>Ooops!</h1>

<h2>I cannot find the file you requested!</h2>

</body>
</html>
```

In the browser window, type 127.0.0.1:8000/masfdfg/ in the address bar, and you will get the customized 404 template:

Django Add Test View

Test View

When testing different aspects of Django, it can be a good idea to have somewhere to test code without destroying the main project.

This is optional off course, but if you like to follow all steps in this tutorial, you should add a test view that is exactly like the one we create below.

Then you can follow the examples and try them out on your own computer.

Add View

Start by adding a view called "testing" in the views.py file:

my_tennis_club/members/views.py:

```
from django.http import HttpResponse
from django.template import loader
from .models import Member

def members(request):
  mymembers = Member.objects.all().values()
  template = loader.get_template('all_members.html')
  context = {
    'mymembers': mymembers,
  }
  return HttpResponse(template.render(context, request))

def details(request, id):
  mymember = Member.objects.get(id=id)
  template = loader.get_template('details.html')
  context = {
    'mymember': mymember,
  }
  return HttpResponse(template.render(context, request))

def main(request):
  template = loader.get_template('main.html')
  return HttpResponse(template.render())

def testing(request):
  template = loader.get_template('template.html')
  context = {
    'fruits': ['Apple', 'Banana', 'Cherry'],
  }
  return HttpResponse(template.render(context, request))
```

URLs
We have to make sure that incoming urls to /testing/ will be redirected to the testing view.

This is done in the urls.py file in the members folder:

my_tennis_club/members/urls.py:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.main, name='main'),
    path('members/', views.members, name='members'),
    path('members/details/<int:id>', views.details, name='details'),
    path('testing/', views.testing, name='testing'),
]
```

Test Template
We also need a template where we can play around with HTML and Django code.

You might noticed that there was a reference to a template in the testing view?

Create a template called "template.html" in the templates folder:

```
my_tennis_club
    manage.py
    my_tennis_club/
    members/
        templates/
            404.html
            all_members.html
            details.html
            main.html
            master.html
            myfirst.html
            template.html
```

Open the template.html file and insert the following:

my_tennis_club/members/templates/template.html:

```
<!DOCTYPE html>
<html>
<body>

{% for x in fruits %}
  <h1>{{ x }}</h1>
{% endfor %}

<p>In views.py you can see what the fruits variable looks like.</p>

</body>
</html>
```

If the server is not running, navigate to the /my_tennis_club folder and execute this command in the command prompt:

```
py manage.py runserver
```

In the browser window, type 127.0.0.1:8000/testing/ in the address bar.

The result should be like this:

Django Admin
Django Admin
Django Admin is a really great tool in Django, it is actually a CRUD* user interface of all your models!

*CRUD stands for Create Read Update Delete.

It is free and comes ready-to-use with Django:

Getting Started
To enter the admin user interface, start the server by navigating to the /myworld folder and execute this command:

```
py manage.py runserver
```

In the browser window, type 127.0.0.1:8000/admin/ in the address bar.

The result should look like this:

The reason why this URL goes to the Django admin log in page can be found in the urls.py file of your project:

my_tennis_club/my_tennis_club/urls.py:

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('', include('members.urls')),
    path('admin/', admin.site.urls),
]
```

The urlpatterns[] list takes requests going to admin/ and sends them to admin.site.urls, which is part of a built-in application that comes with Django, and contains a lot of functionality and user interfaces, one of them being the log-in user interface.

Django Admin - Create User
Create User
To be able to log into the admin application, we need to create a user.

This is done by typing this command in the command view:

```
py manage.py createsuperuser
```

Which will give this prompt:

```
Username:
```

Here you must enter: username, e-mail address, (you can just pick a fake e-mail address), and password:

```
Username: johndoe
Email address: johndoe@dummymail.com
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
```

```
Bypass password validation and create user anyway? [y/N]:
```
My password did not meet the criteria, but this is a test environment, and I choose to create user anyway, by enter y:

```
Bypass password validation and create user anyway? [y/N]: y
```
If you press [Enter], you should have successfully created a user:

```
Superuser created successfully.
```
Now start the server again:

```
py manage.py runserver
```
In the browser window, type 127.0.0.1:8000/admin/ in the address bar.

And fill in the form with the correct username and password:

Which should result in this user interface:

Here you can create, read, update, and delete groups and users, but where is the Members model?

Missing Model

The Members model is missing, as it should be, you have to tell Django which models that should be visible in the admin interface.

You will learn how to include the Members model in the next chapter.

Django Admin - Include Member

Include Member in the Admin Interface

To include the Member model in the admin interface, we have to tell Django that this model should be visible in the admin interface.

This is done in a file called admin.py, and is located in your app's folder, which in our case is the members folder.

Open it, and it should look like this:

my_tennis_club/members/admin.py:

```
from django.contrib import admin

# Register your models here.
```
Insert a couple of lines here to make the Member model visible in the admin page:

my_tennis_club/members/admin.py:

```
from django.contrib import admin
from .models import Member

# Register your models here.
admin.site.register(Member)
```
Now go back to the browser and you should get this result:

Click Members and see the five records we inserted earlier in this tutorial:

Change Display

In the list in the screenshot above, we see "Member object (1)", "Member object (2)" etc. which might not be the data you wanted to be displayed in the list.

It would be better to display "firstname" and "lastname" instead.

This can easily be done by changing some settings in the models.py and/or the admin.py files. You will learn more about this in the next chapter.

Django Admin - Set Fields to Display

Make the List Display More Reader-Friendly

When you display a Model as a list, Django displays each record as the string representation of the record object, which in our case is "Member object (1)", "Member object(2)" etc.:

To change this to a more reader-friendly format, we have two choices:

Change the string representation function, __str__() of the Member Model

Set the list_details property of the Member Model

Change the String Representation Function

To change the string representation, we have to define the __str__() function of the Member Model in models.py, like this:

my_tennis_club/members/models.py:

```
from django.db import models

class Member(models.Model):
  firstname = models.CharField(max_length=255)
  lastname = models.CharField(max_length=255)
  phone = models.IntegerField(null=True)
  joined_date = models.DateField(null=True)

  def __str__(self):
    return f"{self.firstname} {self.lastname}"
```

Which gives us this result:

Defining our own __str__() function is not a Django feature, it is how to change the string representation of objects in Python. Read more about Python objects in our Python object tutorial.

Set list_display

We can control the fields to display by specifying them in in a list_display property in the admin.py file.

First create a MemberAdmin() class and specify the list_display tuple, like this:

my_tennis_club/members/admin.py:

```
from django.contrib import admin
from .models import Member

# Register your models here.

class MemberAdmin(admin.ModelAdmin):
  list_display = ("firstname", "lastname", "joined_date",)

admin.site.register(Member, MemberAdmin)
```

Remember to add the MemberAdmin as an argumet in the admin.site.register(Member, MemberAdmin).

Now go back to the browser and you should get this result:

Django Admin - Update Members
Update Members
Now we are able to create, update, and delete members in our database, and we start by giving them all a date for when they became members.

Click the first member, Stalikken, to open the record for editing, and give him a joined_date:

While we are in here, let us give him a phone number as well:

Click "SAVE" and go back to the list of all members:

Repeat these steps and give all members a date and a phone number, and end up with a list like this:

Django Admin - Add Members
Add Members
To add a new member, click on the "ADD MEMBERS" button in the top right corner:

You will get an empty form where you can fill in the members fields:

Fill in the fields and click "SAVE":

Now the Members Model have 6 members:

Django Admin - Delete Members
Delete Members
To delete a new member, you can either select a member and choose the action "Delete selected members" like this:

Or you can open a member for editing, and click the red DELETE button at the bottom, like this:

Django Template Variables
Template Variables
In Django templates, you can render variables by putting them inside {{ }} brackets:

ExampleGet your own Django Server
templates/template.html:

```
<h1>Hello {{ firstname }}, how are you?</h1>
```

Create Variable in View

The variable firstname in the example above was sent to the template via a view:

views.py:

```
from django.http import HttpResponse
from django.template import loader

def testing(request):
  template = loader.get_template('template.html')
  context = {
    'firstname': 'Linus',
  }
  return HttpResponse(template.render(context, request))
```

As you can see in the view above, we create an object named context and fill it with data, and send it as the first parameter in the template.render() function.

Create Variables in Template

You can also create variables directly in the template, by using the {% with %} template tag.

The variable is available until the {% endwith %} tag appears:

Example

templates/template.html:

```
{% with firstname="Tobias" %}
<h1>Hello {{ firstname }}, how are you?</h1>
{% endwith %}
```

You will learn more about template tags in the next chapter.

Data From a Model

The example above showed a easy approach on how to create and use variables in a template.

Normally, most of the external data you want to use in a template, comes from a model.

We have created a model in the previous chapters, called Member, which we will use in many examples in the next chapters of this tutorial.

To get data from the Member model, we will have to import it in the views.py file, and extract data from it in the view:

members/views.py:

```
from django.http import HttpResponse, HttpResponseRedirect
from django.template import loader
from .models import Member

def testing(request):
  mymembers = Member.objects.all().values()
  template = loader.get_template('template.html')
  context = {
    'mymembers': mymembers,
  }
  return HttpResponse(template.render(context, request))
```

Now we can use the data in the template:

templates/template.html:

```
<ul>
  {% for x in mymembers %}
```

```
    <li>{{ x.firstname }}</li>
  {% endfor %}
</ul>
```
We use the Django template tag {% for %} to loop through the members.

You will learn more about template tags in the next chapter.

Django Template Tags
Template Tags
In Django templates, you can perform programming logic like executing if statements and for loops.

These keywords, if and for, are called "template tags" in Django.

To execute template tags, we surround them in {% %} brackets.

ExampleGet your own Django Server
templates/template.html:

```
{% if greeting == 1 %}
  <h1>Hello</h1>
{% else %}
  <h1>Bye</h1>
{% endif %}
```
Django Code
The template tags are a way of telling Django that here comes something else than plain HTML.

The template tags allows us to to do some programming on the server before sending HTML to the client.

templates/template.html:

```
<ul>
  {% for x in mymembers %}
    <li>{{ x.firstname }}</li>
  {% endfor %}
</ul>
```
In the next chapters you will learn about the most common template tags.

Tag Reference
A list of all template tags:

Tag	Description
autoescape	Specifies if autoescape mode is on or off
block	Specifies a block section
comment	Specifies a comment section
csrf_token	Protects forms from Cross Site Request Forgeries
cycle	Specifies content to use in each cycle of a loop
debug	Specifies debugging information
extends	Specifies a parent template
filter	Filters content before returning it
firstof	Returns the first not empty variable
for	Specifies a for loop
if	Specifies a if statement
ifchanged	Used in for loops. Outputs a block only if a value has changed since the last iteration
include	Specifies included content/template
load	Loads template tags from another library
lorem	Outputs random text
now	Outputs the current date/time
regroup	Sorts an object by a group

resetcycle Used in cycles. Resets the cycle
spaceless   Removes whitespace between HTML tags
templatetag Outputs a specified template tag
url   Returns the absolute URL part of a URL
verbatim   Specifies contents that should not be rendered by the template engine
widthratio Calculates a width value based on the ratio between a given value and a max value
with Specifies a variable to use in the block

Django if Tag
If Statement
An if statement evaluates a variable and executes a block of code if the value is true.

ExampleGet your own Django Server
```
{% if greeting == 1 %}
  <h1>Hello</h1>
{% endif %}
```
Elif
The elif keyword says "if the previous conditions were not true, then try this condition".

Example
```
{% if greeting == 1 %}
  <h1>Hello</h1>
{% elif greeting == 2 %}
  <h1>Welcome</h1>
{% endif %}
```
Else
The else keyword catches anything which isn't caught by the preceding conditions.

Example
```
{% if greeting == 1 %}
  <h1>Hello</h1>
{% elif greeting == 2 %}
  <h1>Welcome</h1>
{% else %}
  <h1>Goodbye</h1>
{% endif %}
```
Operators
The above examples uses the == operator, which is used to check if a variable is equal to a value, but there are many other operators you can use, or you can even drop the operator if you just want to check if a variable is not empty:

Example
```
{% if greeting %}
  <h1>Hello</h1>
{% endif %}
```
==
Is equal to.

Example
```
{% if greeting == 2 %}
  <h1>Hello</h1>
{% endif %}
```
!=
Is not equal to.

Example
```
{% if greeting != 1 %}
```

```
  <h1>Hello</h1>
{% endif %}
```

<

Is less than.

Example

```
{% if greeting < 3 %}
  <h1>Hello</h1>
{% endif %}
```

<=

Is less than, or equal to.

Example

```
{% if greeting <= 3 %}
  <h1>Hello</h1>
{% endif %}
```

>

Is greater than.

Example

```
{% if greeting > 1 %}
  <h1>Hello</h1>
{% endif %}
```

>=

Is greater than, or equal to.

Example

```
{% if greeting >= 1 %}
  <h1>Hello</h1>
{% endif %}
```

and

To check if more than one condition is true.

Example

```
{% if greeting == 1 and day == "Friday" %}
  <h1>Hello Weekend!</h1>
{% endif %}
```

or

To check if one of the conditions is true.

Example

```
{% if greeting == 1 or greeting == 5 %}
  <h1>Hello</h1>
{% endif %}
```

and/or

Combine and and or.

Example

```
{% if greeting == 1 and day == "Friday" or greeting == 5 %}
```

Parentheses are not allowed in if statements in Django, so when you combine and and or operators, it is important to know that parentheses are added for and but not for or.

Meaning that the above example is read by the interpreter like this:

```
{% if (greeting == 1 and day == "Friday") or greeting == 5 %}
```

in

To check if a certain item is present in an object.

Example

```
{% if 'Banana' in fruits %}
  <h1>Hello</h1>
{% else %}
  <h1>Goodbye</h1>
```

```
{% endif %}
```

not in

To check if a certain item is not present in an object.

Example

```
{% if 'Banana' not in fruits %}
  <h1>Hello</h1>
{% else %}
  <h1>Goodbye</h1>
{% endif %}
```

is

Check if two objects are the same.

This operator is different from the == operator, because the == operator checks the values of two objects, but the is operator checks the identity of two objects.

In the view we have two objects, x and y, with the same values:

Example

views.py:

```
from django.http import HttpResponse
from django.template import loader

def testing(request):
  template = loader.get_template('template.html')
  context = {
    'x': ['Apple', 'Banana', 'Cherry'],
    'y': ['Apple', 'Banana', 'Cherry'],
  }
  return HttpResponse(template.render(context, request))
```

The two objects have the same value, but is it the same object?

Example

```
{% if x is y %}
  <h1>YES</h1>
{% else %}
  <h1>NO</h1>
{% endif %}
```

Let us try the same example with the == operator instead:

Example

```
{% if x == y %}
  <h1>YES</h1>
{% else %}
  <h1>NO</h1>
{% endif %}
```

How can two objects be the same? Well, if you have two objects that points to the same object, then the is operator evaluates to true:

We will demonstrate this by using the {% with %} tag, which allows us to create variables in the template:

Example

```
{% with var1=x var2=x %}
  {% if var1 is var2 %}
    <h1>YES</h1>
  {% else %}
    <h1>NO</h1>
  {% endif %}
{% endwith %}
```

is not

To check if two objects are not the same.

Example
```
{% if x is not y %}
  <h1>YES</h1>
{% else %}
  <h1>NO</h1>
{% endif %}
```

Django for Tag
For Loops
A for loop is used for iterating over a sequence, like looping over items in an array, a list, or a dictionary.

ExampleGet your own Django Server
Loop through the items of a list:

```
{% for x in fruits %}
  <h1>{{ x }}</h1>
{% endfor %}
```
Example
Loop through a list of dictionaries:

```
{% for x in cars %}
  <h1>{{ x.brand }}</h1>
  <p>{{ x.model }}</p>
  <p>{{ x.year }}</p>
{% endfor %}
```
Data From a Model
Data in a model is like a table with rows and columns.

The Member model we created earlier has five rows, and each row has three columns:

id	firstname	lastname	phone	joined_date
1	Emil	Refsnes	5551234	2022-01-05
2	Tobias	Refsnes	5557777	2022-04-01
3	Linus	Refsnes	5554321	2021-12-24
4	Lene	Refsnes	5551234	2021-05-01
5	Stalikken	Refsnes	5559876	2022-09-29

When we fetch data from the model, it comes as a QuerySet object, with a similar format as the cars example above: a list with dictionaries:

```
<QuerySet [
  {
    'id': 1,
    'firstname': 'Emil',
    'lastname': 'Refsnes',
    'phone': 5551234,
    'joined_date': datetime.date(2022, 1, 5)
  },
  {
    'id': 2,
    'firstname': 'Tobias',
    'lastname': 'Refsnes'
    'phone': 5557777,
    'joined_date': datetime.date(2021, 4, 1)
  },
  {
    'id': 3,
    'firstname': 'Linus',
    'lastname': 'Refsnes'
    'phone': 5554321,
```

```
    'joined_date': datetime.date(2021, 12, 24)
  },
  {
    'id': 4,
    'firstname': 'Lene',
    'lastname': 'Refsnes'
    'phone': 5551234,
    'joined_date': datetime.date(2021, 5, 1)
  },
  {
    'id': 5,
    'firstname': 'Stalikken',
    'lastname': 'Refsnes'
    'phone': 5559876,
    'joined_date': datetime.date(2022, 9, 29)
  }
]>
```
Example
Loop through items fetched from a database:

```
{% for x in members %}
  <h1>{{ x.id }}</h1>
  <p>
    {{ x.firstname }}
    {{ x.lastname }}
  </p>
{% endfor %}
```
Reversed
The reversed keyword is used when you want to do the loop in reversed order.

Example
```
{% for x in members reversed %}
  <h1>{{ x.id }}</h1>
  <p>
    {{ x.firstname }}
    {{ x.lastname }}
  </p>
{% endfor %}
```
Empty
The empty keyword can be used if you want to do something special if the object is empty.

Example
```
<ul>
  {% for x in emptytestobject %}
    <li>{{ x.firstname }}</li>
  {% empty %}
    <li>No members</li>
  {% endfor %}
</ul>
```
The empty keyword can also be used if the object does not exist:

Example
```
<ul>
  {% for x in myobject %}
    <li>{{ x.firstname }}</li>
  {% empty %}
    <li>No members</li>
  {% endfor %}
</ul>
```
Loop Variables
Django has some variables that are available for you inside a loop:

forloop.counter

forloop.counter0
forloop.first
forloop.last
forloop.parentloop
forloop.revcounter
forloop.revcounter0
forloop.counter
The current iteration, starting at 1.

Example
```
<ul>
  {% for x in fruits %}
    <li>{{ forloop.counter }}</li>
  {% endfor %}
</ul>
```
forloop.counter0
The current iteration, starting at 0.

Example
```
<ul>
  {% for x in fruits %}
    <li>{{ forloop.counter0 }}</li>
  {% endfor %}
</ul>
```
forloop.first
Allows you to test if the loop is on its first iteration.

Example
Draw a blue background for the first iteration of the loop:

```
<ul>
  {% for x in fruits %}
    <li
      {% if forloop.first %}
        style='background-color:lightblue;'
      {% endif %}
    >{{ x }}</li>
  {% endfor %}
</ul>
```
forloop.last
Allows you to test if the loop is on its last iteration.

Example
Draw a blue background for the last iteration of the loop:

```
<ul>
  {% for x in fruits %}
    <li
      {% if forloop.last %}
        style='background-color:lightblue;'
      {% endif %}
    >{{ x }}</li>
  {% endfor %}
</ul>
```
forloop.revcounter
The current iteration if you start at the end and count backwards, ending up at 1.

Example
```
<ul>
  {% for x in fruits %}
    <li>{{ forloop.revcounter }}</li>
  {% endfor %}
</ul>
```

forloop.revcounter0
The current iteration if you start at the end and count backwards, ending up at 0.

Example
```
<ul>
  {% for x in fruits %}
    <li>{{ forloop.revcounter0 }}</li>
  {% endfor %}
</ul>
```

Django comment Tag
Comments
Comments allows you to have sections of code that should be ignored.

ExampleGet your own Django Server
```
<h1>Welcome Everyone</h1>
{% comment %}
  <h1>Welcome ladies and gentlemen</h1>
{% endcomment %}
```
Comment Description
You can add a message to your comment, to help you remember why you wrote the comment, or as message to other people reading the code.

Example
Add a description to your comment:

```
<h1>Welcome Everyone</h1>
{% comment "this was the original welcome message" %}
    <h1>Welcome ladies and gentlemen</h1>
{% endcomment %}
```
Smaller Comments
You can also use the {# ... #} tags when commenting out code, which can be easier for smaller comments:

Example
Comment out the word Everyone:

```
<h1>Welcome{# Everyone#}</h1>
```
Comment in Views
Views are written in Python, and Python comments are written with the # character:

Example
Comment out a section in the view:

```
from django.http import HttpResponse
from django.template import loader

def testing(request):
  template = loader.get_template('template.html')
  #context = {
  # 'var1': 'John',
  #}
  return HttpResponse(template.render())
```
Read more about Python Comments in out Python Comment Tutorial.

Django include Tag
Include
The include tag allows you to include a template inside the current template.

This is useful when you have a block of content that is the same for many pages.

ExampleGet your own Django Server
templates/footer.html:

```
<p>You have reached the bottom of this page, thank you for your time.</p>
```

templates/template.html:

```
<h1>Hello</h1>

<p>This page contains a footer in a template.</p>

{% include 'footer.html' %}
```

Variables in Include
You can send variables into the template by using the with keyword.

In the include file, you refer to the variables by using the {{ variablename }} syntax:

Example
templates/mymenu.html:

```
<div>HOME | {{ me }} | ABOUT | FORUM | {{ sponsor }}</div>
```

templates/template.html:

```
<!DOCTYPE html>
<html>
<body>

{% include "mymenu.html" with me="TOBIAS" sponsor="W3SCHOOLS" %}

<h1>Welcome</h1>

<p>This is my webpage</p>

</body>
</html>
```

Django QuerySet
Django QuerySet
A QuerySet is a collection of data from a database.

A QuerySet is built up as a list of objects.

QuerySets makes it easier to get the data you actually need, by allowing you to filter and order the data at an early stage.

In this tutorial we will be querying data from the Member table.

Member:

id	firstname	lastname	phone	joined_date
1	Emil	Refsnes	5551234	2022-01-05
2	Tobias	Refsnes	5557777	2022-04-01
3	Linus	Refsnes	5554321	2021-12-24
4	Lene	Refsnes	5551234	2021-05-01
5	Stalikken	Refsnes	5559876	2022-09-29

Querying Data
In views.py, we have a view for testing called testing where we will test different queries.

In the example below we use the .all() method to get all the records and fields of the Member model:

ViewGet your own Django Server
views.py:

```
from django.http import HttpResponse
from django.template import loader
from .models import Member

def testing(request):
  mydata = Member.objects.all()
  template = loader.get_template('template.html')
  context = {
    'mymembers': mydata,
  }
  return HttpResponse(template.render(context, request))
```

The object is placed in a variable called mydata, and is sent to the template via the context object as mymembers, and looks like this:

```
<QuerySet [
  <Member: Member object (1)>,
  <Member: Member object (2)>,
  <Member: Member object (3)>,
  <Member: Member object (4)>,
  <Member: Member object (5)>
]>
```

As you can see, our Member model contains 5 records, and are listed inside the QuerySet as 5 objects.

In the template you can use the mymembers object to generate content:

Template
templates/template.html:

```
<table border='1'>
  <tr>
    <th>ID</th>
    <th>Firstname</th>
    <th>Lastname</th>
  </tr>
  {% for x in mymembers %}
    <tr>
      <td>{{ x.id }}</td>
        <td>{{ x.firstname }}</td>
      <td>{{ x.lastname }}</td>
    </tr>
  {% endfor %}
</table>
```

Django QuerySet - Get Data
Get Data
There are different methods to get data from a model into a QuerySet.

The values() Method
The values() method allows you to return each object as a Python dictionary, with the names and values as key/value pairs:

ExampleGet your own Django Server
views.py:

```
from django.http import HttpResponse
from django.template import loader
from .models import Member

def testing(request):
  mydata = Member.objects.all().values()
  template = loader.get_template('template.html')
  context = {
    'mymembers': mydata,
  }
  return HttpResponse(template.render(context, request))
```
Return Specific Columns
The values_list() method allows you to return only the columns that you specify.

Example
Return only the firstname columns:

views.py:

```
from django.http import HttpResponse
from django.template import loader
from .models import Member

def testing(request):
  mydata = Member.objects.values_list('firstname')
  template = loader.get_template('template.html')
  context = {
    'mymembers': mydata,
  }
  return HttpResponse(template.render(context, request))
```
Return Specific Rows
You can filter the search to only return specific rows/records, by using the filter() method.

Example
Return only the records where firstname is 'Emil'

views.py:

```
from django.http import HttpResponse
from django.template import loader
from .models import Member

def testing(request):
  mydata = Member.objects.filter(firstname='Emil').values()
  template = loader.get_template('template.html')
  context = {
    'mymembers': mydata,
  }
  return HttpResponse(template.render(context, request))
```
You will learn more about the filter() method in the next chapter.

Django QuerySet - Filter
QuerySet Filter
The filter() method is used to filter your search, and allows you to return only the rows that matches the search term.

As we learned in the previous chapter, we can filter on field names like this:

ExampleGet your own Django Server
Return only the records where the firstname is 'Emil':

```
mydata = Member.objects.filter(firstname='Emil').values()
```
In SQL, the above statement would be written like this:

```
SELECT * FROM members WHERE firstname = 'Emil';
```
AND
The filter() method takes the arguments as **kwargs (keyword arguments), so you can filter on more than one field by separating them by a comma.

Example
Return records where lastname is "Refsnes" and id is 2:

```
mydata = Member.objects.filter(lastname='Refsnes', id=2).values()
```
In SQL, the above statement would be written like this:

```
SELECT * FROM members WHERE lastname = 'Refsnes' AND id = 2;
```
OR
To return records where firstname is Emil or firstname is Tobias (meaning: returning records that matches either query, not necessarily both) is not as easy as the AND example above.

We can use multiple filter() methods, separated by a pipe | character. The results will merge into one model.

Example
Return records where firstname is either "Emil" or Tobias":

```
mydata = Member.objects.filter(firstname='Emil').values() |
Member.objects.filter(firstname='Tobias').values()
```
Another common method is to import and use Q expressions:

Example
Return records where firstname is either "Emil" or Tobias":

```
from django.http import HttpResponse
from django.template import loader
from .models import Member
from django.db.models import Q

def testing(request):
  mydata = Member.objects.filter(Q(firstname='Emil') |
Q(firstname='Tobias')).values()
  template = loader.get_template('template.html')
  context = {
    'mymembers': mydata,
  }
  return HttpResponse(template.render(context, request))
```
In SQL, the above statement would be written like this:

```
SELECT * FROM members WHERE firstname = 'Emil' OR firstname = 'Tobias';
```
Field Lookups
Django has its own way of specifying SQL statements and WHERE clauses.

To make specific where clauses in Django, use "Field lookups".

Field lookups are keywords that represents specific SQL keywords.

Example:
Use the __startswith keyword:

```
.filter(firstname__startswith='L');
```
Is the same as the SQL statement:

```
WHERE firstname LIKE 'L%'
```
The above statement will return records where firstname starts with 'L'.

Field Lookups Syntax
All Field lookup keywords must be specified with the fieldname, followed by two(!) underscore characters, and the keyword.

In our Member model, the statement would be written like this:

Example
Return the records where firstname starts with the letter 'L':

```
mydata = Member.objects.filter(firstname__startswith='L').values()
```

Field Lookups Reference
A list of all field look up keywords:

Keyword	Description
contains	Contains the phrase
icontains	Same as contains, but case-insensitive
date	Matches a date
day	Matches a date (day of month, 1-31) (for dates)
endswith	Ends with
iendswith	Same as endswidth, but case-insensitive
exact	An exact match
iexact	Same as exact, but case-insensitive
in	Matches one of the values
isnull	Matches NULL values
gt	Greater than
gte	Greater than, or equal to
hour	Matches an hour (for datetimes)
lt	Less than
lte	Less than, or equal to
minute	Matches a minute (for datetimes)
month	Matches a month (for dates)
quarter	Matches a quarter of the year (1-4) (for dates)
range	Match between
regex	Matches a regular expression
iregex	Same as regex, but case-insensitive
second	Matches a second (for datetimes)
startswith	Starts with
istartswith	Same as startswith, but case-insensitive
time	Matches a time (for datetimes)
week	Matches a week number (1-53) (for dates)
week_day	Matches a day of week (1-7) 1 is sunday
iso_week_day	Matches a ISO 8601 day of week (1-7) 1 is monday
year	Matches a year (for dates)
iso_year	Matches an ISO 8601 year (for dates)

Django QuerySet - Order By
Order By
To sort QuerySets, Django uses the order_by() method:

ExampleGet your own Django Server
Order the result alphabetically by firstname:

```
mydata = Member.objects.all().order_by('firstname').values()
```

In SQL, the above statement would be written like this:

```
SELECT * FROM members ORDER BY firstname;
```

Descending Order
By default, the result is sorted ascending (the lowest value first), to change the direction to descending (the highest value first), use the minus sign (NOT), - in front of the field name:

Example
Order the result firstname descending:

```
mydata = Member.objects.all().order_by('-firstname').values()
```

In SQL, the above statement would be written like this:

```
SELECT * FROM members ORDER BY firstname DESC;
```

Multiple Order Bys
To order by more than one field, separate the fieldnames with a comma in the order_by() method:

Example
Order the result first by lastname ascending, then descending on id:

```
mydata = Member.objects.all().order_by('lastname', '-id').values()
```

In SQL, the above statement would be written like this:

```
SELECT * FROM members ORDER BY lastname ASC, id DESC;
```

Django - Add Static File
Create Static Folder
When building web applications, you probably want to add some static files like images or css files.

Start by creating a folder named static in your project, the same place where you created the templates folder:

The name of the folder has to be static.

```
my_tennis_club
    manage.py
    my_tennis_club/
    members/
        templates/
        static/
```

Add a CSS file in the static folder, the name is your choice, we will call it myfirst.css in this example:

```
my_tennis_club
    manage.py
    my_tennis_club/
    members/
        templates/
        static/
            myfirst.css
```

Open the CSS file and insert the following:

my_tennis_club/members/static/myfirst.css:

```
body {
  background-color: lightblue;
  font-family: verdana;
}
```

Modify the Template
Now you have a CSS file, with some CSS styling. The next step will be to include this file in a HTML template:

Open the HTML file and add the following:

```
{% load static %}
```

And:

```
<link rel="stylesheet" href="{% static 'myfirst.css' %}">
```

ExampleGet your own Django Server
my_tennis_club/members/templates/template.html:

```
{% load static %}
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="{% static 'myfirst.css' %}">
<body>

{% for x in fruits %}
  <h1>{{ x }}</h1>
{% endfor %}

</body>
</html>
```

Restart the server for the changes to take effect:

```
py manage.py runserver
```

And check out the result in your own browser: 127.0.0.1:8000/testing/.

Didn't Work?
Just testing? If you just want to play around, and not going to deploy your work, you can set DEBUG = True in the settings.py file, and the example above will work.

Plan to deploy? If you plan to deploy your work, you should set DEBUG = False in the settings.py file. The example above will fail, because Django has no built-in solution for serving static files, but there are other ways to serve static files, you will learn how in the next chapter.

Example (in development):
my_tennis_club/my_tennis_club/settings.py:

```
.
.
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
.
.
```

This will make the example work, but we want you to choose DEBUG = False, because that is the best way to learn how to work with Django.

Choose Debug = False
For the rest of this tutorial, we will run with DEBUG = False, even in development, because that is the best way to learn how to work with Django.

Example:
my_tennis_club/my_tennis_club/settings.py:

```
.
.
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = False

ALLOWED_HOSTS = ['*']

.
.
```

ALLOWED_HOSTS
When using DEBUG = False you have to specify which host name(s) are allowed to host your work. You could choose '127.0.0.1' or 'localhost' which both represents the address of your local machine.

We choose '*', which means any address are allowed to host this site. This should be change into a real domain name when you deploy your project to a public server.

Didn't Work?
That is right, the example still does not work.

You will have install a third-party library in order to handle static files.

There are many alternatives, we will show you how to use a Python library called WhiteNoise in the next chapter.

Django - Installing WhiteNoise
WhiteNoise
Django does not have a built-in solution for serving static files, at least not in production when DEBUG has to be False.

We have to use a third-party solution to accomplish this.

In this Tutorial we will use WhiteNoise, which is a Python library, built for serving static files.

Install WhiteNoise
To install WhiteNoise in your virtual environment, type the command below:

```
pip install whitenoise
```
The result should be something like this:

```
Collecting whitenoise
  Downloading whitenoise-6.2.0-py3-none-any.whl (19 kB)
Installing collected packages: whitenoise
Successfully installed whitenoise-6.2.0
WARNING: You are using pip version 20.2.3; however, version 22.3.1 is available.
You should consider upgrading via the 'c:\users\Your Name\myworld\scripts\python.exe -m pip install --upgrade pip' command.
```
Modify Settings
To make Django aware of you wanting to run WhitNoise, you have to specify it in the MIDDLEWARE list in settings.py file:

my_tennis_club/my_tennis_club/settings.py:

```
.
.
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'whitenoise.middleware.WhiteNoiseMiddleware',
].
.
```
Collect Static Files
There are one more action you have to perform before you can serve the static file from the example in the previous chapter. You have to collect all static files and put them into one specified folder. You will learn how in the next chapter.

Django - Collect Static Files
Handle Static Files
Static files in your project, like stylesheets, JavaScripts, and images, are not handled automatically by Django when DEBUG = False.

When DEBUG = True, this worked fine, all we had to do was to put them in the static folder of the application.

When DEBUG = False, static files have to be collected and put in a specified folder before we can use it.

Collect Static Files
To collect all necessary static files for your project, start by specifying a STATIC_ROOT property in the settings.py file.

This specifies a folder where you want to collect your static files.

You can call the folder whatever you like, we will call it productionfiles:

my_tennis_club/my_tennis_club/settings.py:

```
.
.

STATIC_ROOT = BASE_DIR / 'productionfiles'

STATIC_URL = 'static/'

.
.
```

You could manually create this folder and collect and put all static files of your project into this folder, but Django has a command that do this for you:

```
py manage.py collectstatic
```

Which will produce this result:

```
131 static files copied to 'C:\Users\your_name\myworld\my_tennis_club\productionfiles'.
```

131 files? Why so many? Well this is because of the admin user interface, that comes built-in with Django. We want to keep this feature in production, and it comes with a whole bunch of files including stylesheets, fonts, images, and JavaScripts.

```
my_tennis_club
    members/
    my_tennis_club/
    productionfiles/
        admin/
        myfirst.css
```

The Example Should Work
Now you have collected the static files of your project, and if you have installed WhiteNoise, the example from the Add Static Files chapter will finally work.

Start the server and see the result:

```
py manage.py runserver
```

And check out the result in your own browser: 127.0.0.1:8000/testing/.

ExampleGet your own Django Server
my_tennis_club/members/templates/template.html:

```
{% load static %}
```

```
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="{% static 'myfirst.css' %}">
<body>

{% for x in fruits %}
  <h1>{{ x }}</h1>
{% endfor %}

</body>
</html>
```

Django - Global Static Files

Add a Global CSS File

We have learned how to add a static file in the application's static folder, and how to use it in the application.

But what if other applications in your project wants to use the file?

Then we have to create a folder on the root directory and put the file(s) there.

It is not enough to create a static folder in the root directory, and Django will fix the rest. We have to tell Django where to look for these static files.

Start by creating a folder on the project's root level, this folder can be called whatever you like, I will call it mystaticfiles in this tutorial:

```
my_tennis_club
    db.sqlite3
    manage.py
    my_tennis_club/
    members/
    mystaticfiles/
```

Add a CSS file in the mystaticfiles folder, the name is your choice, we will call it myglobal.css in this example:

```
my_tennis_club
    db.sqlite3
    manage.py
    my_tennis_club/
    members/
    mystaticfiles/
        myglobal.css
```

Open the CSS file and insert the following:

my_tennis_club/mystaticfiles/myglobal.css:

```
body {
  color: violet;
}
```

Modify Settings

You will have to tell Django to also look for static files in the mystaticfiles folder in the root directory, this is done in the settings.py file:

Add a STATICFILES_DIRS list:

my_tennis_club/my_tennis_club/settings.py:

```
.
.

STATIC_ROOT = BASE_DIR / 'productionfiles'
```

```
STATIC_URL = 'static/'

#Add this in your settings.py file:
STATICFILES_DIRS = [
    BASE_DIR / 'mystaticfiles'
]
.
.
```

In the STATICFILES_DIRS list, you can list all the directories where Django should look for static files.

The BASE_DIR keyword represents the root directory of the project, and together with the / "mystaticfiles", it means the mystaticfiles folder in the root directory.

Search Order

If you have files with the same name, Django will use the first occurrence of the file.

The search starts in the directories listed in STATICFILES_DIRS, using the order you have provided. Then, if the file is not found, the search continues in the static folder of each application.

Modify the Template

Now you have a global CSS file for the entire project, which can be accessed from all your applications.

To use it in a template, use the same syntax as you did for the myfirst.css file:

Begin the template with the following:

```
{% load static %}
```

And refer to the file like this:

```
<link rel="stylesheet" href="{% static 'myglobal.css' %}">
```

ExampleGet your own Django Server

my_tennis_club/members/templates/template.html:

```
{% load static %}
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="{% static 'myglobal.css' %}">
<body>

{% for x in fruits %}
  <h1>{{ x }}</h1>
{% endfor %}

</body>
</html>
```

Didn't Work?

That is correct. You need to collect the static files once again.

Collect Static Files

Run the collectstatic command to collect the new static file:

```
py manage.py collectstatic
```

Which will produce this result:

```
You have requested to collect static files at the destination
location as specified in your settings:

    C:\Users\Your Name\myworld\my_tennis_club\productionfiles

This will overwrite existing files!
Are you sure you want to do this?

Type 'yes' to continue, or 'no' to cancel:
```

Type yes:

```
Type 'yes' to continue, or 'no' to cancel: yes
```

Which will produce this result:

```
1 static file copied to 'C:\Users\Your Name\myworld\my_tennis_club\
productionfiles', 131 unmodified.
```

The Example Should Work

Start the server, and the example will work:

```
py manage.py runserver
```

Check out the result in your own browser: 127.0.0.1:8000/testing/.

Example

my_tennis_club/members/templates/template.html:

```
{% load static %}
<!DOCTYPE html>
<html>
<link rel="stylesheet" href="{% static 'myglobal.css' %}">
<body>

{% for x in fruits %}
  <h1>{{ x }}</h1>
{% endfor %}

</body>
</html>
```

Add CSS File to the Project

The Project - My Tennis Club

If you have followed the steps in the entire Django tutorial, you will have a my_tennis_club project on your computer, with 5 members:

We want to add a stylesheet to this project, and put it in the mystaticfiles folder:

```
my_tennis_club
    manage.py
    my_tennis_club/
    members/
    mystaticfiles/
        mystyles.css
```

The name of the CSS file is your choice, we call it mystyles.css in this project.

Open the CSS file and insert the following:

my_tennis_club/mystaticfiles/mystyles.css:

```
body {
```

```
  background-color: violet;
}
```
Modify the Master Template
Now you have a css file, the next step will be to include this file in the master template:

Open the master template file and add the following:

my_tennis_club/members/templates/master.html:

```
{% load static %}
<!DOCTYPE html>
<html>
<head>
  <title>{% block title %}{% endblock %}</title>
  <link rel="stylesheet" href="{% static 'mystyles.css' %}">
</head>
<body>

{% block content %}
{% endblock %}

</body>
</html>
```

Check Settings
Make sure your settings.py file contains a STATICFILES_DIRS list with a reference to the mystaticfiles folder on the root directory, and that you have specified a STATICFILES_ROOT folder:

my_tennis_club/my_tennis_club/settings.py:

```
.
.
STATIC_ROOT = BASE_DIR / 'productionfiles'

STATIC_URL = 'static/'

#Add this in your settings.py file:
STATICFILES_DIRS = [
    BASE_DIR / 'mystaticfiles'
]
.
.
```
Collect Static Files
Every time you make a change in a static file, you must run the collectstatic command to make the changes take effect:

```
py manage.py collectstatic
```
If you have executed the command earlier in the project, Django will prompt you with a question:

```
You have requested to collect static files at the destination
location as specified in your settings:

    C:\Users\Your Name\myworld\my_tennis_club\productionfiles

This will overwrite existing files!
Are you sure you want to do this?

Type 'yes' to continue, or 'no' to cancel:
```
Type 'yes'. This will update any changes done in the static files, and give you this result:

1 static file copied to 'C:\Users\Your Name\minverden\my_tennis_club\productionfiles', 132 unmodified.

Now, if you run the project:

```
py manage.py runserver
```

It will look like this:

If you have followed all the steps on you own computer, you can see the result in your own browser:

In the browser window, type 127.0.0.1:8000/members/ in the address bar.

Spice up the Style!

In the example above we showed you how to include a stylesheet to your project.

We ended up with a purple web page, but CSS can do more than just change the background color.

We want to do more with the styles, and end up with a result like this:

First, replace the content of the mystyles.css file with this:

my_tennis_club/mystaticfiles/mystyles.css:

```
@import url('https://fonts.googleapis.com/css2?family=Source+Sans+Pro:wght@400;600&display=swap');
body {
  margin:0;
  font: 600 18px 'Source Sans Pro', sans-serif;
  letter-spacing: 0.64px;
  color: #585d74;
}
.topnav {
  background-color:#375BDC;
  color:#ffffff;
  padding:10px;
}
.topnav a:link, .topnav a:visited {
  text-decoration: none;
  color: #ffffff;
}
.topnav a:hover, .topnav a:active {
  text-decoration: underline;
}
.mycard {
  background-color: #f1f1f1;
  background-image: linear-gradient(to bottom, #375BDC, #4D70EF);
  background-size: 100% 120px;
  background-repeat: no-repeat;
  margin: 40px auto;
  width: 350px;
  border-radius: 5px;
  box-shadow: 0 5px 7px -1px rgba(51, 51, 51, 0.23);
  padding: 20px;
}
.mycard h1 {
  text-align: center;
  color:#ffffff;
  margin:20px 0 60px 0;
}
```

```
ul {
  list-style-type: none;
  padding: 0;
  margin: 0;
}
li {
  background-color: #ffffff;
  background-image: linear-gradient(to right, #375BDC, #4D70EF);
  background-size: 50px 60px;
  background-repeat: no-repeat;
  cursor: pointer;
  transition: transform .25s;
  border-radius: 5px;
  box-shadow: 0 5px 7px -1px rgba(51, 51, 51, 0.23);
  padding: 15px;
  padding-left: 70px;
  margin-top: 5px;
}
li:hover {
  transform: scale(1.1);
}
a:link, a:visited {
  color: #375BDC;
}
.main, .main h1 {
  text-align:center;
  color:#375BDC;
}
```

Modify Templates

You also have to make some changes to the templates:

Master

We want all pages to have the same top navigation, therefor we insert the top navigation into master.html:

my_tennis_club/members/templates/master.html:

```
{% load static %}
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="{% static 'mystyles.css' %}">
  <title>{% block title %}{% endblock %}</title>
</head>
<body>

<div class="topnav">
  <a href="/">HOME</a> |
  <a href="/members">MEMBERS</a>
</div>

{% block content %}
{% endblock %}

</body>
</html>
```

Members

In all_members.html we want to make som changes in the HTML code.

The members are put in a div element, and the links become list items with onclick attributes.

We also want to remove the navigation, because that is now a part of the master template.

my_tennis_club/members/templates/all_members.html:

```
{% extends "master.html" %}

{% block title %}
  My Tennis Club - List of all members
{% endblock %}


{% block content %}
  <div class="mycard">
    <h1>Members</h1>
    <ul>
      {% for x in mymembers %}
        <li onclick="window.location = 'details/{{ x.id }}'">{{ x.firstname }}
{{ x.lastname }}</li>
      {% endfor %}
    </ul>
  </div>
{% endblock %}
```

Details

In details.html we will put the member details in a div element, and remove the link back to members, because that is now a part of the navigation in the master template.

my_tennis_club/members/templates/details.html:

```
{% extends "master.html" %}

{% block title %}
  Details about {{ mymember.firstname }} {{ mymember.lastname }}
{% endblock %}

{% block content %}
  <div class="mycard">
    <h1>{{ mymember.firstname }} {{ mymember.lastname }}</h1>
    <p>Phone {{ mymember.phone }}</p>
    <p>Member since: {{ mymember.joined_date }}</p>
  </div>
{% endblock %}
```

Main

In the main.html template, we will put some of the HTML code into a div element:

my_tennis_club/members/templates/main.html:

```
{% extends "master.html" %}

{% block title %}
  My Tennis Club
{% endblock %}

{% block content %}
  <div class="main">
    <h1>My Tennis Club</h1>

    <h3>Members</h3>

    <p>Check out all our <a href="members/">members</a></p>
  </div>
{% endblock %}
```

Collect Static Files
Since we did some changes in the static mystyles.css file, we have to run the collectstatic command to make the changes take effect:

```
py manage.py collectstatic
```

Now, if you run the project:

```
py manage.py runserver
```

You can see what the result should look like:

Or, if you have followed all the steps on you own computer, you can see the result in your own browser:

In the browser window, type 127.0.0.1:8000/members/ in the address bar.

Introduction to PostgreSQL
Database Engines
Django comes with a SQLite database which is great for testing and debugging at the beginning of a project.

However, it is not very suitable for production.

Django also support these database engines:

PostgreSQL
MariaDB
MySQL
Oracle
We will take a closer look at the PostgreSQL database engine.

PostgreSQL
PostgreSQL database is an open source relational database, which should cover most demands you have when creating a database for a Django project.

It has a good reputation, it is reliable, and it perform well under most circumstances.

We will add a PostgreSQL database to our Django project.

To be able to use PostgreSQL in Django we have to install a package called psycopg2.

Install psycopg2
Type this command in the command line to install the package. Make sure you are still inn the virtual environment:

```
pip install psycopg2-binary
```

The result should be something like this:

```
Collecting psycopg2-binary
  Downloading psycopg2_binary-2.9.5-cp39-cp39-win_amd64.whl (1.2 MB)
    |âââââââââââââââââââââââââââââââââ| 1.2 MB 3.3 MB/s
Installing collected packages: psycopg2-binary
Successfully installed psycopg2-binary-2.9.5
WARNING: You are using pip version 20.2.3; however, version 22.3.1 is available.
You should consider upgrading via the 'c:\users\Your Name\myworld\scripts\
python.exe -m pip install --upgrade pip' command.
```

The psycopg2 package is a driver that is necessary for PostgreSQL to work in Python.

We also need a server where we can host the database.

In this tutorial we have chosen the Amazon Web Services (AWS) platform, you will learn more about that in the next chapter.

Create AWS Account
Why AWS?
There are many providers out there that can host Django projects and PostgreSQL databases.

In this tutorial we will use the Amazon Web Services (AWS) platform, mainly because they offer a free solution that can host both Django projects and PostgreSQL databases. All you need is an AWS account.

Note: you can choose whatever server provider you like, they will most likely all give you a satisfying result, but they will have some provider-specific settings that you should be aware of when following this tutorial.

AWS
Go to aws.amazon.com, and create an account:

Sign In
Once you have created an AWS account, it is time to sign in for the first time:

AWS Console
If this is your first time you sign into your AWS account, you will be directed to the AWS Console Home page:

Add the RDS Service
Once you have an AWS account, you can start creating a database.

We will use a database service at AWS, called RDS.

In the search field, search for "RDS", and click to start the service:

Once the service has started, you should see something like this:

In the next chapter we will create the PostgreSQL database.

Create PostgreSQL Database
Create Database
Inside the RDS service, create a database, either by navigating to the Database section, or just click the "Create database" button:

Settings
Once you have started creating a database, you will be given some choices for the type and settings of your database.

To add a PostgreSQL database to your Django project, choose the following options:

Standard creation method:

PostgreSQL engine method:

Free Tier Template:

Name of database, username, and password
You can choose any name, username, and password:

Keep the default instance configuration:

Check off the storage autoscaling:
It can be a good thing to enable storage autoscaling, but for this tutorial it is not necessary.

Grant public access, and create a new security group:
Give the security group a name, we will call it "w3-django":

Keep default db authentications:

Keep default monitoring:

Click Create database:

This will take a few minutes, but when it is finished, you will have a new PostgreSQL database, almost ready to run on your Django project!

In the next chapter you will learn how to connect your project to the database.

Connect to Database
Modify Settings
To make Django able to connect to your database, you have to specify it in the DATABASES tuple in the settings.py file.

Before, it looked like this:

SQLiteGet your own Django Server
my_tennis_club/my_tennis_club/settings.py:

```
.
.
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

.
Now, you should change it to look like this:

PostgreSQL
my_tennis_club/my_tennis_club/settings.py:

```
.
.
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'postgres',
        'USER': 'masteruser',
        'PASSWORD': '12345678',
        'HOST': 'w3-django-project.cdxmgq9zqqlr.us-east-1.rds.amazonaws.com',
        'PORT': '5432'
    }
}
.
```

Note: The values will be different for your project.

Engine?
As you can see in the settings.py file, we insert postgresql instead of sqlite.

Name?
The database does not have a name, but you have to assign one in order to access the database.

If no name is given, the provider accepts 'postgres' as the name of the database.

Username and Password?
Insert the username and password that you specified when you created the database.

Host? Port?
As you can see in the settings.py file, we insert postgresql instead of sqlite, and insert the username and password that we specified when we created the database.

The HOST and PORT can be found under the "Connectivity & security" section in the RDS instance. They are described as "Endpoint" and "Port":

Which for my project is this:

```
'HOST': 'w3-django-project.cdxmgq9zqqlr.us-east-1.rds.amazonaws.com'
'PORT': '5432'
```

Migrate
Once we have done the changes in settings.py, we must run a migration in our virtual environment, before the changes will take place:

```
py manage.py migrate
```

Which will give you this result:

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, members, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
```

```
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying members.0001_initial... OK
  Applying members.0002_members_delete_member... OK
  Applying members.0003_rename_members_member... OK
  Applying sessions.0001_initial... OK
```

Now, if you run the project:

```
py manage.py runserver
```

And view it in your browser: 127.0.0.1:8000/.

You will get the home page of the project, but if you click on the "members" link, you will see that there are no members.

That is because the database is empty. In the next chapter we will fill the database with members.

PostgreSQL - Add Members

Members

The "My Tennis Club" project has no members: 127.0.0.1:8000/.

That is because we have created a brand new database, and it is empty.

The old SQLite database contained 5 members, so let us dive into the admin interface and add the same 5 members.

But first we have to create a new superuser.

Create superuser

Since we now have a new database, we have to create the superuser once again:

This is done by typing this command in the command view:

```
py manage.py createsuperuser
```

Which will give this prompt:

```
Username:
```

Here you must enter: username, e-mail address, (you can just pick a fake e-mail address), and password:

```
Username: johndoe
Email address: johndoe@dummymail.com
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]:
```

My password did not meet the criteria, but this is a test environment, and I choose to create user anyway, by enter y:

Bypass password validation and create user anyway? [y/N]: y
If you press [Enter], you should have successfully created a user:

Superuser created successfully.
Now start the server again:

py manage.py runserver
In the browser window, type 127.0.0.1:8000/admin in the address bar.

And fill in the form with the correct username and password:

Which should result in this interface:

Add Members
When you are in the admin interface, click the "Add" button for "Members", and start inserting new members until you have a list like this:

My Tennis Club
In the browser window, type 127.0.0.1:8000/members in the address bar.

And once again you have a Tennis Club page with 5 members!

Next: let's deploy this project, so that the whole world can see it!

Deploy Django - Choose Provider
Deploy to the World
To deploy a project means to make it visible for other people on the internet.

So far, in this tutorial, we have made a Django project that runs locally on your computer. This is often called, "in development", and when we have deployed it, we call it "in production".

Where to Deploy?
There are many providers out there that offers servers for Django projects. In this tutorial we will use the Amazon Web Services (AWS) platform, mainly because they offer a free solution that only requires you to create an AWS account.

Note: you can choose whatever server provider you like, they will all give you the same result, but they will have some provider-specific settings that you should be aware of when following this tutorial.

AWS
Log into your AWS account. (If you do not have an AWS account, follow the steps in the Create AWS Account chapter.)

AWS Console
Once you have signed in, you should be directed to the AWS Console Home page:

Elastic Beanstalk
We will be using a service called "Elastic Beanstalk" to deploy the Django project.

In the search field at the top, search for "elastic beanstalk", and click to start the service:

Lock in Dependencies
Once you have started the "Elastic Beanstalk" service, we could start with the deployment, but first we need to lock in some dependencies, which means to make you local Django project ready for deployment.

You will learn how to in the next chapters.

Deploy Django - Create Requirements
Lock in Dependencies
When you create a Django application, there are some Python packages that your project depends on.

Django itself is a Python package, and we have to make sure that the server where we deploy our project also has the Django package installed, and all the other packages your project requires.

Luckily there is a command for this as well, just run this command in the command view:

```
py -m pip freeze > requirements.txt
```

The result of the above command, is a file called requirements.txt being created in the project:

```
my_tennis_club
    members/
    my_tennis_club/
    mystaticfiles/
    productionfiles/
    db.sqlite3
    manage.py
    requirements.txt
```

The file contains all the packages that this project depends on: with this content:

my_tennis_club/requirements.txt:

```
asgiref==3.5.2
Django==4.1.4
psycopg2-binary==2.9.5
sqlparse==0.4.3
tzdata==2022.7
whitenoise==6.2.0
```

Note: You can create this file on your own, and insert the packages manually, just make sure you get all the packages your project depends on, and you must name the file requirements.txt.

Now the hosting provider knows which packages to install when we deploy our project.

But Elastic Beanstalk needs more information, go to the next chapter to create an "EB" config file.

Deploy Django - django.config
Provider-Specific Settings
We have chosen AWS as our hosting provider, and Elastic Beanstalk as a service to deploy the Django project, and it has some specific requirements.

.ebextension Folder
It requires that you create a folder on the root level of your project

called .ebextensions:

```
my_tennis_club
    .ebextensions/
    members/
    my_tennis_club/
    mystaticfiles/
    productionfiles/
    db.sqlite3
    manage.py
    requirements.txt
```

Create django.config File
In the .ebextensions folder, create a file called django.config:

```
my_tennis_club
    .ebextensions/
        django.config
```

Open the file and insert these settings:

my_tennis_club/.ebextensions/django.config:

```
option_settings:
  aws:elasticbeanstalk:container:python:
    WSGIPath: my_tennis_club.wsgi:application
```

Note: These steps are specific for AWS and Elastic beanstalk, but every provider has some provider-specific settings.

The next step is to wrap all the dependencies into one .zip file, which you will learn in the next chapter.

Deploy Django Project - Create zip File
Zip Your Project
To wrap your project into a .zip file, you cannot zip the entire project folder, but choose the files and folders manually.

The files to include in the .zip file are highlighted (blue) in the example below:

```
my_tennis_club
    .ebextensions/
    members/
    my_tennis_club/
    mystaticfiles/
    productionfiles/
    db.sqlite3
    manage.py
    requirements.txt
```

With your file explorer, navigate to the project folder, select these files and folders, right-click and choose to create a zip file.

Zip File
Now you have a .zip file of your project which you can upload to Elastic beanstalk:

```
my_tennis_club
    .ebextensions/
    members/
    my_tennis_club/
    mystaticfiles/
    productionfiles/
    db.sqlite3
    manage.py
```

my_tennis_clup.zip
    requirements.txt

Deploy Django - Elastic Beanstalk
Elastic Beanstalk
In AWS, navigate to the Elastic Beanstalk application, as we did in the Choose Provider chapter, and click the "Create application" button:

Create EB Application
Once you have clicked the "Create Application" button, you will be taken to this page, where you can give your Django project a name. I will name it "my-first-django":

Choose Platform
Then scroll down until you see the "Platform" section, and choose "Python", with the recommended version:

Upload .zip File
Next, scroll down to the next section, the "Application code" section, and choose "Upload your code".

Click on the "Choose file" button, navigate to the .zip file you created in the previous chapter and upload it:

The .zip file is uploaded, and we are ready to deploy!

Deploy
Click the "Create application" button to start deploying.

Waiting
The deployment will take a few minutes.

Success!
Finally the project is live, and you can view it by clicking the link below the Myfirstdjango-env header, or type the host address in your browser:

Deploy Django - Update Project
Deploy Changes
Any changes you do with the application locally, on your own computer, is not visible on the deployed version.

So if you make a change, and you want it to be visible on the deployed version, you have to upload a new .zip file.

Create .zip file
To wrap your project into a .zip file, follow the steps described in the Create .zip File chapter:

Start by selecting the relevant files and folders of your project, if you have the same project as we have in our tutorial, you should select the highlighted files in the example below:

```
my_tennis_club
    .ebextensions/
    members/
    my_tennis_club/
    mystaticfiles/
    productionfiles/
    db.sqlite3
    manage.py
    requirements.txt
```
Right-click and choose to create a .zip file.

Now you have a .zip file, containing the changes, and you can upload it to Elastic beanstalk:

```
my_tennis_club
    .ebextensions/
    members/
    my_tennis_club/
    mystaticfiles/
    productionfiles/
    db.sqlite3
    manage.py
    my_tennis_clup.zip
    requirements.txt
```
Upload to Elastic Beanstalk

Log into your Amazon Web Services account, and find your project under the "Elastic Beanstalk" application:

Click the "Upload and deploy" button.

Choose .zip File

Click the "Choose file" button to upload the .zip file you just created:

Deploy

Click the "Deploy" button:

Uploaded!

That's it, your project is updated with all the new changes.

Note: Follow these steps every time you want to update your project.

Django Slug Field

What is Slug?

Have you ever seen url's that look like this:

w3schools.com/django/learn-about-slug-field

The "learn-about-slug-field" part is a slug.

It is a description containing only letters, hyphens, numbers or underscores.

It is often used in url's to make them easier to read, but also to make them more search engine friendly.

Url Without Slug

If you have followed our Django Project created in this tutorial, you will have a small Django project looking like this:

And if you click the first member, you will jump to this page:

Check out the address bar:

127.0.0.1:8000/members/details/1
The number "1" refers to the ID of that particular record in the database.

Makes sense to the developer, but probably not to anyone else.

Url With Slug
It would have made more sense if the url looked like this:

Check out the address bar:

127.0.0.1:8000/members/details/emil-refsnes
That is a more user friendly url, and Django can help you create such url's in your project.

Modify the models.py File
Start by adding a new field in the database.

Open the models.py file and add a field called slug with the data type SlugField:

my_tennis_club/members/models.py:

```
from django.db import models

class Member(models.Model):
  firstname = models.CharField(max_length=255)
  lastname = models.CharField(max_length=255)
  phone = models.IntegerField(null=True)
  joined_date = models.DateField(null=True)
  slug = models.SlugField(default="", null=False)

  def __str__(self):
    return f"{self.firstname} {self.lastname}"
```

This is a change in the Model's structure, and therefor we have to make a migration to tell Django that it has to update the database:

```
py manage.py makemigrations
```

And the migrate command:

```
py manage.py migrate
```

Change Admin
Now we have a new field in the database, but we also want this field to be updated automatically when we set the firstname or lastname of a member.

This can be done with a built-in Django feature called prepopulated_fields where you specify the field you want to pre-populate, and a tuple with the field(s) you want to populate it with.

This is done in the admin.py file:

my_tennis_club/members/admin.py:

```
from django.contrib import admin
from .models import Member

# Register your models here.

class MemberAdmin(admin.ModelAdmin):
  list_display = ("firstname", "lastname", "joined_date",)
  prepopulated_fields = {"slug": ("firstname", "lastname")}

admin.site.register(Member, MemberAdmin)
```

Enter the Admin interface and open a record for editing:

Click "SAVE" and the "slug" field will be auto populated with the firstname and the lastname, and since the "slug" field is of type SlugField, it will "slugify" the value, meaning it will put a hyphen between each word.

Next time you open the member for editing you will see the slug field with value:

Note: Since the new field is empty by default, you have to do this save operation for each member.

Modify Template
Now we can replace the ID field with the slug field throughout the project.

Start with the all_members.html template, where we have a link to the details page:

my_tennis_club/members/templates/all_members.html:

```
{% extends "master.html" %}

{% block title %}
  My Tennis Club - List of all members
{% endblock %}


{% block content %}
  <div class="mycard">
    <h1>Members</h1>
    <ul>
      {% for x in mymembers %}
        <li onclick="window.location = 'details/{{ x.slug }}'">{{ x.firstname }}
{{ x.lastname }}</li>
      {% endfor %}
    </ul>
  </div>
{% endblock %}
```

Modify URL
We also have to make some changes in the urls.pyfile.

Change from <int:id> to <slug:slug>:

my_tennis_club/members/urls.py:

```
from django.urls import path
from . import views
```

```
urlpatterns = [
    path('', views.main, name='main'),
    path('members/', views.members, name='members'),
    path('members/details/<slug:slug>', views.details, name='details'),
    path('testing/', views.testing, name='testing'),
]
```

Modify View

Finally, change the details view to handle incoming request as slug instead of ID:

my_tennis_club/members/views.py:

```
from django.http import HttpResponse
from django.template import loader
from .models import Member

def members(request):
  mymembers = Member.objects.all().values()
  template = loader.get_template('all_members.html')
  context = {
    'mymembers': mymembers,
  }
  return HttpResponse(template.render(context, request))

def details(request, slug):
  mymember = Member.objects.get(slug=slug)
  template = loader.get_template('details.html')
  context = {
    'mymember': mymember,
  }
  return HttpResponse(template.render(context, request))

def main(request):
  template = loader.get_template('main.html')
  return HttpResponse(template.render())

def testing(request):
  template = loader.get_template('template.html')
  context = {
    'fruits': ['Apple', 'Banana', 'Cherry'],
  }
  return HttpResponse(template.render(context, request))
```

Now the link to details works with the new slugified url:

If you have followed all the steps on your own computer, you can see the result in your own browser: 127.0.0.1:8000/members/.

If the server is down, you have to start it again with the runserver command:

```
py manage.py runserver
```

Django - Add Static File

Add Bootstrap 5

There are two main methods to use bootstrap in your Django project. Either by downloading the required files and include them in your project, or you can install the bootstrap 5 module in your virtual environment.

We will use the second method, installing Bootstrap 5 in the virtual environment.

Install Bootstrap 5

Bootstrap 5 should be installed in the virtual environment.

We will install it in an existing project, the My Tennis Club project, created earlier in this tutorial.

Open the command view, navigate to the virtual environment folder and activate the virtual environment:

```
Scripts\activate.bat
```

Once you are inside the virtual environment, install Bootstrap 5 with this command:

```
pip install django-bootstrap-v5
```

Which will give you a result like this:

```
Collecting django-bootstrap-v5
Downloading django_bootstrap_v5-1.0.11-py3-none-any.whl (24 kB)
Requirement already satisfied: django<5.0,>=2.2 in c:\users\your name\myworld\
lib\site-packages (from django-bootstrap-v5) (4.1.4)
Collecting beautifulsoup4<5.0.0,>=4.8.0
  Downloading beautifulsoup4-4.11.1-py3-none-any.whl (128 kB)
     |âââââââââââââââââââââââââââââââââ| 128 kB 6.4 MB/s
Requirement already satisfied: tzdata; sys_platform == "win32" in c:\users\your
name\myworld\lib\site-packages (from django<5.0,>=2.2->django-bootstrap-v5)
(2022.7)
Requirement already satisfied: asgiref<4,>=3.5.2 in c:\users\your name\myworld\
lib\site-packages (from django<5.0,>=2.2->django-bootstrap-v5) (3.5.2)
Requirement already satisfied: sqlparse>=0.2.2 in c:\users\your name\myworld\
lib\site-packages (from django<5.0,>=2.2->django-bootstrap-v5) (0.4.3)
Collecting soupsieve>1.2
  Downloading soupsieve-2.3.2.post1-py3-none-any.whl (37 kB)
Installing collected packages: soupsieve, beautifulsoup4, django-bootstrap-v5
Successfully installed beautifulsoup4-4.11.1 django-bootstrap-v5-1.0.11
soupsieve-2.3.2.post1
```

Update Settings

Next step is to include the bootstrap module in the INSTALLED_APPS list in settings.py:

my_tennis_club/my_tennis_club/settings.py:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'members',
    'bootstrap5',
]
```

Bootstrap 5 is now ready to use in your project!

Remove Old Styling

ThThe My Tennis Club project already has a stylesheet, remove it and the Members page without styling will look like this:

Add Bootstrap 5 to Template

To use Bootstrap 5 in the project, start by inserting some lines of code in the master.html template:

my_tennis_club/members/templates/master.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>{% block title %}{% endblock %}</title>
  {% load bootstrap5 %}
  {% bootstrap_css %}
  {% bootstrap_javascript %}
</head>
<body>

<div class="container">
  <ul class="nav bg-info">
    <li class="nav-item">
      <a class="nav-link link-light" href="/">HOME</a>
    </li>
    <li class="nav-item">
      <a class="nav-link link-light" href="/members">MEMBERS</a>
    </li>
  </ul>

  {% block content %}
  {% endblock %}
</div>
</body>
</html>
```

As you can see, we inserted these three lines in the <head> section:

```
  {% load bootstrap5 %}
  {% bootstrap_css %}
  {% bootstrap_javascript %}
```

The first line tells Django that it should load the Bootstrap 5 module in this template.

The second line inserts the <link> element with the referral to the bootstrap stylesheet.

The third line inserts the <script> element with the referral to the necessary javascript file.

We also did some changes to the HTML elements in the template, like inserting bootstrap classes to the navigation bar:

```
<div class="container">
  <ul class="nav bg-info">
    <li class="nav-item">
      <a class="nav-link link-light" href="/">HOME</a>
    </li>
    <li class="nav-item">
      <a class="nav-link link-light" href="/members">MEMBERS</a>
    </li>
```

```
  </ul>
  {% block content %}
  {% endblock %}
</div>
```
If you run the project now, the members page will look like this:

That's it! Bootstrap 5 is now a part of your project!

Learn more about Bootstrap 5 in our Bootstrap 5 Tutorial.

Django Template Tags Reference
Template Tags Reference
A list of all template tags:

Tag	Description
autoescape	Specifies if autoescape mode is on or off
block	Specifies a block section
comment	Specifies a comment section
csrf_token	Protects forms from Cross Site Request Forgeries
cycle	Specifies content to use in each cycle of a loop
debug	Specifies debugging information
extends	Specifies a parent template
filter	Filters content before returning it
firstof	Returns the first not empty variable
for	Specifies a for loop
if	Specifies a if statement
ifchanged	Used in for loops. Outputs a block only if a value has changed since the last iteration
include	Specifies included content/template
load	Loads template tags from another library
lorem	Outputs random text
now	Outputs the current date/time
regroup	Sorts an object by a group
resetcycle	Used in cycles. Resets the cycle
spaceless	Removes whitespace between HTML tags
templatetag	Outputs a specified template tag
url	Returns the absolute URL part of a URL
verbatim	Specifies contents that should not be rendered by the template engine
widthratio	Calculates a width value based on the ratio between a given value and a max value
with	Specifies a variable to use in the block

Filter Reference
Filter Reference
A list of all filter keywords:

Keyword	Description
add	Adds a specified value.
addslashes	Adds a slash before any quote characters, to escape strings.
capfirst	Returns the first letter in uppercase.
center	Centers the value in the middle of a specified width.
cut	Removes any specified character or phrases.
date	Returns dates in the specified format.
default	Returns a specified value if the value is False.
default_if_none	Returns a specified value if the value is None.
dictsort	Sorts a dictionary by the given value.
dictsortreversed	Sorts a dictionary reversed, by the given value.

divisibleby Returns True if the value can be divided by the specified number, otherwise it returns False.
escape Escapes HTML code from a string.
escapejs Escapes JavaScript code from a string.
filesizeformat Returns a number into a file size format.
first Returns the first item of an object (for Strings, the first character is returned).
floatformat Rounds floating numbers to a specified number of decimals, default one decimal.
force_escape Escapes HTML code from a string.
get_digit Returns a specific digit of a number.
iriencode Convert an IRI into a URL friendly string.
join Returns the items of a list into a string.
json_script Returns an object into a JSON object surrounded by <script></script> tags.
last Returns the last item of an object (for Strings, the last character is returned).
length Returns the number of items in an object, or the number of characters in a string.
length_is Returns True if the length is the same as the specified number
linebreaks Returns the text with <br> instead of line breaks, and <p> instead of more than one line break.
linebreaksbr Returns the text with <br> instead of line breaks.
linenumbers Returns the text with line numbers for each line.
ljust Left aligns the value according to a specified width
lower Returns the text in lower case letters.
make_list Converts a value into a list object.
phone2numeric Converts phone numbers with letters into numeric phone numbers.
pluralize Adds a 's' at the end of a value if the specified numeric value is not 1.
pprint
random Returns a random item of an object
rjust Right aligns the value according to a specified width
safe Marks that this text is safe and should not be HTML escaped.
safeseq Marks each item of an object as safe and the item should not be HTML escaped.
slice Returns a specified slice of a text or object.
slugify Converts text into one long alphanumeric-lower-case word.
stringformat Converts the value into a specified format.
striptags Removes HTML tags from a text.
time Returns a time in the specified format.
timesince Returns the difference between two datetimes.
timeuntil Returns the difference between two datetimes.
title Upper cases the first character of each word in a text, all other characters are converted to lower case.
truncatechars Shortens a string into the specified number of characters.
truncatechars_html Shortens a string into the specified number of characters, not considering the length of any HTML tags.
truncatewords Shortens a string into the specified number of words.
truncatewords_html Shortens a string into the specified number of words, not considering any HTML tags.
unordered_list Returns the items of an object as an unordered HTML list.
upper Returns the text in upper case letters.
urlencode URL encodes a string.
urlize Returns any URLs in a string as HTML links.
urlizetrunc Returns any URLs in a string as HTML links, but shortens the links into the specified number of characters.
wordcount Returns the number of words in a text.
wordwrap Wrap words at a specified number of characters.
yesno Converts Booleans values into specified values.
i18n
l10n
tz

QuerySet Field Lookups Reference
Field Lookups Reference
A list of all field look up keywords:

Keyword	Description
contains	Contains the phrase
icontains	Same as contains, but case-insensitive
date	Matches a date
day	Matches a date (day of month, 1-31) (for dates)
endswith	Ends with
iendswith	Same as endswidth, but case-insensitive
exact	An exact match
iexact	Same as exact, but case-insensitive
in	Matches one of the values
isnull	Matches NULL values
gt	Greater than
gte	Greater than, or equal to
hour	Matches an hour (for datetimes)
lt	Less than
lte	Less than, or equal to
minute	Matches a minute (for datetimes)
month	Matches a month (for dates)
quarter	Matches a quarter of the year (1-4) (for dates)
range	Match between
regex	Matches a regular expression
iregex	Same as regex, but case-insensitive
second	Matches a second (for datetimes)
startswith	Starts with
istartswith	Same as startswith, but case-insensitive
time	Matches a time (for datetimes)
week	Matches a week number (1-53) (for dates)
week_day	Matches a day of week (1-7) 1 is Sunday
iso_week_day	Matches a ISO 8601 day of week (1-7) 1 is Monday
year	Matches a year (for dates)
iso_year	Matches an ISO 8601 year (for dates)

Matplotlib Tutorial

What is Matplotlib?
Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

Matplotlib was created by John D. Hunter.

Matplotlib is open source and we can use it freely.

Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

Where is the Matplotlib Codebase?
The source code for Matplotlib is located at this github repository
https://github.com/matplotlib/matplotlib

Matplotlib Getting Started
Installation of Matplotlib
If you have Python and PIP already installed on a system, then installation of Matplotlib is very easy.

Install it using this command:

```
C:\Users\Your Name>pip install matplotlib
```
If this command fails, then use a python distribution that already has Matplotlib installed, like Anaconda, Spyder etc.

Import Matplotlib
Once Matplotlib is installed, import it in your applications by adding the import module statement:

```
import matplotlib
```
Now Matplotlib is imported and ready to use:

Checking Matplotlib Version
The version string is stored under __version__ attribute.

ExampleGet your own Python Server
```
import matplotlib

print(matplotlib.__version__)
```
Note: two underscore characters are used in __version__.

Matplotlib Pyplot
Pyplot
Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

```
import matplotlib.pyplot as plt
```
Now the Pyplot package can be referred to as plt.

ExampleGet your own Python Server
Draw a line in a diagram from position (0,0) to position (6,250):

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```
Result:

You will learn more about drawing (plotting) in the next chapters.

Matplotlib Plotting
Plotting x and y points
The plot() function is used to draw points (markers) in a diagram.

By default, the plot() function draws a line from point to point.

The function takes parameters for specifying points in the diagram.

Parameter 1 is an array containing the points on the x-axis.

Parameter 2 is an array containing the points on the y-axis.

If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.

ExampleGet your own Python Server
Draw a line in a diagram from position (1, 3) to position (8, 10):

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints)
plt.show()
```

Result:

The x-axis is the horizontal axis.

The y-axis is the vertical axis.

Plotting Without Line
To plot only the markers, you can use shortcut string notation parameter 'o', which means 'rings'.

Example
Draw two points in the diagram, one at position (1, 3) and one in position (8, 10):

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints, 'o')
plt.show()
```

Result:

You will learn more about markers in the next chapter.

Multiple Points
You can plot as many points as you like, just make sure you have the same number of points in both axis.

Example
Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10):

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

plt.plot(xpoints, ypoints)
plt.show()
```

Result:

Default X-Points
If we do not specify the points on the x-axis, they will get the default values

0, 1, 2, 3 etc., depending on the length of the y-points.

So, if we take the same example as above, and leave out the x-points, the diagram will look like this:

Example
Plotting without x-points:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10, 5, 7])

plt.plot(ypoints)
plt.show()
```

Result:

The x-points in the example above are [0, 1, 2, 3, 4, 5].

Matplotlib Markers
Markers
You can use the keyword argument marker to emphasize each point with a specified marker:

ExampleGet your own Python Server
Mark each point with a circle:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o')
plt.show()
```

Result:

Example
Mark each point with a star:

```
...
plt.plot(ypoints, marker = '*')
...
```

Result:

Marker Reference
You can choose any of these markers:

Marker	Description
'o'	Circle
'*'	Star
'.'	Point
','	Pixel
'x'	X
'X'	X (filled)
'+'	Plus
'P'	Plus (filled)
's'	Square

'D'   Diamond
'd'   Diamond (thin)
'p'   Pentagon
'H'   Hexagon
'h'   Hexagon
'v'   Triangle Down
'^'   Triangle Up
'<'   Triangle Left
'>'   Triangle Right
'1'   Tri Down
'2'   Tri Up
'3'   Tri Left
'4'   Tri Right
'|'   Vline
'_'   Hline

Format Strings fmt

You can also use the shortcut string notation parameter to specify the marker.

This parameter is also called fmt, and is written with this syntax:

marker|line|color

Example

Mark each point with a circle:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, 'o:r')
plt.show()
```

Result:

The marker value can be anything from the Marker Reference above.

The line value can be one of the following:

Line Reference

Line Syntax	Description
'-'	Solid line
':'	Dotted line
'--'	Dashed line
'-.'	Dashed/dotted line

Note: If you leave out the line value in the fmt parameter, no line will be plotted.

The short color value can be one of the following:

Color Reference

Color Syntax	Description
'r'	Red
'g'	Green
'b'	Blue
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

Marker Size

You can use the keyword argument markersize or the shorter version, ms to set the size of the markers:

Example

Set the size of the markers to 20:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20)
plt.show()
```

Result:

Marker Color

You can use the keyword argument markeredgecolor or the shorter mec to set the color of the edge of the markers:

Example

Set the EDGE color to red:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r')
plt.show()
```

Result:

You can use the keyword argument markerfacecolor or the shorter mfc to set the color inside the edge of the markers:

Example

Set the FACE color to red:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20, mfc = 'r')
plt.show()
```

Result:

Use both the mec and mfc arguments to color the entire marker:

Example

Set the color of both the edge and the face to red:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r', mfc = 'r')
plt.show()
```

Result:

You can also use Hexadecimal color values:

Example

Mark each point with a beautiful green color:

```
...
plt.plot(ypoints, marker = 'o', ms = 20, mec = '#4CAF50', mfc = '#4CAF50')
...
```
Result:

Or any of the 140 supported color names.

Example
Mark each point with the color named "hotpink":

```
...
plt.plot(ypoints, marker = 'o', ms = 20, mec = 'hotpink', mfc = 'hotpink')
...
```
Result:

Matplotlib Line
Linestyle
You can use the keyword argument linestyle, or shorter ls, to change the style of the plotted line:

ExampleGet your own Python Server
Use a dotted line:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linestyle = 'dotted')
plt.show()
```
Result:

Example
Use a dashed line:

```
plt.plot(ypoints, linestyle = 'dashed')
```

Result:

Shorter Syntax
The line style can be written in a shorter syntax:

linestyle can be written as ls.

dotted can be written as :.

dashed can be written as --.

Example
Shorter syntax:

```
plt.plot(ypoints, ls = ':')
```
Result:

Line Styles
You can choose any of these styles:

Style	Or
'solid' (default)	'-'
'dotted'	':'
'dashed'	'--'
'dashdot'	'-.'
'None'	'' or ' '

Line Color
You can use the keyword argument color or the shorter c to set the color of the line:

Example
Set the line color to red:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, color = 'r')
plt.show()
```

Result:

You can also use Hexadecimal color values:

Example
Plot with a beautiful green line:

```
...
plt.plot(ypoints, c = '#4CAF50')
...
```

Result:

Or any of the 140 supported color names.

Example
Plot with the color named "hotpink":

```
...
plt.plot(ypoints, c = 'hotpink')
...
```

Result:

Line Width
You can use the keyword argument linewidth or the shorter lw to change the width of the line.

The value is a floating number, in points:

Example
Plot with a 20.5pt wide line:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linewidth = '20.5')
plt.show()
```

Result:

Multiple Lines
You can plot as many lines as you like by simply adding more plt.plot() functions:

Example
Draw two lines by specifying a plt.plot() function for each line:

```
import matplotlib.pyplot as plt
import numpy as np

y1 = np.array([3, 8, 1, 10])
y2 = np.array([6, 2, 7, 11])

plt.plot(y1)
plt.plot(y2)

plt.show()
```

Result:

You can also plot many lines by adding the points for the x- and y-axis for each line in the same plt.plot() function.

(In the examples above we only specified the points on the y-axis, meaning that the points on the x-axis got the the default values (0, 1, 2, 3).)

The x- and y- values come in pairs:

Example
Draw two lines by specifiyng the x- and y-point values for both lines:

```
import matplotlib.pyplot as plt
import numpy as np

x1 = np.array([0, 1, 2, 3])
y1 = np.array([3, 8, 1, 10])
x2 = np.array([0, 1, 2, 3])
y2 = np.array([6, 2, 7, 11])

plt.plot(x1, y1, x2, y2)
plt.show()
```

Result:

Matplotlib Labels and Title
Create Labels for a Plot
With Pyplot, you can use the xlabel() and ylabel() functions to set a label for the x- and y-axis.

ExampleGet your own Python Server
Add labels to the x- and y-axis:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)
```

```
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```
Result:

Create a Title for a Plot

With Pyplot, you can use the title() function to set a title for the plot.

Example

Add a plot title and labels for the x- and y-axis:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```
Result:

Set Font Properties for Title and Labels

You can use the fontdict parameter in xlabel(), ylabel(), and title() to set font properties for the title and labels.

Example

Set font properties for the title and labels:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}

plt.title("Sports Watch Data", fontdict = font1)
plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burnage", fontdict = font2)

plt.plot(x, y)
plt.show()
```
Result:

Position the Title

You can use the loc parameter in title() to position the title.

Legal values are: 'left', 'right', and 'center'. Default value is 'center'.

Example

Position the title to the left:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data", loc = 'left')
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)
plt.show()
```

Result:

Matplotlib Adding Grid Lines

Add Grid Lines to a Plot

With Pyplot, you can use the grid() function to add grid lines to the plot.

ExampleGet your own Python Server

Add grid lines to the plot:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid()

plt.show()
```

Result:

Specify Which Grid Lines to Display

You can use the axis parameter in the grid() function to specify which grid lines to display.

Legal values are: 'x', 'y', and 'both'. Default value is 'both'.

Example

Display only grid lines for the x-axis:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)
```

```
plt.grid(axis = 'x')

plt.show()
```
Result:

Example
Display only grid lines for the y-axis:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid(axis = 'y')

plt.show()
```
Result:

Set Line Properties for the Grid
You can also set the line properties of the grid, like this: grid(color = 'color', linestyle = 'linestyle', linewidth = number).

Example
Set the line properties of the grid:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)

plt.show()
```
Result:

Matplotlib Subplot
Display Multiple Plots
With the subplot() function you can draw multiple plots in one figure:

ExampleGet your own Python Server
Draw 2 plots:

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
```

```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)

plt.show()
```

Result:

The subplot() Function

The subplot() function takes three arguments that describes the layout of the figure.

The layout is organized in rows and columns, which are represented by the first and second argument.

The third argument represents the index of the current plot.

```
plt.subplot(1, 2, 1)
#the figure has 1 row, 2 columns, and this plot is the first plot.

plt.subplot(1, 2, 2)
#the figure has 1 row, 2 columns, and this plot is the second plot.
```

So, if we want a figure with 2 rows an 1 column (meaning that the two plots will be displayed on top of each other instead of side-by-side), we can write the syntax like this:

Example

Draw 2 plots on top of each other:

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 1, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 1, 2)
plt.plot(x,y)

plt.show()
```

Result:

You can draw as many plots you like on one figure, just descibe the number of rows, columns, and the index of the plot.

Example

Draw 6 plots:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 1)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 2)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 5)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 6)
plt.plot(x,y)

plt.show()
```

Result:

Title

You can add a title to each plot with the title() function:

Example

2 plots, with titles:

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

#plot 2:
```

```
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

plt.show()
```

Result:

Super Title

You can add a title to the entire figure with the suptitle() function:

Example

Add a title for the entire figure:

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

plt.suptitle("MY SHOP")
plt.show()
```

Result:

Matplotlib Scatter

Creating Scatter Plots

With Pyplot, you can use the scatter() function to draw a scatter plot.

The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

ExampleGet your own Python Server

A simple scatter plot:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x, y)
plt.show()
```

Result:

The observation in the example above is the result of 13 cars passing by.

The X-axis shows how old the car is.

The Y-axis shows the speed of the car when it passes.

Are there any relationships between the observations?

It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

Compare Plots

In the example above, there seems to be a relationship between speed and age, but what if we plot the observations from another day as well? Will the scatter plot tell us something else?

Example

Draw two plots on the same figure:

```
import matplotlib.pyplot as plt
import numpy as np

#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)

plt.show()
```

Result:

Note: The two plots are plotted with two different colors, by default blue and orange, you will learn how to change colors later in this chapter.

By comparing the two plots, I think it is safe to say that they both gives us the same conclusion: the newer the car, the faster it drives.

Colors

You can set your own color for each scatter plot with the color or the c argument:

Example

Set your own color of the markers:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y, color = 'hotpink')

x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y, color = '#88c999')

plt.show()
```

Result:

Color Each Dot
You can even set a specific color for each dot by using an array of colors as value for the c argument:

Note: You cannot use the color argument for this, only the c argument.

Example
Set your own color of the markers:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = 
np.array(["red","green","blue","yellow","pink","black","orange","purple","beige"
,"brown","gray","cyan","magenta"])

plt.scatter(x, y, c=colors)

plt.show()
```

Result:

ColorMap
The Matplotlib module has a number of available colormaps.

A colormap is like a list of colors, where each color has a value that ranges from 0 to 100.

Here is an example of a colormap:

This colormap is called 'viridis' and as you can see it ranges from 0, which is a purple color, up to 100, which is a yellow color.

How to Use the ColorMap
You can specify the colormap with the keyword argument cmap with the value of the colormap, in this case 'viridis' which is one of the built-in colormaps available in Matplotlib.

In addition you have to create an array with values (from 0 to 100), one value for each point in the scatter plot:

Example
Create a color array, and specify a colormap in the scatter plot:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='viridis')

plt.show()
```

Result:

You can include the colormap in the drawing by including the plt.colorbar()

statement:

Example
Include the actual colormap:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='viridis')

plt.colorbar()

plt.show()
```

Result:

Available ColorMaps
You can choose any of the built-in colormaps:

Name	Reverse
Accent	Accent_r
Blues	Blues_r
BrBG	BrBG_r
BuGn	BuGn_r
BuPu	BuPu_r
CMRmap	CMRmap_r
Dark2	Dark2_r
GnBu	GnBu_r
Greens	Greens_r
Greys	Greys_r
OrRd	OrRd_r
Oranges	Oranges_r
PRGn	PRGn_r
Paired	Paired_r
Pastel1	Pastel1_r
Pastel2	Pastel2_r
PiYG	PiYG_r
PuBu	PuBu_r
PuBuGn	PuBuGn_r
PuOr	PuOr_r
PuRd	PuRd_r
Purples	Purples_r
RdBu	RdBu_r
RdGy	RdGy_r
RdPu	RdPu_r
RdYlBu	RdYlBu_r
RdYlGn	RdYlGn_r
Reds	Reds_r
Set1	Set1_r
Set2	Set2_r
Set3	Set3_r
Spectral	Spectral_r
Wistia	Wistia_r
YlGn	YlGn_r
YlGnBu	YlGnBu_r
YlOrBr	YlOrBr_r
YlOrRd	YlOrRd_r
afmhot	afmhot_r
autumn	autumn_r
binary	binary_r

```
bone            bone_r
brg             brg_r
bwr             bwr_r
cividis         cividis_r
cool            cool_r
coolwarm        coolwarm_r
copper          copper_r
cubehelix       cubehelix_r
flag            flag_r
gist_earth      gist_earth_r
gist_gray       gist_gray_r
gist_heat       gist_heat_r
gist_ncar       gist_ncar_r
gist_rainbow    gist_rainbow_r
gist_stern      gist_stern_r
gist_yarg       gist_yarg_r
gnuplot         gnuplot_r
gnuplot2        gnuplot2_r
gray            gray_r
hot             hot_r
hsv             hsv_r
inferno         inferno_r
jet             jet_r
magma           magma_r
nipy_spectral   nipy_spectral_r
ocean           ocean_r
pink            pink_r
plasma          plasma_r
prism           prism_r
rainbow         rainbow_r
seismic         seismic_r
spring          spring_r
summer          summer_r
tab10           tab10_r
tab20           tab20_r
tab20b          tab20b_r
tab20c          tab20c_r
terrain         terrain_r
twilight        twilight_r
twilight_shifted        twilight_shifted_r
viridis         viridis_r
winter          winter_r
```

Size

You can change the size of the dots with the s argument.

Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis:

Example

Set your own size for the markers:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])

plt.scatter(x, y, s=sizes)

plt.show()
```

Result:

Alpha
You can adjust the transparency of the dots with the alpha argument.

Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis:

Example
Set your own size for the markers:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])

plt.scatter(x, y, s=sizes, alpha=0.5)

plt.show()
```

Result:

Combine Color Size and Alpha
You can combine a colormap with different sizes of the dots. This is best visualized if the dots are transparent:

Example
Create random arrays with 100 values for x-points, y-points, colors and sizes:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.randint(100, size=(100))
y = np.random.randint(100, size=(100))
colors = np.random.randint(100, size=(100))
sizes = 10 * np.random.randint(100, size=(100))

plt.scatter(x, y, c=colors, s=sizes, alpha=0.5, cmap='nipy_spectral')

plt.colorbar()

plt.show()
```

Result:

Matplotlib Bars
Creating Bars
With Pyplot, you can use the bar() function to draw bar graphs:

ExampleGet your own Python Server
Draw 4 bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y)
plt.show()
```

Result:

The bar() function takes arguments that describes the layout of the bars.

The categories and their values represented by the first and second argument as arrays.

Example

```
x = ["APPLES", "BANANAS"]
y = [400, 350]
plt.bar(x, y)
```


Horizontal Bars

If you want the bars to be displayed horizontally instead of vertically, use the barh() function:

Example

Draw 4 horizontal bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y)
plt.show()
```

Result:

Bar Color

The bar() and barh() take the keyword argument color to set the color of the bars:

Example

Draw 4 red bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "red")
plt.show()
```

Result:

Color Names

You can use any of the 140 supported color names.

Example

Draw 4 "hot pink" bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "hotpink")
plt.show()
```

Result:

Color Hex
Or you can use Hexadecimal color values:

Example
Draw 4 bars with a beautiful green color:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "#4CAF50")
plt.show()
```

Result:

Bar Width
The bar() takes the keyword argument width to set the width of the bars:

Example
Draw 4 very thin bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, width = 0.1)
plt.show()
```

Result:

The default width value is 0.8

Note: For horizontal bars, use height instead of width.

Bar Height
The barh() takes the keyword argument height to set the height of the bars:

Example
Draw 4 very thin bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y, height = 0.1)
plt.show()
```

Result:

The default height value is 0.8

Matplotlib Histograms
Histogram
A histogram is a graph showing frequency distributions.

It is a graph showing the number of observations within each given interval.

Example: Say you ask for the height of 250 people, you might end up with a histogram like this:

You can read from the histogram that there are approximately:

2 people from 140 to 145cm
5 people from 145 to 150cm
15 people from 151 to 156cm
31 people from 157 to 162cm
46 people from 163 to 168cm
53 people from 168 to 173cm
45 people from 173 to 178cm
28 people from 179 to 184cm
21 people from 185 to 190cm
4 people from 190 to 195cm

Create Histogram
In Matplotlib, we use the hist() function to create histograms.

The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

For simplicity we use NumPy to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10. Learn more about Normal Data Distribution in our Machine Learning Tutorial.

ExampleGet your own Python Server
A Normal Data Distribution by NumPy:

```
import numpy as np

x = np.random.normal(170, 10, 250)

print(x)
```

Result:
This will generate a random result, and could look like this:

```
  [167.62255766 175.32495609 152.84661337 165.50264047 163.17457988
   162.29867872 172.83638413 168.67303667 164.57361342 180.81120541
   170.57782187 167.53075749 176.15356275 176.95378312 158.4125473
   187.8842668  159.03730075 166.69284332 160.73882029 152.22378865
   164.01255164 163.95288674 176.58146832 173.19849526 169.40206527
   166.88861903 149.90348576 148.39039643 177.90349066 166.72462233
   177.44776004 170.93335636 173.26312881 174.76534435 162.28791953
   166.77301551 160.53785202 170.67972019 159.11594186 165.36992993
   178.38979253 171.52158489 173.32636678 159.63894401 151.95735707
   175.71274153 165.00458544 164.80607211 177.50988211 149.28106703
   179.43586267 181.98365273 170.98196794 179.1093176  176.91855744
   168.32092784 162.33939782 165.18364866 160.52300507 174.14316386
   163.01947601 172.01767945 173.33491959 169.75842718 198.04834503
   192.82490521 164.54557943 206.36247244 165.47748898 195.26377975
   164.37569092 156.15175531 162.15564208 179.34100362 167.22138242
   147.23667125 162.86940215 167.84986671 172.99302505 166.77279814
   196.6137667  159.79012341 166.5840824  170.68645637 165.62204521
   174.5559345  165.0079216  187.92545129 166.86186393 179.78383824
   161.0973573  167.44890343 157.38075812 151.35412246 171.3107829
   162.57149341 182.49985133 163.24700057 168.72639903 169.05309467
   167.19232875 161.06405208 176.87667712 165.48750185 179.68799986
   158.7913483  170.22465411 182.66432721 173.5675715  176.85646836
```

```
   157.31299754 174.88959677 183.78323508 174.36814558 182.55474697
   180.03359793 180.53094948 161.09560099 172.29179934 161.22665588
   171.88382477 159.04626132 169.43886536 163.75793589 157.73710983
   174.68921523 176.19843414 167.39315397 181.17128255 174.2674597
   186.05053154 177.06516302 171.78523683 166.14875436 163.31607668
   174.01429569 194.98819875 169.75129209 164.25748789 180.25773528
   170.44784934 157.81966006 171.33315907 174.71390637 160.55423274
   163.92896899 177.29159542 168.30674234 165.42853878 176.46256226
   162.61719142 166.60810831 165.83648812 184.83238352 188.99833856
   161.3054697  175.30396693 175.28109026 171.54765201 162.08762813
   164.53011089 189.86213299 170.83784593 163.25869004 198.68079225
   166.95154328 152.03381334 152.25444225 149.75522816 161.79200594
   162.13535052 183.37298831 165.40405341 155.59224806 172.68678385
   179.35359654 174.19668349 163.46176882 168.26621173 162.97527574
   192.80170974 151.29673582 178.65251432 163.17266558 165.11172588
   183.11107905 169.69556831 166.35149789 178.74419135 166.28562032
   169.96465166 178.24368042 175.3035525  170.16496554 158.80682882
   187.10006553 178.90542991 171.65790645 183.19289193 168.17446717
   155.84544031 177.96091745 186.28887898 187.89867406 163.26716924
   169.71242393 152.9410412  158.68101969 171.12655559 178.1482624
   187.45272185 173.02872935 163.8047623  169.95676819 179.36887054
   157.01955088 185.58143864 170.19037101 157.221245   168.90639755
   178.7045601  168.64074373 172.37416382 165.61890535 163.40873027
   168.98683006 149.48186389 172.20815568 172.82947206 173.71584064
   189.42642762 172.79575803 177.00005573 169.24498561 171.55576698
   161.36400372 176.47928342 163.02642822 165.09656415 186.70951892
   153.27990317 165.59289527 180.34566865 189.19506385 183.10723435
   173.48070474 170.28701875 157.24642079 157.9096498  176.4248199 ]
```

The hist() function will read the array and produce a histogram:

Example
A simple histogram:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()
```
Result:

Matplotlib Pie Charts
Creating Pie Charts
With Pyplot, you can use the pie() function to draw pie charts:

ExampleGet your own Python Server
A simple pie chart:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```
Result:

As you can see the pie chart draws one piece (called a wedge) for each value in the array (in this case [35, 25, 25, 15]).

By default the plotting of the first wedge starts from the x-axis and moves counterclockwise:

Note: The size of each wedge is determined by comparing the value with all the other values, by using this formula:

The value divided by the sum of all values: x/sum(x)

Labels
Add labels to the pie chart with the label parameter.

The label parameter must be an array with one label for each wedge:

Example
A simple pie chart:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.show()
```

Result:

Start Angle
As mentioned the default start angle is at the x-axis, but you can change the start angle by specifying a startangle parameter.

The startangle parameter is defined with an angle in degrees, default angle is 0:

Example
Start the first wedge at 90 degrees:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels, startangle = 90)
plt.show()
```

Result:

Explode
Maybe you want one of the wedges to stand out? The explode parameter allows you to do that.

The explode parameter, if specified, and not None, must be an array with one value for each wedge.

Each value represents how far from the center each wedge is displayed:

Example
Pull the "Apples" wedge 0.2 from the center of the pie:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [0.2, 0, 0, 0]

plt.pie(y, labels = mylabels, explode = myexplode)
plt.show()
```

Result:

Shadow
Add a shadow to the pie chart by setting the shadows parameter to True:

Example
Add a shadow:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [0.2, 0, 0, 0]

plt.pie(y, labels = mylabels, explode = myexplode, shadow = True)
plt.show()
```

Result:

Colors
You can set the color of each wedge with the colors parameter.

The colors parameter, if specified, must be an array with one value for each wedge:

Example
Specify a new color for each wedge:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
mycolors = ["black", "hotpink", "b", "#4CAF50"]

plt.pie(y, labels = mylabels, colors = mycolors)
plt.show()
```

Result:

You can use Hexadecimal color values, any of the 140 supported color names, or one of these shortcuts:

'r' - Red
'g' - Green
'b' - Blue
'c' - Cyan
'm' - Magenta
'y' - Yellow
'k' - Black

'w' - White

Legend
To add a list of explanation for each wedge, use the legend() function:

Example
Add a legend:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.legend()
plt.show()
```
Result:

Legend With Header
To add a header to the legend, add the title parameter to the legend function.

Example
Add a legend with a header:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.legend(title = "Four Fruits:")
plt.show()
```
Result:

Machine Learning
Machine Learning is making the computer learn from studying data and statistics.

Machine Learning is a step into the direction of artificial intelligence (AI).

Machine Learning is a program that analyses data and learns to predict the outcome.

Where To Start?
In this tutorial we will go back to mathematics and study statistics, and how to calculate important numbers based on data sets.

We will also learn how to use various Python modules to get the answers we need.

And we will learn how to make functions that are able to predict the outcome based on what we have learned.

Data Set
In the mind of a computer, a data set is any collection of data. It can be anything from an array to a complete database.

Example of an array:

[99,86,87,88,111,86,103,87,94,78,77,85,86]

Example of a database:

Carname	Color	Age	Speed	AutoPass
BMW	red	5	99	Y
Volvo	black	7	86	Y
VW	gray	8	87	N
VW	white	7	88	Y
Ford	white	2	111	Y
VW	white	17	86	Y
Tesla	red	2	103	Y
BMW	black	9	87	Y
Volvo	gray	4	94	N
Ford	white	11	78	N
Toyota	gray	12	77	N
VW	white	9	85	N
Toyota	blue	6	86	Y

By looking at the array, we can guess that the average value is probably around 80 or 90, and we are also able to determine the highest value and the lowest value, but what else can we do?

And by looking at the database we can see that the most popular color is white, and the oldest car is 17 years, but what if we could predict if a car had an AutoPass, just by looking at the other values?

That is what Machine Learning is for! Analyzing data and predicting the outcome!

In Machine Learning it is common to work with very large data sets. In this tutorial we will try to make it as easy as possible to understand the different concepts of machine learning, and we will work with small easy-to-understand data sets.

Data Types

To analyze data, it is important to know what type of data we are dealing with.

We can split the data types into three main categories:

Numerical
Categorical
Ordinal

Numerical data are numbers, and can be split into two numerical categories:

Discrete Data
- numbers that are limited to integers. Example: The number of cars passing by.

Continuous Data
- numbers that are of infinite value. Example: The price of an item, or the size of an item

Categorical data are values that cannot be measured up against each other. Example: a color value, or any yes/no values.

Ordinal data are like categorical data, but can be measured up against each other. Example: school grades where A is better than B and so on.

By knowing the data type of your data source, you will be able to know what technique to use when analyzing them.

You will learn more about statistics and analyzing data in the next chapters.

Machine Learning - Mean Median Mode

Mean, Median, and Mode

What can we learn from looking at a group of numbers?

In Machine Learning (and in mathematics) there are often three values that interests us:

Mean - The average value
Median - The mid point value
Mode - The most common value
Example: We have registered the speed of 13 cars:

speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]

What is the average, the middle, or the most common speed value?

Mean
The mean value is the average value.

To calculate the mean, find the sum of all values, and divide the sum by the number of values:

(99+86+87+88+111+86+103+87+94+78+77+85+86) / 13 = 89.77

The NumPy module has a method for this. Learn about the NumPy module in our NumPy Tutorial.

ExampleGet your own Python Server
Use the NumPy mean() method to find the average speed:

```
import numpy

speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]

x = numpy.mean(speed)

print(x)
```


Median
The median value is the value in the middle, after you have sorted all the values:

77, 78, 85, 86, 86, 86, 87, 87, 88, 94, 99, 103, 111

It is important that the numbers are sorted before you can find the median.

The NumPy module has a method for this:

Example
Use the NumPy median() method to find the middle value:

```
import numpy

speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]

x = numpy.median(speed)

print(x)
```

If there are two numbers in the middle, divide the sum of those numbers by two.

77, 78, 85, 86, 86, 86, 87, 87, 94, 98, 99, 103

(86 + 87) / 2 = 86.5

Example
Using the NumPy module:

```
import numpy

speed = [99,86,87,88,86,103,87,94,78,77,85,86]

x = numpy.median(speed)

print(x)
```

Mode

The Mode value is the value that appears the most number of times:

99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86 = 86

The SciPy module has a method for this. Learn about the SciPy module in our SciPy Tutorial.

Example

Use the SciPy mode() method to find the number that appears the most:

```
from scipy import stats

speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]

x = stats.mode(speed)

print(x)
```

Chapter Summary

The Mean, Median, and Mode are techniques that are often used in Machine Learning, so it is important to understand the concept behind them.

Machine Learning - Standard Deviation

What is Standard Deviation?

Standard deviation is a number that describes how spread out the values are.

A low standard deviation means that most of the numbers are close to the mean (average) value.

A high standard deviation means that the values are spread out over a wider range.

Example: This time we have registered the speed of 7 cars:

speed = [86,87,88,86,87,85,86]

The standard deviation is:

0.9

Meaning that most of the values are within the range of 0.9 from the mean value, which is 86.4.

Let us do the same with a selection of numbers with a wider range:

speed = [32,111,138,28,59,77,97]

The standard deviation is:

37.85

Meaning that most of the values are within the range of 37.85 from the mean value, which is 77.4.

As you can see, a higher standard deviation indicates that the values are spread out over a wider range.

The NumPy module has a method to calculate the standard deviation:

ExampleGet your own Python Server
Use the NumPy std() method to find the standard deviation:

```
import numpy

speed = [86,87,88,86,87,85,86]

x = numpy.std(speed)

print(x)
```

Example

```
import numpy

speed = [32,111,138,28,59,77,97]

x = numpy.std(speed)

print(x)
```

Learn to Filter Data in Python Like a Data Analyst

Try a hands-on training sessions with step-by-step guidance from an expert. Try the guided project made in collaboration with Coursera now!

Variance
Variance is another number that indicates how spread out the values are.

In fact, if you take the square root of the variance, you get the standard deviation!

Or the other way around, if you multiply the standard deviation by itself, you get the variance!

To calculate the variance you have to do as follows:

1. Find the mean:

(32+111+138+28+59+77+97) / 7 = 77.4

2. For each value: find the difference from the mean:

```
 32 - 77.4 = -45.4
111 - 77.4 =  33.6
138 - 77.4 =  60.6
 28 - 77.4 = -49.4
 59 - 77.4 = -18.4
 77 - 77.4 = - 0.4
 97 - 77.4 =  19.6
```

3. For each difference: find the square value:

```
(-45.4)2 = 2061.16
 (33.6)2 = 1128.96
 (60.6)2 = 3672.36
(-49.4)2 = 2440.36
(-18.4)2 =  338.56
(- 0.4)2 =    0.16
 (19.6)2 =  384.16
```

4. The variance is the average number of these squared differences:

(2061.16+1128.96+3672.36+2440.36+338.56+0.16+384.16) / 7 = 1432.2

Luckily, NumPy has a method to calculate the variance:

Example
Use the NumPy var() method to find the variance:

```
import numpy

speed = [32,111,138,28,59,77,97]

x = numpy.var(speed)

print(x)
```

Standard Deviation
As we have learned, the formula to find the standard deviation is the square root of the variance:

â1432.25 = 37.85

Or, as in the example from before, use the NumPy to calculate the standard deviation:

Example
Use the NumPy std() method to find the standard deviation:

```
import numpy

speed = [32,111,138,28,59,77,97]

x = numpy.std(speed)

print(x)
```

Symbols
Standard Deviation is often represented by the symbol Sigma: Ï

Variance is often represented by the symbol Sigma Squared: Ï2

Chapter Summary
The Standard Deviation and Variance are terms that are often used in Machine Learning, so it is important to understand how to get them, and the concept behind them.

Machine Learning - Percentiles
What are Percentiles?
Percentiles are used in statistics to give you a number that describes the value that a given percent of the values are lower than.

Example: Let's say we have an array of the ages of all the people that live in a street.

ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]

What is the 75. percentile? The answer is 43, meaning that 75% of the people are 43 or younger.

The NumPy module has a method for finding the specified percentile:

ExampleGet your own Python Server
Use the NumPy percentile() method to find the percentiles:

```
import numpy

ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]

x = numpy.percentile(ages, 75)

print(x)
```

Example

What is the age that 90% of the people are younger than?

```
import numpy

ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]

x = numpy.percentile(ages, 90)

print(x)
```

Machine Learning - Data Distribution

Data Distribution

Earlier in this tutorial we have worked with very small amounts of data in our examples, just to understand the different concepts.

In the real world, the data sets are much bigger, but it can be difficult to gather real world data, at least at an early stage of a project.

How Can we Get Big Data Sets?

To create big data sets for testing, we use the Python module NumPy, which comes with a number of methods to create random data sets, of any size.

ExampleGet your own Python Server

Create an array containing 250 random floats between 0 and 5:

```
import numpy

x = numpy.random.uniform(0.0, 5.0, 250)

print(x)
```

Histogram

To visualize the data set we can draw a histogram with the data we collected.

We will use the Python module Matplotlib to draw a histogram.

Learn about the Matplotlib module in our Matplotlib Tutorial.

Example

Draw a histogram:

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.uniform(0.0, 5.0, 250)

plt.hist(x, 5)
plt.show()
```

Result:

Histogram Explained

We use the array from the example above to draw a histogram with 5 bars.

The first bar represents how many values in the array are between 0 and 1.

The second bar represents how many values are between 1 and 2.

Etc.

Which gives us this result:

52 values are between 0 and 1
48 values are between 1 and 2
49 values are between 2 and 3
51 values are between 3 and 4
50 values are between 4 and 5
Note: The array values are random numbers and will not show the exact same result on your computer.

Big Data Distributions
An array containing 250 values is not considered very big, but now you know how to create a random set of values, and by changing the parameters, you can create the data set as big as you want.

Example
Create an array with 100000 random numbers, and display them using a histogram with 100 bars:

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.uniform(0.0, 5.0, 100000)

plt.hist(x, 100)
plt.show()
```

Machine Learning - Normal Data Distribution
Normal Data Distribution
In the previous chapter we learned how to create a completely random array, of a given size, and between two given values.

In this chapter we will learn how to create an array where the values are concentrated around a given value.

In probability theory this kind of data distribution is known as the normal data distribution, or the Gaussian data distribution, after the mathematician Carl Friedrich Gauss who came up with the formula of this data distribution.

ExampleGet your own Python Server
A typical normal data distribution:

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.normal(5.0, 1.0, 100000)

plt.hist(x, 100)
plt.show()
```
Result:

Note: A normal distribution graph is also known as the bell curve because of it's characteristic shape of a bell.

Histogram Explained

We use the array from the numpy.random.normal() method, with 100000 values, to draw a histogram with 100 bars.

We specify that the mean value is 5.0, and the standard deviation is 1.0.

Meaning that the values should be concentrated around 5.0, and rarely further away than 1.0 from the mean.

And as you can see from the histogram, most values are between 4.0 and 6.0, with a top at approximately 5.0.

Machine Learning - Scatter Plot
Scatter Plot
A scatter plot is a diagram where each value in the data set is represented by a dot.

The Matplotlib module has a method for drawing scatter plots, it needs two arrays of the same length, one for the values of the x-axis, and one for the values of the y-axis:

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]

y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

The x array represents the age of each car.

The y array represents the speed of each car.

ExampleGet your own Python Server
Use the scatter() method to draw a scatter plot diagram:

```
import matplotlib.pyplot as plt

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(x, y)
plt.show()
```
Result:

Scatter Plot Explained
The x-axis represents ages, and the y-axis represents speeds.

What we can read from the diagram is that the two fastest cars were both 2 years old, and the slowest car was 12 years old.

Note: It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

Random Data Distributions
In Machine Learning the data sets can contain thousands-, or even millions, of values.

You might not have real world data when you are testing an algorithm, you might have to use randomly generated values.

As we have learned in the previous chapter, the NumPy module can help us with that!

Let us create two arrays that are both filled with 1000 random numbers from a normal data distribution.

The first array will have the mean set to 5.0 with a standard deviation of 1.0.

The second array will have the mean set to 10.0 with a standard deviation of 2.0:

Example
A scatter plot with 1000 dots:

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.normal(5.0, 1.0, 1000)
y = numpy.random.normal(10.0, 2.0, 1000)

plt.scatter(x, y)
plt.show()
```
Result:

Scatter Plot Explained
We can see that the dots are concentrated around the value 5 on the x-axis, and 10 on the y-axis.

We can also see that the spread is wider on the y-axis than on the x-axis.

Machine Learning - Linear Regression
Regression
The term regression is used when you try to find the relationship between variables.

In Machine Learning, and in statistical modeling, that relationship is used to predict the outcome of future events.

Linear Regression
Linear regression uses the relationship between the data-points to draw a straight line through all them.

This line can be used to predict future values.

In Machine Learning, predicting the future is very important.

How Does it Work?
Python has methods for finding a relationship between data-points and to draw a line of linear regression. We will show you how to use these methods instead of going through the mathematic formula.

In the example below, the x-axis represents age, and the y-axis represents speed. We have registered the age and speed of 13 cars as they were passing a tollbooth. Let us see if the data we collected could be used in a linear regression:

ExampleGet your own Python Server
Start by drawing a scatter plot:

```
import matplotlib.pyplot as plt

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
```

```
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(x, y)
plt.show()
```

Result:

Example

Import scipy and draw the line of Linear Regression:

```
import matplotlib.pyplot as plt
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

Result:

Example Explained

Import the modules you need.

You can learn about the Matplotlib module in our Matplotlib Tutorial.

You can learn about the SciPy module in our SciPy Tutorial.

```
import matplotlib.pyplot as plt
from scipy import stats
```

Create the arrays that represent the values of the x and y axis:

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

Execute a method that returns some important key values of Linear Regression:

```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

Create a function that uses the slope and intercept values to return a new value. This new value represents where on the y-axis the corresponding x value will be placed:

```
def myfunc(x):
  return slope * x + intercept
```

Run each value of the x array through the function. This will result in a new array with new values for the y-axis:

```
mymodel = list(map(myfunc, x))
```

Draw the original scatter plot:

```
plt.scatter(x, y)
```

Draw the line of linear regression:

```
plt.plot(x, mymodel)
```

Display the diagram:

```
plt.show()
```


R for Relationship
It is important to know how the relationship between the values of the x-axis and the values of the y-axis is, if there are no relationship the linear regression can not be used to predict anything.

This relationship - the coefficient of correlation - is called r.

The r value ranges from -1 to 1, where 0 means no relationship, and 1 (and -1) means 100% related.

Python and the Scipy module will compute this value for you, all you have to do is feed it with the x and y values.

Example
How well does my data fit in a linear regression?

```
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

print(r)
```

Note: The result -0.76 shows that there is a relationship, not perfect, but it indicates that we could use linear regression in future predictions.

Predict Future Values
Now we can use the information we have gathered to predict future values.

Example: Let us try to predict the speed of a 10 years old car.

To do so, we need the same myfunc() function from the example above:

```
def myfunc(x):
  return slope * x + intercept
```

Example
Predict the speed of a 10 years old car:

```
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

speed = myfunc(10)

print(speed)
```

The example predicted a speed at 85.6, which we also could read from the diagram:

Bad Fit?
Let us create an example where linear regression would not be the best method to predict future values.

Example
These values for the x- and y-axis should result in a very bad fit for linear regression:

```
import matplotlib.pyplot as plt
from scipy import stats

x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

Result:

And the r for relationship?

Example
You should get a very low r value.

```
import numpy
from scipy import stats

x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]

slope, intercept, r, p, std_err = stats.linregress(x, y)

print(r)
```

The result: 0.013 indicates a very bad relationship, and tells us that this data set is not suitable for linear regression.

Machine Learning - Polynomial Regression
Polynomial Regression
If your data points clearly will not fit a linear regression (a straight line through all data points), it might be ideal for polynomial regression.

Polynomial regression, like linear regression, uses the relationship between the variables x and y to find the best way to draw a line through the data points.

How Does it Work?
Python has methods for finding a relationship between data-points and to draw a line of polynomial regression. We will show you how to use these methods instead of going through the mathematic formula.

In the example below, we have registered 18 cars as they were passing a certain tollbooth.

We have registered the car's speed, and the time of day (hour) the passing occurred.

The x-axis represents the hours of the day and the y-axis represents the speed:

ExampleGet your own Python Server
Start by drawing a scatter plot:

```
import matplotlib.pyplot as plt

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

plt.scatter(x, y)
plt.show()
```

Result:

Example
Import numpy and matplotlib then draw the line of Polynomial Regression:

```
import numpy
import matplotlib.pyplot as plt

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

myline = numpy.linspace(1, 22, 100)

plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Result:

Example Explained
Import the modules you need.

You can learn about the NumPy module in our NumPy Tutorial.

You can learn about the SciPy module in our SciPy Tutorial.

```
import numpy
import matplotlib.pyplot as plt
```

Create the arrays that represent the values of the x and y axis:

```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
```

NumPy has a method that lets us make a polynomial model:

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

Then specify how the line will display, we start at position 1, and end at position 22:

```
myline = numpy.linspace(1, 22, 100)
```

Draw the original scatter plot:

```
plt.scatter(x, y)
```

Draw the line of polynomial regression:

```
plt.plot(myline, mymodel(myline))
```

Display the diagram:

```
plt.show()
```


R-Squared

It is important to know how well the relationship between the values of the x- and y-axis is, if there are no relationship the polynomial regression can not be used to predict anything.

The relationship is measured with a value called the r-squared.

The r-squared value ranges from 0 to 1, where 0 means no relationship, and 1 means 100% related.

Python and the Sklearn module will compute this value for you, all you have to do is feed it with the x and y arrays:

Example
How well does my data fit in a polynomial regression?

```
import numpy
from sklearn.metrics import r2_score

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

print(r2_score(y, mymodel(x)))
```

Note: The result 0.94 shows that there is a very good relationship, and we can use polynomial regression in future predictions.

Predict Future Values

Now we can use the information we have gathered to predict future values.

Example: Let us try to predict the speed of a car that passes the tollbooth at around the time 17:00:

To do so, we need the same mymodel array from the example above:

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

Example
Predict the speed of a car passing at 17:00:

```
import numpy
from sklearn.metrics import r2_score

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

speed = mymodel(17)
print(speed)
```

The example predicted a speed to be 88.87, which we also could read from the diagram:

Bad Fit?

Let us create an example where polynomial regression would not be the best method to predict future values.

Example

These values for the x- and y-axis should result in a very bad fit for polynomial regression:

```
import numpy
import matplotlib.pyplot as plt

x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

myline = numpy.linspace(2, 95, 100)

plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Result:

And the r-squared value?

Example

You should get a very low r-squared value.

```
import numpy
from sklearn.metrics import r2_score

x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

print(r2_score(y, mymodel(x)))
```

The result: 0.00995 indicates a very bad relationship, and tells us that this data set is not suitable for polynomial regression.

Machine Learning - Multiple Regression

Multiple Regression

Multiple regression is like linear regression, but with more than one independent value, meaning that we try to predict a value based on two or more variables.

Take a look at the data set below, it contains some information about cars.

Car	Model	Volume	Weight	CO2
Toyota	Aygo	1000	790	99
Mitsubishi	Space Star	1200	1160	95
Skoda	Citigo	1000	929	95
Fiat	500	900	865	90
Mini	Cooper	1500	1140	105
VW	Up!	1000	929	105
Skoda	Fabia	1400	1109	90
Mercedes	A-Class	1500	1365	92
Ford	Fiesta	1500	1112	98
Audi	A1	1600	1150	99

Hyundai	I20	1100	980	99
Suzuki	Swift	1300	990	101
Ford	Fiesta	1000	1112	99
Honda	Civic	1600	1252	94
Hundai	I30	1600	1326	97
Opel	Astra	1600	1330	97
BMW	1	1600	1365	99
Mazda	3	2200	1280	104
Skoda	Rapid	1600	1119	104
Ford	Focus	2000	1328	105
Ford	Mondeo	1600	1584	94
Opel	Insignia	2000	1428	99
Mercedes	C-Class	2100	1365	99
Skoda	Octavia	1600	1415	99
Volvo	S60	2000	1415	99
Mercedes	CLA	1500	1465	102
Audi	A4	2000	1490	104
Audi	A6	2000	1725	114
Volvo	V70	1600	1523	109
BMW	5	2000	1705	114
Mercedes	E-Class	2100	1605	115
Volvo	XC70	2000	1746	117
Ford	B-Max	1600	1235	104
BMW	2	1600	1390	108
Opel	Zafira	1600	1405	109
Mercedes	SLK	2500	1395	120

We can predict the CO2 emission of a car based on the size of the engine, but with multiple regression we can throw in more variables, like the weight of the car, to make the prediction more accurate.

How Does it Work?

In Python we have modules that will do the work for us. Start by importing the Pandas module.

```
import pandas
```

Learn about the Pandas module in our Pandas Tutorial.

The Pandas module allows us to read csv files and return a DataFrame object.

The file is meant for testing purposes only, you can download it here: data.csv

```
df = pandas.read_csv("data.csv")
```

Then make a list of the independent values and call this variable X.

Put the dependent values in a variable called y.

```
X = df[['Weight', 'Volume']]
y = df['CO2']
```

Tip: It is common to name the list of independent values with a upper case X, and the list of dependent values with a lower case y.

We will use some methods from the sklearn module, so we will have to import that module as well:

```
from sklearn import linear_model
```

From the sklearn module we will use the LinearRegression() method to create a linear regression object.

This object has a method called fit() that takes the independent and dependent values as parameters and fills the regression object with data that describes the relationship:

```
regr = linear_model.LinearRegression()
regr.fit(X, y)
```

Now we have a regression object that are ready to predict CO2 values based on a car's weight and volume:

```
#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:
predictedCO2 = regr.predict([[2300, 1300]])
```

ExampleGet your own Python Server
See the whole example in action:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:
predictedCO2 = regr.predict([[2300, 1300]])

print(predictedCO2)
```

Result:
[107.2087328]

We have predicted that a car with 1.3 liter engine, and a weight of 2300 kg, will release approximately 107 grams of CO2 for every kilometer it drives.

Coefficient
The coefficient is a factor that describes the relationship with an unknown variable.

Example: if x is a variable, then 2x is x two times. x is the unknown variable, and the number 2 is the coefficient.

In this case, we can ask for the coefficient value of weight against CO2, and for volume against CO2. The answer(s) we get tells us what would happen if we increase, or decrease, one of the independent values.

Example
Print the coefficient values of the regression object:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)
```

```
print(regr.coef_)
```

Result:

[0.00755095 0.00780526]

Result Explained

The result array represents the coefficient values of weight and volume.

Weight: 0.00755095
Volume: 0.00780526

These values tell us that if the weight increase by 1kg, the CO2 emission increases by 0.00755095g.

And if the engine size (Volume) increases by 1 cm3, the CO2 emission increases by 0.00780526 g.

I think that is a fair guess, but let test it!

We have already predicted that if a car with a 1300cm3 engine weighs 2300kg, the CO2 emission will be approximately 107g.

What if we increase the weight with 1000kg?

Example

Copy the example from before, but change the weight from 2300 to 3300:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

predictedCO2 = regr.predict([[3300, 1300]])

print(predictedCO2)
```

Result:

[114.75968007]

We have predicted that a car with 1.3 liter engine, and a weight of 3300 kg, will release approximately 115 grams of CO2 for every kilometer it drives.

Which shows that the coefficient of 0.00755095 is correct:

107.2087328 + (1000 * 0.00755095) = 114.75968

Machine Learning - Scale

Scale Features

When your data has different values, and even different measurement units, it can be difficult to compare them. What is kilograms compared to meters? Or altitude compared to time?

The answer to this problem is scaling. We can scale data into new values that are easier to compare.

Take a look at the table below, it is the same data set that we used in the multiple regression chapter, but this time the volume column contains values in liters instead of cm3 (1.0 instead of 1000).

Car	Model	Volume	Weight	CO2
Toyota	Aygo	1.0	790	99
Mitsubishi	Space Star	1.2	1160	95
Skoda	Citigo	1.0	929	95
Fiat	500	0.9	865	90
Mini	Cooper	1.5	1140	105
VW	Up!	1.0	929	105
Skoda	Fabia	1.4	1109	90
Mercedes	A-Class	1.5	1365	92
Ford	Fiesta	1.5	1112	98
Audi	A1	1.6	1150	99
Hyundai	I20	1.1	980	99
Suzuki	Swift	1.3	990	101
Ford	Fiesta	1.0	1112	99
Honda	Civic	1.6	1252	94
Hundai	I30	1.6	1326	97
Opel	Astra	1.6	1330	97
BMW	1	1.6	1365	99
Mazda	3	2.2	1280	104
Skoda	Rapid	1.6	1119	104
Ford	Focus	2.0	1328	105
Ford	Mondeo	1.6	1584	94
Opel	Insignia	2.0	1428	99
Mercedes	C-Class	2.1	1365	99
Skoda	Octavia	1.6	1415	99
Volvo	S60	2.0	1415	99
Mercedes	CLA	1.5	1465	102
Audi	A4	2.0	1490	104
Audi	A6	2.0	1725	114
Volvo	V70	1.6	1523	109
BMW	5	2.0	1705	114
Mercedes	E-Class	2.1	1605	115
Volvo	XC70	2.0	1746	117
Ford	B-Max	1.6	1235	104
BMW	2	1.6	1390	108
Opel	Zafira	1.6	1405	109
Mercedes	SLK	2.5	1395	120

It can be difficult to compare the volume 1.0 with the weight 790, but if we scale them both into comparable values, we can easily see how much one value is compared to the other.

There are different methods for scaling data, in this tutorial we will use a method called standardization.

The standardization method uses this formula:

z = (x - u) / s

Where z is the new value, x is the original value, u is the mean and s is the standard deviation.

If you take the weight column from the data set above, the first value is 790, and the scaled value will be:

(790 - 1292.23) / 238.74 = -2.1

If you take the volume column from the data set above, the first value is 1.0, and the scaled value will be:

(1.0 - 1.61) / 0.38 = -1.59

Now you can compare -2.1 with -1.59 instead of comparing 790 with 1.0.

You do not have to do this manually, the Python sklearn module has a method called StandardScaler() which returns a Scaler object with methods for transforming data sets.

ExampleGet your own Python Server
Scale all values in the Weight and Volume columns:

```
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]

scaledX = scale.fit_transform(X)

print(scaledX)
```

Result:
Note that the first two values are -2.1 and -1.59, which corresponds to our calculations:

```
[[-2.10389253 -1.59336644]
 [-0.55407235 -1.07190106]
 [-1.52166278 -1.59336644]
 [-1.78973979 -1.85409913]
 [-0.63784641 -0.28970299]
 [-1.52166278 -1.59336644]
 [-0.76769621 -0.55043568]
 [ 0.3046118  -0.28970299]
 [-0.7551301  -0.28970299]
 [-0.59595938 -0.0289703 ]
 [-1.30803892 -1.33263375]
 [-1.26615189 -0.81116837]
 [-0.7551301  -1.59336644]
 [-0.16871166 -0.0289703 ]
 [ 0.14125238 -0.0289703 ]
 [ 0.15800719 -0.0289703 ]
 [ 0.3046118  -0.0289703 ]
 [-0.05142797  1.53542584]
 [-0.72580918 -0.0289703 ]
 [ 0.14962979  1.01396046]
 [ 1.2219378  -0.0289703 ]
 [ 0.5685001   1.01396046]
 [ 0.3046118   1.27469315]
 [ 0.51404696 -0.0289703 ]
 [ 0.51404696  1.01396046]
 [ 0.72348212 -0.28970299]
 [ 0.8281997   1.01396046]
 [ 1.81254495  1.01396046]
 [ 0.96642691 -0.0289703 ]
 [ 1.72877089  1.01396046]
 [ 1.30990057  1.27469315]
 [ 1.90050772  1.01396046]
 [-0.23991961 -0.0289703 ]
 [ 0.40932938 -0.0289703 ]
 [ 0.47215993 -0.0289703 ]
 [ 0.4302729   2.31762392]]
```


Predict CO2 Values
The task in the Multiple Regression chapter was to predict the CO2 emission from a car when you only knew its weight and volume.

When the data set is scaled, you will have to use the scale when you predict values:

Example
Predict the CO2 emission from a 1.3 liter car that weighs 2300 kilograms:

```
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

scaledX = scale.fit_transform(X)

regr = linear_model.LinearRegression()
regr.fit(scaledX, y)

scaled = scale.transform([[2300, 1.3]])

predictedCO2 = regr.predict([scaled[0]])
print(predictedCO2)
```

Result:
[107.2087328]

Machine Learning - Train/Test
Evaluate Your Model
In Machine Learning we create models to predict the outcome of certain events, like in the previous chapter where we predicted the CO2 emission of a car when we knew the weight and engine size.

To measure if the model is good enough, we can use a method called Train/Test.

What is Train/Test
Train/Test is a method to measure the accuracy of your model.

It is called Train/Test because you split the data set into two sets: a training set and a testing set.

80% for training, and 20% for testing.

You train the model using the training set.

You test the model using the testing set.

Train the model means create the model.

Test the model means test the accuracy of the model.

Start With a Data Set
Start with a data set you want to test.

Our data set illustrates 100 customers in a shop, and their shopping habits.

ExampleGet your own Python Server

```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)
```

```
x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

plt.scatter(x, y)
plt.show()
```
Result:
The x axis represents the number of minutes before making a purchase.

The y axis represents the amount of money spent on the purchase.

Split Into Train/Test
The training set should be a random selection of 80% of the original data.

The testing set should be the remaining 20%.

```
train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]
```

Display the Training Set
Display the same scatter plot with the training set:

Example
```
plt.scatter(train_x, train_y)
plt.show()
```
Result:
It looks like the original data set, so it seems to be a fair selection:

Display the Testing Set
To make sure the testing set is not completely different, we will take a look at the testing set as well.

Example
```
plt.scatter(test_x, test_y)
plt.show()
```
Result:
The testing set also looks like the original data set:

Fit the Data Set
What does the data set look like? In my opinion I think the best fit would be a polynomial regression, so let us draw a line of polynomial regression.

To draw a line through the data points, we use the plot() method of the matplotlib module:

Example
Draw a polynomial regression line through the data points:

```
import numpy
import matplotlib.pyplot as plt
```

```
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

myline = numpy.linspace(0, 6, 100)

plt.scatter(train_x, train_y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Result:

The result can back my suggestion of the data set fitting a polynomial regression, even though it would give us some weird results if we try to predict values outside of the data set. Example: the line indicates that a customer spending 6 minutes in the shop would make a purchase worth 200. That is probably a sign of overfitting.

But what about the R-squared score? The R-squared score is a good indicator of how well my data set is fitting the model.

R2

Remember R2, also known as R-squared?

It measures the relationship between the x axis and the y axis, and the value ranges from 0 to 1, where 0 means no relationship, and 1 means totally related.

The sklearn module has a method called r2_score() that will help us find this relationship.

In this case we would like to measure the relationship between the minutes a customer stays in the shop and how much money they spend.

Example

How well does my training data fit in a polynomial regression?

```
import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

r2 = r2_score(train_y, mymodel(train_x))

print(r2)
```

Note: The result 0.799 shows that there is a OK relationship.

Bring in the Testing Set
Now we have made a model that is OK, at least when it comes to training data.

Now we want to test the model with the testing data as well, to see if gives us the same result.

Example
Let us find the R2 score when using testing data:

```
import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

r2 = r2_score(test_y, mymodel(test_x))

print(r2)
```

Note: The result 0.809 shows that the model fits the testing set as well, and we are confident that we can use the model to predict future values.

Predict Values
Now that we have established that our model is OK, we can start predicting new values.

Example
How much money will a buying customer spend, if she or he stays in the shop for 5 minutes?

```
print(mymodel(5))
```

The example predicted the customer to spend 22.88 dollars, as seems to correspond to the diagram:

Machine Learning - Decision Tree

Decision Tree
In this chapter we will show you how to make a "Decision Tree". A Decision Tree is a Flow Chart, and can help you make decisions based on previous experience.

In the example, a person will try to decide if he/she should go to a comedy show or not.

Luckily our example person has registered every time there was a comedy show in town, and registered some information about the comedian, and also registered if he/she went or not.

Age	Experience	Rank	Nationality	Go
36	10	9	UK	NO
42	12	4	USA	NO
23	4	6	N	NO

52	4	4	USA	NO
43	21	8	USA	YES
44	14	5	UK	NO
66	3	7	N	YES
35	14	9	UK	YES
52	13	7	N	YES
35	5	9	N	YES
24	3	5	USA	NO
18	3	7	UK	YES
45	9	9	UK	YES

Now, based on this data set, Python can create a decision tree that can be used to decide if any new shows are worth attending to.

How Does it Work?
First, read the dataset with pandas:

ExampleGet your own Python Server
Read and print the data set:

```
import pandas

df = pandas.read_csv("data.csv")

print(df)
```

To make a decision tree, all data has to be numerical.

We have to convert the non numerical columns 'Nationality' and 'Go' into numerical values.

Pandas has a map() method that takes a dictionary with information on how to convert the values.

{'UK': 0, 'USA': 1, 'N': 2}

Means convert the values 'UK' to 0, 'USA' to 1, and 'N' to 2.

Example
Change string values into numerical values:

```
d = {'UK': 0, 'USA': 1, 'N': 2}
df['Nationality'] = df['Nationality'].map(d)
d = {'YES': 1, 'NO': 0}
df['Go'] = df['Go'].map(d)

print(df)
```

Then we have to separate the feature columns from the target column.

The feature columns are the columns that we try to predict from, and the target column is the column with the values we try to predict.

Example
X is the feature columns, y is the target column:

```
features = ['Age', 'Experience', 'Rank', 'Nationality']

X = df[features]
y = df['Go']

print(X)
print(y)
```

Now we can create the actual decision tree, fit it with our details. Start by importing the modules we need:

Example
Create and display a Decision Tree:

```
import pandas
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt

df = pandas.read_csv("data.csv")

d = {'UK': 0, 'USA': 1, 'N': 2}
df['Nationality'] = df['Nationality'].map(d)
d = {'YES': 1, 'NO': 0}
df['Go'] = df['Go'].map(d)

features = ['Age', 'Experience', 'Rank', 'Nationality']

X = df[features]
y = df['Go']

dtree = DecisionTreeClassifier()
dtree = dtree.fit(X, y)

tree.plot_tree(dtree, feature_names=features)
```

Result Explained
The decision tree uses your earlier decisions to calculate the odds for you to wanting to go see a comedian or not.

Let us read the different aspects of the decision tree:

Rank
Rank <= 6.5 means that every comedian with a rank of 6.5 or lower will follow the True arrow (to the left), and the rest will follow the False arrow (to the right).

gini = 0.497 refers to the quality of the split, and is always a number between 0.0 and 0.5, where 0.0 would mean all of the samples got the same result, and 0.5 would mean that the split is done exactly in the middle.

samples = 13 means that there are 13 comedians left at this point in the decision, which is all of them since this is the first step.

value = [6, 7] means that of these 13 comedians, 6 will get a "NO", and 7 will get a "GO".

Gini
There are many ways to split the samples, we use the GINI method in this tutorial.

The Gini method uses this formula:

Gini = 1 - (x/n)2 - (y/n)2

Where x is the number of positive answers("GO"), n is the number of samples, and y is the number of negative answers ("NO"), which gives us this calculation:

1 - (7 / 13)2 - (6 / 13)2 = 0.497

The next step contains two boxes, one box for the comedians with a 'Rank' of 6.5 or lower, and one box with the rest.

True - 5 Comedians End Here:
gini = 0.0 means all of the samples got the same result.

samples = 5 means that there are 5 comedians left in this branch (5 comedian with a Rank of 6.5 or lower).

value = [5, 0] means that 5 will get a "NO" and 0 will get a "GO".

False - 8 Comedians Continue:
Nationality
Nationality <= 0.5 means that the comedians with a nationality value of less than 0.5 will follow the arrow to the left (which means everyone from the UK,), and the rest will follow the arrow to the right.

gini = 0.219 means that about 22% of the samples would go in one direction.

samples = 8 means that there are 8 comedians left in this branch (8 comedian with a Rank higher than 6.5).

value = [1, 7] means that of these 8 comedians, 1 will get a "NO" and 7 will get a "GO".

True - 4 Comedians Continue:
Age
Age <= 35.5 means that comedians at the age of 35.5 or younger will follow the arrow to the left, and the rest will follow the arrow to the right.

gini = 0.375 means that about 37,5% of the samples would go in one direction.

samples = 4 means that there are 4 comedians left in this branch (4 comedians from the UK).

value = [1, 3] means that of these 4 comedians, 1 will get a "NO" and 3 will get a "GO".

False - 4 Comedians End Here:
gini = 0.0 means all of the samples got the same result.

samples = 4 means that there are 4 comedians left in this branch (4 comedians not from the UK).

value = [0, 4] means that of these 4 comedians, 0 will get a "NO" and 4 will get a "GO".

True - 2 Comedians End Here:
gini = 0.0 means all of the samples got the same result.

samples = 2 means that there are 2 comedians left in this branch (2 comedians at the age 35.5 or younger).

value = [0, 2] means that of these 2 comedians, 0 will get a "NO" and 2 will get a "GO".

False - 2 Comedians Continue:
Experience
Experience <= 9.5 means that comedians with 9.5 years of experience, or less, will follow the arrow to the left, and the rest will follow the arrow to the right.

gini = 0.5 means that 50% of the samples would go in one direction.

samples = 2 means that there are 2 comedians left in this branch (2 comedians older than 35.5).

value = [1, 1] means that of these 2 comedians, 1 will get a "NO" and 1 will get a "GO".

True - 1 Comedian Ends Here:
gini = 0.0 means all of the samples got the same result.

samples = 1 means that there is 1 comedian left in this branch (1 comedian with 9.5 years of experience or less).

value = [0, 1] means that 0 will get a "NO" and 1 will get a "GO".

False - 1 Comedian Ends Here:
gini = 0.0 means all of the samples got the same result.

samples = 1 means that there is 1 comedians left in this branch (1 comedian with more than 9.5 years of experience).

value = [1, 0] means that 1 will get a "NO" and 0 will get a "GO".

Predict Values
We can use the Decision Tree to predict new values.

Example: Should I go see a show starring a 40 years old American comedian, with 10 years of experience, and a comedy ranking of 7?

Example
Use predict() method to predict new values:

```
print(dtree.predict([[40, 10, 7, 1]]))
```

Example
What would the answer be if the comedy rank was 6?

```
print(dtree.predict([[40, 10, 6, 1]]))
```

Different Results
You will see that the Decision Tree gives you different results if you run it enough times, even if you feed it with the same data.

That is because the Decision Tree does not give us a 100% certain answer. It is based on the probability of an outcome, and the answer will vary.

Machine Learning - Confusion Matrix
On this page, W3schools.com collaborates with NYC Data Science Academy, to

deliver digital training content to our students.

What is a confusion matrix?
It is a table that is used in classification problems to assess where errors in the model were made.

The rows represent the actual classes the outcomes should have been. While the columns represent the predictions we have made. Using this table it is easy to see which predictions are wrong.

Creating a Confusion Matrix
Confusion matrixes can be created by predictions made from a logistic regression.

For now we will generate actual and predicted values by utilizing NumPy:

```
import numpy
```

Next we will need to generate the numbers for "actual" and "predicted" values.

```
actual = numpy.random.binomial(1, 0.9, size = 1000)
predicted = numpy.random.binomial(1, 0.9, size = 1000)
```

In order to create the confusion matrix we need to import metrics from the sklearn module.

```
from sklearn import metrics
```

Once metrics is imported we can use the confusion matrix function on our actual and predicted values.

```
confusion_matrix = metrics.confusion_matrix(actual, predicted)
```

To create a more interpretable visual display we need to convert the table into a confusion matrix display.

```
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = [False, True])
```

Vizualizing the display requires that we import pyplot from matplotlib.

```
import matplotlib.pyplot as plt
```

Finally to display the plot we can use the functions plot() and show() from pyplot.

```
cm_display.plot()
plt.show()
```

See the whole example in action:

ExampleGet your own Python Server
```
import matplotlib.pyplot as plt
import numpy
from sklearn import metrics

actual = numpy.random.binomial(1,.9,size = 1000)
predicted = numpy.random.binomial(1,.9,size = 1000)

confusion_matrix = metrics.confusion_matrix(actual, predicted)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = [False, True])
```

cm_display.plot()
plt.show()
Result

Results Explained
The Confusion Matrix created has four different quadrants:

False Negative (Top-Left Quadrant)
False Positive (Top-Right Quadrant)
True Negative (Bottom-Left Quadrant)
True Positive (Bottom-Right Quadrant)

True means that the values were accurately predicted, False means that there was an error or wrong prediction.

Now that we have made a Confusion Matrix, we can calculate different measures to quantify the quality of the model. First, lets look at Accuracy.

Created Metrics
The matrix provides us with many useful metrics that help us to evaluate out classification model.

The different measures include: Accuracy, Precision, Sensitivity (Recall), Specificity, and the F-score, explained below.

Accuracy
Accuracy measures how often the model is correct.

How to Calculate
(True Positive + True Negative) / Total Predictions

Example
Accuracy = metrics.accuracy_score(actual, predicted)
Precision
Of the positives predicted, what percentage is truly positive?

How to Calculate
True Positive / (True Positive + False Positive)

Precision does not evaluate the correctly predicted negative cases:

Example
Precision = metrics.precision_score(actual, predicted)
Sensitivity (Recall)
Of all the positive cases, what percentage are predicted positive?

Sensitivity (sometimes called Recall) measures how good the model is at predicting positives.

This means it looks at true positives and false negatives (which are positives that have been incorrectly predicted as negative).

How to Calculate
True Positive / (True Positive + False Negative)

Sensitivity is good at understanding how well the model predicts something is positive:

Example
Sensitivity_recall = metrics.recall_score(actual, predicted)

Specificity
How well the model is at prediciting negative results?

Specificity is similar to sensitivity, but looks at it from the persepctive of negative results.

How to Calculate
True Negative / (True Negative + False Positive)

Since it is just the opposite of Recall, we use the recall_score function, taking the opposite position label:

Example
```
Specificity = metrics.recall_score(actual, predicted, pos_label=0)
```
F-score
F-score is the "harmonic mean" of precision and sensitivity.

It considers both false positive and false negative cases and is good for imbalanced datasets.

How to Calculate
2 * ((Precision * Sensitivity) / (Precision + Sensitivity))

This score does not take into consideration the True Negative values:

Example
```
F1_score = metrics.f1_score(actual, predicted)
```
All calulations in one:

Example
```
#metrics
print({"Accuracy":Accuracy,"Precision":Precision,"Sensitivity_recall":Sensitivity_recall,"Specificity":Specificity,"F1_score":F1_score})
```

Machine Learning - Hierarchical Clustering
On this page, W3schools.com collaborates with NYC Data Science Academy, to deliver digital training content to our students.

Hierarchical Clustering
Hierarchical clustering is an unsupervised learning method for clustering data points. The algorithm builds clusters by measuring the dissimilarities between data. Unsupervised learning means that a model does not have to be trained, and we do not need a "target" variable. This method can be used on any data to visualize and interpret the relationship between individual data points.

Here we will use hierarchical clustering to group data points and visualize the clusters using both a dendrogram and scatter plot.

How does it work?
We will use Agglomerative Clustering, a type of hierarchical clustering that follows a bottom up approach. We begin by treating each data point as its own cluster. Then, we join clusters together that have the shortest distance between them to create larger clusters. This step is repeated until one large cluster is formed containing all of the data points.

Hierarchical clustering requires us to decide on both a distance and linkage method. We will use euclidean distance and the Ward linkage method, which attempts to minimize the variance between clusters.

ExampleGet your own Python Server
Start by visualizing some data points:

```
import numpy as np
import matplotlib.pyplot as plt

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

plt.scatter(x, y)
plt.show()
```

Result

Now we compute the ward linkage using euclidean distance, and visualize it using a dendrogram:

Example

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

data = list(zip(x, y))

linkage_data = linkage(data, method='ward', metric='euclidean')
dendrogram(linkage_data)

plt.show()
```

Result

Here, we do the same thing with Python's scikit-learn library. Then, visualize on a 2-dimensional plot:

Example

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

data = list(zip(x, y))

hierarchical_cluster = AgglomerativeClustering(n_clusters=2,
affinity='euclidean', linkage='ward')
labels = hierarchical_cluster.fit_predict(data)

plt.scatter(x, y, c=labels)
plt.show()
```

Result

Example Explained

Import the modules you need.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering
```

You can learn about the Matplotlib module in our "Matplotlib Tutorial.

You can learn about the SciPy module in our SciPy Tutorial.

NumPy is a library for working with arrays and matricies in Python, you can learn about the NumPy module in our NumPy Tutorial.

scikit-learn is a popular library for machine learning.

Create arrays that resemble two variables in a dataset. Note that while we only use two variables here, this method will work with any number of variables:

```
x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
```

Turn the data into a set of points:

```
data = list(zip(x, y))
print(data)
```

Result:

```
[(4, 21), (5, 19), (10, 24), (4, 17), (3, 16), (11, 25), (14, 24), (6, 22), (10, 21), (12, 21)]
```

Compute the linkage between all of the different points. Here we use a simple euclidean distance measure and Ward's linkage, which seeks to minimize the variance between clusters.

```
linkage_data = linkage(data, method='ward', metric='euclidean')
```

Finally, plot the results in a dendrogram. This plot will show us the hierarchy of clusters from the bottom (individual points) to the top (a single cluster consisting of all data points).

plt.show() lets us visualize the dendrogram instead of just the raw linkage data.

```
dendrogram(linkage_data)
plt.show()
```

Result:

The scikit-learn library allows us to use hierarchichal clustering in a different manner. First, we initialize the AgglomerativeClustering class with 2 clusters, using the same euclidean distance and Ward linkage.

```
hierarchical_cluster = AgglomerativeClustering(n_clusters=2,
affinity='euclidean', linkage='ward')
```

The .fit_predict method can be called on our data to compute the clusters using the defined parameters across our chosen number of clusters.

```
labels = hierarchical_cluster.fit_predict(data) print(labels)
```

Result:

```
[0 0 1 0 0 1 1 0 1 1]
```

Finally, if we plot the same data and color the points using the labels assigned to each index by the hierarchical clustering method, we can see the cluster each point was assigned to:

```
plt.scatter(x, y, c=labels)
plt.show()
```

Result:

Machine Learning - Logistic Regression
On this page, W3schools.com collaborates with NYC Data Science Academy, to deliver digital training content to our students.

Logistic Regression
Logistic regression aims to solve classification problems. It does this by predicting categorical outcomes, unlike linear regression that predicts a continuous outcome.

In the simplest case there are two outcomes, which is called binomial, an example of which is predicting if a tumor is malignant or benign. Other cases have more than two outcomes to classify, in this case it is called multinomial. A common example for multinomial logistic regression would be predicting the class of an iris flower between 3 different species.

Here we will be using basic logistic regression to predict a binomial variable. This means it has only two possible outcomes.

How does it work?
In Python we have modules that will do the work for us. Start by importing the NumPy module.

```
import numpy
```

Store the independent variables in X.

Store the dependent variable in y.

Below is a sample dataset:

```
#X represents the size of a tumor in centimeters.
X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52,
3.69, 5.88]).reshape(-1,1)

#Note: X has to be reshaped into a column from a row for the
LogisticRegression() function to work.
#y represents whether or not the tumor is cancerous (0 for "No", 1 for "Yes").
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

We will use a method from the sklearn module, so we will have to import that module as well:

```
from sklearn import linear_model
```

From the sklearn module we will use the LogisticRegression() method to create a logistic regression object.

This object has a method called fit() that takes the independent and dependent values as parameters and fills the regression object with data that describes the relationship:

```
logr = linear_model.LogisticRegression()
logr.fit(X,y)
```

Now we have a logistic regression object that is ready to whether a tumor is cancerous based on the tumor size:

```
#predict if tumor is cancerous where the size is 3.46mm:
predicted = logr.predict(numpy.array([3.46]).reshape(-1,1))
```

ExampleGet your own Python Server
See the whole example in action:

```
import numpy
from sklearn import linear_model

#Reshaped for Logistic function.
X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52,
3.69, 5.88]).reshape(-1,1)
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear_model.LogisticRegression()
logr.fit(X,y)

#predict if tumor is cancerous where the size is 3.46mm:
predicted = logr.predict(numpy.array([3.46]).reshape(-1,1))
print(predicted)
```

Result

```
 [0]
```

We have predicted that a tumor with a size of 3.46mm will not be cancerous.

Coefficient
In logistic regression the coefficient is the expected change in log-odds of having the outcome per unit change in X.

This does not have the most intuitive understanding so let's use it to create something that makes more sense, odds.

Example
See the whole example in action:

```
import numpy
from sklearn import linear_model

#Reshaped for Logistic function.
X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52,
3.69, 5.88]).reshape(-1,1)
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear_model.LogisticRegression()
logr.fit(X,y)

log_odds = logr.coef_
odds = numpy.exp(log_odds)

print(odds)
```

Result

```
 [4.03541657]
```

This tells us that as the size of a tumor increases by 1mm the odds of it being a cancerous tumor increases by 4x.

Probability

The coefficient and intercept values can be used to find the probability that each tumor is cancerous.

Create a function that uses the model's coefficient and intercept values to return a new value. This new value represents probability that the given observation is a tumor:

```
def logit2prob(logr,x):
  log_odds = logr.coef_ * x + logr.intercept_
  odds = numpy.exp(log_odds)
  probability = odds / (1 + odds)
  return(probability)
```

Function Explained
To find the log-odds for each observation, we must first create a formula that looks similar to the one from linear regression, extracting the coefficient and the intercept.

```
log_odds = logr.coef_ * x + logr.intercept_
```

To then convert the log-odds to odds we must exponentiate the log-odds.

```
odds = numpy.exp(log_odds)
```

Now that we have the odds, we can convert it to probability by dividing it by 1 plus the odds.

```
probability = odds / (1 + odds)
```

Let us now use the function with what we have learned to find out the probability that each tumor is cancerous.

Example
See the whole example in action:

```
import numpy
from sklearn import linear_model

X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52,
3.69, 5.88]).reshape(-1,1)
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear_model.LogisticRegression()
logr.fit(X,y)

def logit2prob(logr, X):
  log_odds = logr.coef_ * X + logr.intercept_
  odds = numpy.exp(log_odds)
  probability = odds / (1 + odds)
  return(probability)

print(logit2prob(logr, X))
```

Result

```
  [[0.60749955]
   [0.19268876]
   [0.12775886]
   [0.00955221]
   [0.08038616]
   [0.07345637]
   [0.88362743]
   [0.77901378]
   [0.88924409]
   [0.81293497]
```

```
 [0.57719129]
 [0.96664243]]
```

Results Explained
3.78 0.61 The probability that a tumor with the size 3.78cm is cancerous is 61%.

2.44 0.19 The probability that a tumor with the size 2.44cm is cancerous is 19%.

2.09 0.13 The probability that a tumor with the size 2.09cm is cancerous is 13%.

Machine Learning - Grid Search
On this page, W3schools.com collaborates with NYC Data Science Academy, to deliver digital training content to our students.

Grid Search
The majority of machine learning models contain parameters that can be adjusted to vary how the model learns. For example, the logistic regression model, from sklearn, has a parameter C that controls regularization,which affects the complexity of the model.

How do we pick the best value for C? The best value is dependent on the data used to train the model.

How does it work?
One method is to try out different values and then pick the value that gives the best score. This technique is known as a grid search. If we had to select the values for two or more parameters, we would evaluate all combinations of the sets of values thus forming a grid of values.

Before we get into the example it is good to know what the parameter we are changing does. Higher values of C tell the model, the training data resembles real world information, place a greater weight on the training data. While lower values of C do the opposite.

Using Default Parameters
First let's see what kind of results we can generate without a grid search using only the base parameters.

To get started we must first load in the dataset we will be working with.

```
from sklearn import datasets
iris = datasets.load_iris()
```

Next in order to create the model we must have a set of independent variables X and a dependant variable y.

```
X = iris['data']
y = iris['target']
```

Now we will load the logistic model for classifying the iris flowers.

```
from sklearn.linear_model import LogisticRegression
```

Creating the model, setting max_iter to a higher value to ensure that the model finds a result.

Keep in mind the default value for C in a logistic regression model is 1, we will compare this later.

In the example below, we look at the iris data set and try to train a model with varying values for C in logistic regression.

```
logit = LogisticRegression(max_iter = 10000)
```

After we create the model, we must fit the model to the data.

```
print(logit.fit(X,y))
```

To evaluate the model we run the score method.

```
print(logit.score(X,y))
```

ExampleGet your own Python Server

```
from sklearn import datasets
from sklearn.linear_model import LogisticRegression

iris = datasets.load_iris()

X = iris['data']
y = iris['target']

logit = LogisticRegression(max_iter = 10000)

print(logit.fit(X,y))

print(logit.score(X,y))
```

With the default setting of C = 1, we achieved a score of 0.973.

Let's see if we can do any better by implementing a grid search with difference values of 0.973.

Implementing Grid Search

We will follow the same steps of before except this time we will set a range of values for C.

Knowing which values to set for the searched parameters will take a combination of domain knowledge and practice.

Since the default value for C is 1, we will set a range of values surrounding it.

```
C = [0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2]
```

Next we will create a for loop to change out the values of C and evaluate the model with each change.

First we will create an empty list to store the score within.

```
scores = []
```

To change the values of C we must loop over the range of values and update the parameter each time.

```
for choice in C:
  logit.set_params(C=choice)
  logit.fit(X, y)
  scores.append(logit.score(X, y))
```

With the scores stored in a list, we can evaluate what the best choice of C is.

```
print(scores)
```

Example

```
from sklearn import datasets
from sklearn.linear_model import LogisticRegression

iris = datasets.load_iris()

X = iris['data']
y = iris['target']

logit = LogisticRegression(max_iter = 10000)

C = [0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2]

scores = []

for choice in C:
  logit.set_params(C=choice)
  logit.fit(X, y)
  scores.append(logit.score(X, y))

print(scores)
```

Results Explained

We can see that the lower values of C performed worse than the base parameter of 1. However, as we increased the value of C to 1.75 the model experienced increased accuracy.

It seems that increasing C beyond this amount does not help increase model accuracy.

Note on Best Practices

We scored our logistic regression model by using the same data that was used to train it. If the model corresponds too closely to that data, it may not be great at predicting unseen data. This statistical error is known as over fitting.

To avoid being misled by the scores on the training data, we can put aside a portion of our data and use it specifically for the purpose of testing the model. Refer to the lecture on train/test splitting to avoid being misled and overfitting.

Preprocessing - Categorical Data

On this page, W3schools.com collaborates with NYC Data Science Academy, to deliver digital training content to our students.

Categorical Data

When your data has categories represented by strings, it will be difficult to use them to train machine learning models which often only accepts numeric data.

Instead of ignoring the categorical data and excluding the information from our model, you can tranform the data so it can be used in your models.

Take a look at the table below, it is the same data set that we used in the multiple regression chapter.

ExampleGet your own Python Server

```
import pandas as pd

cars = pd.read_csv('data.csv')
print(cars.to_string())
```

Result

```
             Car       Model  Volume  Weight  CO2
  0       Toyoty        Aygo    1000     790   99
```

1	Mitsubishi	Space Star	1200	1160	95
2	Skoda	Citigo	1000	929	95
3	Fiat	500	900	865	90
4	Mini	Cooper	1500	1140	105
5	VW	Up!	1000	929	105
6	Skoda	Fabia	1400	1109	90
7	Mercedes	A-Class	1500	1365	92
8	Ford	Fiesta	1500	1112	98
9	Audi	A1	1600	1150	99
10	Hyundai	I20	1100	980	99
11	Suzuki	Swift	1300	990	101
12	Ford	Fiesta	1000	1112	99
13	Honda	Civic	1600	1252	94
14	Hundai	I30	1600	1326	97
15	Opel	Astra	1600	1330	97
16	BMW	1	1600	1365	99
17	Mazda	3	2200	1280	104
18	Skoda	Rapid	1600	1119	104
19	Ford	Focus	2000	1328	105
20	Ford	Mondeo	1600	1584	94
21	Opel	Insignia	2000	1428	99
22	Mercedes	C-Class	2100	1365	99
23	Skoda	Octavia	1600	1415	99
24	Volvo	S60	2000	1415	99
25	Mercedes	CLA	1500	1465	102
26	Audi	A4	2000	1490	104
27	Audi	A6	2000	1725	114
28	Volvo	V70	1600	1523	109
29	BMW	5	2000	1705	114
30	Mercedes	E-Class	2100	1605	115
31	Volvo	XC70	2000	1746	117
32	Ford	B-Max	1600	1235	104
33	BMW	216	1600	1390	108
34	Opel	Zafira	1600	1405	109
35	Mercedes	SLK	2500	1395	120

In the multiple regression chapter, we tried to predict the CO2 emitted based on the volume of the engine and the weight of the car but we excluded information about the car brand and model.

The information about the car brand or the car model might help us make a better prediction of the CO2 emitted.

One Hot Encoding

We cannot make use of the Car or Model column in our data since they are not numeric. A linear relationship between a categorical variable, Car or Model, and a numeric variable, CO2, cannot be determined.

To fix this issue, we must have a numeric representation of the categorical variable. One way to do this is to have a column representing each group in the category.

For each column, the values will be 1 or 0 where 1 represents the inclusion of the group and 0 represents the exclusion. This transformation is called one hot encoding.

You do not have to do this manually, the Python Pandas module has a function that called get_dummies() which does one hot encoding.

Learn about the Pandas module in our Pandas Tutorial.

Example
One Hot Encode the Car column:

```
import pandas as pd

cars = pd.read_csv('data.csv')
ohe_cars = pd.get_dummies(cars[['Car']])

print(ohe_cars.to_string())
```

Result

```
      Car_Audi  Car_BMW  Car_Fiat  Car_Ford  Car_Honda  Car_Hundai  Car_Hyundai
Car_Mazda  Car_Mercedes  Car_Mini  Car_Mitsubishi  Car_Opel  Car_Skoda
Car_Suzuki  Car_Toyoty  Car_VW  Car_Volvo
  0          0        0         0         0          0           0            0
0             0         0               0         0          0           0
1       0          0
  1          0        0         0         0          0           0            0
0             0         0               1         0          0           0
0       0          0
  2          0        0         0         0          0           0            0
0             0         0               0         0          1           0
0       0          0
  3          0        0         1         0          0           0            0
0             0         0               0         0          0           0
0       0          0
  4          0        0         0         0          0           0            0
0             0         1               0         0          0           0
0       0          0
  5          0        0         0         0          0           0            0
0             0         0               0         0          0           0
0       1          0
  6          0        0         0         0          0           0            0
0             0         0               0         0          1           0
0       0          0
  7          0        0         0         0          0           0            0
0             1         0               0         0          0           0
0       0          0
  8          0        0         0         1          0           0            0
0             0         0               0         0          0           0
0       0          0
  9          1        0         0         0          0           0            0
0             0         0               0         0          0           0
0       0          0
  10         0        0         0         0          0           0            1
0             0         0               0         0          0           0
0       0          0
  11         0        0         0         0          0           0            0
0             0         0               0         0          0           1
0       0          0
  12         0        0         0         1          0           0            0
0             0         0               0         0          0           0
0       0          0
  13         0        0         0         0          1           0            0
0             0         0               0         0          0           0
0       0          0
  14         0        0         0         0          0           1            0
0             0         0               0         0          0           0
0       0          0
  15         0        0         0         0          0           0            0
0             0         0               0         1          0           0
0       0          0
  16         0        1         0         0          0           0            0
0             0         0               0         0          0           0
```

```
0       0          0
  17         0        0         0         0          0           0            0
1            0         0              0         0          0           0
0       0          0
  18         0        0         0         0          0           0            0
0            0         0              0         0          1           0
0       0          0
  19         0        0         0         1          0           0            0
0            0         0              0         0          0           0
0       0          0
  20         0        0         0         1          0           0            0
0            0         0              0         0          0           0
0       0          0
  21         0        0         0         0          0           0            0
0            0         0              0         1          0           0
0       0          0
  22         0        0         0         0          0           0            0
0            1         0              0         0          0           0
0       0          0
  23         0        0         0         0          0           0            0
0            0         0              0         0          1           0
0       0          0
  24         0        0         0         0          0           0            0
0            0         0              0         0          0           0
0       0          1
  25         0        0         0         0          0           0            0
0            1         0              0         0          0           0
0       0          0
  26         1        0         0         0          0           0            0
0            0         0              0         0          0           0
0       0          0
  27         1        0         0         0          0           0            0
0            0         0              0         0          0           0
0       0          0
  28         0        0         0         0          0           0            0
0            0         0              0         0          0           0
0       0          1
  29         0        1         0         0          0           0            0
0            0         0              0         0          0           0
0       0          0
  30         0        0         0         0          0           0            0
0            1         0              0         0          0           0
0       0          0
  31         0        0         0         0          0           0            0
0            0         0              0         0          0           0
0       0          1
  32         0        0         0         1          0           0            0
0            0         0              0         0          0           0
0       0          0
  33         0        1         0         0          0           0            0
0            0         0              0         0          0           0
0       0          0
  34         0        0         0         0          0           0            0
0            0         0              0         1          0           0
0       0          0
  35         0        0         0         0          0           0            0
0            1         0              0         0          0           0
0       0          0
```

Results
A column was created for every car brand in the Car column.

Predict CO2
We can use this additional information alongside the volume and weight to

predict CO2

To combine the information, we can use the concat() function from pandas.

First we will need to import a couple modules.

We will start with importing the Pandas.

```
import pandas
```

The pandas module allows us to read csv files and manipulate DataFrame objects:

```
cars = pandas.read_csv("data.csv")
```

It also allows us to create the dummy variables:

```
ohe_cars = pandas.get_dummies(cars[['Car']])
```

Then we must select the independent variables (X) and add the dummy variables columnwise.

Also store the dependent variable in y.

```
X = pandas.concat([cars[['Volume', 'Weight']], ohe_cars], axis=1)
y = cars['CO2']
```

We also need to import a method from sklearn to create a linear model

Learn about linear regression.

```
from sklearn import linear_model
```

Now we can fit the data to a linear regression:

```
regr = linear_model.LinearRegression()
regr.fit(X,y)
```

Finally we can predict the CO2 emissions based on the car's weight, volume, and manufacturer.

```
##predict the CO2 emission of a Volvo where the weight is 2300kg, and the volume is 1300cm3:
predictedCO2 = regr.predict([[2300, 1300,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]])
```

Example

```
import pandas
from sklearn import linear_model

cars = pandas.read_csv("data.csv")
ohe_cars = pandas.get_dummies(cars[['Car']])

X = pandas.concat([cars[['Volume', 'Weight']], ohe_cars], axis=1)
y = cars['CO2']

regr = linear_model.LinearRegression()
regr.fit(X,y)

##predict the CO2 emission of a Volvo where the weight is 2300kg, and the volume is 1300cm3:
predictedCO2 = regr.predict([[2300, 1300,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]])

print(predictedCO2)
```

Result

```
 [122.45153299]
```

We now have a coefficient for the volume, the weight, and each car brand in the data set

Dummifying

It is not necessary to create one column for each group in your category. The information can be retained using 1 column less than the number of groups you have.

For example, you have a column representing colors and in that column, you have two colors, red and blue.

Example

```
import pandas as pd

colors = pd.DataFrame({'color': ['blue', 'red']})

print(colors)
```

Result

```
    color
  0  blue
  1   red
```

You can create 1 column called red where 1 represents red and 0 represents not red, which means it is blue.

To do this, we can use the same function that we used for one hot encoding, get_dummies, and then drop one of the columns. There is an argument, drop_first, which allows us to exclude the first column from the resulting table.

Example

```
import pandas as pd

colors = pd.DataFrame({'color': ['blue', 'red']})
dummies = pd.get_dummies(colors, drop_first=True)

print(dummies)
```

Result

```
     color_red
  0          0
  1          1
```

What if you have more than 2 groups? How can the multiple groups be represented by 1 less column?

Let's say we have three colors this time, red, blue and green. When we get_dummies while dropping the first column, we get the following table.

Example

```
import pandas as pd

colors = pd.DataFrame({'color': ['blue', 'red', 'green']})
dummies = pd.get_dummies(colors, drop_first=True)
dummies['color'] = colors['color']

print(dummies)
```

Result

```
     color_green  color_red  color
  0            0          0   blue
  1            0          1    red
  2            1          0  green
```

Machine Learning - K-means
On this page, W3schools.com collaborates with NYC Data Science Academy, to deliver digital training content to our students.

K-means
K-means is an unsupervised learning method for clustering data points. The algorithm iteratively divides data points into K clusters by minimizing the variance in each cluster.

Here, we will show you how to estimate the best value for K using the elbow method, then use K-means clustering to group the data points into clusters.

How does it work?
First, each data point is randomly assigned to one of the K clusters. Then, we compute the centroid (functionally the center) of each cluster, and reassign each data point to the cluster with the closest centroid. We repeat this process until the cluster assignments for each data point are no longer changing.

K-means clustering requires us to select K, the number of clusters we want to group the data into. The elbow method lets us graph the inertia (a distance-based metric) and visualize the point at which it starts decreasing linearly. This point is referred to as the "eblow" and is a good estimate for the best value for K based on our data.

ExampleGet your own Python Server
Start by visualizing some data points:

```
import matplotlib.pyplot as plt

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

plt.scatter(x, y)
plt.show()
```
Result

Now we utilize the elbow method to visualize the intertia for different values of K:

Example
```
from sklearn.cluster import KMeans

data = list(zip(x, y))
inertias = []

for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```
Result

The elbow method shows that 2 is a good value for K, so we retrain and visualize the result:

Example

```
kmeans = KMeans(n_clusters=2)
kmeans.fit(data)

plt.scatter(x, y, c=kmeans.labels_)
plt.show()
```

Result

Example Explained

Import the modules you need.

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

You can learn about the Matplotlib module in our "Matplotlib Tutorial.

scikit-learn is a popular library for machine learning.

Create arrays that resemble two variables in a dataset. Note that while we only use two variables here, this method will work with any number of variables:

```
x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
```

Turn the data into a set of points:

```
data = list(zip(x, y))
print(data)
```

Result:

```
[(4, 21), (5, 19), (10, 24), (4, 17), (3, 16), (11, 25), (14, 24), (6, 22), (10, 21), (12, 21)]
```

In order to find the best value for K, we need to run K-means across our data for a range of possible values. We only have 10 data points, so the maximum number of clusters is 10. So for each value K in range(1,11), we train a K-means model and plot the intertia at that number of clusters:

```
inertias = []

for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```

Result:

We can see that the "elbow" on the graph above (where the interia becomes more linear) is at K=2. We can then fit our K-means algorithm one more time and plot the different clusters assigned to the data:

```
kmeans = KMeans(n_clusters=2)
kmeans.fit(data)
```

```
plt.scatter(x, y, c=kmeans.labels_)
plt.show()
```

Result:

Machine Learning - Bootstrap Aggregation (Bagging)
On this page, W3schools.com collaborates with NYC Data Science Academy, to deliver digital training content to our students.

Bagging
Methods such as Decision Trees, can be prone to overfitting on the training set which can lead to wrong predictions on new data.

Bootstrap Aggregation (bagging) is a ensembling method that attempts to resolve overfitting for classification or regression problems. Bagging aims to improve the accuracy and performance of machine learning algorithms. It does this by taking random subsets of an original dataset, with replacement, and fits either a classifier (for classification) or regressor (for regression) to each subset. The predictions for each subset are then aggregated through majority vote for classification or averaging for regression, increasing prediction accuracy.

Evaluating a Base Classifier
To see how bagging can improve model performance, we must start by evaluating how the base classifier performs on the dataset. If you do not know what decision trees are review the lesson on decision trees before moving forward, as bagging is an continuation of the concept.

We will be looking to identify different classes of wines found in Sklearn's wine dataset.

Let's start by importing the necessary modules.
```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
```

Next we need to load in the data and store it into X (input features) and y (target). The parameter as_frame is set equal to True so we do not lose the feature names when loading the data. (sklearn version older than 0.23 must skip the as_frame argument as it is not supported)

```
data = datasets.load_wine(as_frame = True)

X = data.data
y = data.target
```

In order to properly evaluate our model on unseen data, we need to split X and y into train and test sets. For information on splitting data, see the Train/Test lesson.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 22)
```

With our data prepared, we can now instantiate a base classifier and fit it to the training data.

```
dtree = DecisionTreeClassifier(random_state = 22)
dtree.fit(X_train,y_train)
```

Result:

DecisionTreeClassifier(random_state=22)

We can now predict the class of wine the unseen test set and evaluate the model performance.

```
y_pred = dtree.predict(X_test)

print("Train data accuracy:",accuracy_score(y_true = y_train, y_pred =
dtree.predict(X_train)))
print("Test data accuracy:",accuracy_score(y_true = y_test, y_pred = y_pred))
```

Result:

```
Train data accuracy: 1.0
Test data accuracy: 0.8222222222222222
```

ExampleGet your own Python Server
Import the necessary data and evaluate base classifier performance.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier

data = datasets.load_wine(as_frame = True)

X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 22)

dtree = DecisionTreeClassifier(random_state = 22)
dtree.fit(X_train,y_train)

y_pred = dtree.predict(X_test)

print("Train data accuracy:",accuracy_score(y_true = y_train, y_pred =
dtree.predict(X_train)))
print("Test data accuracy:",accuracy_score(y_true = y_test, y_pred = y_pred))
```

The base classifier performs reasonably well on the dataset achieving 82% accuracy on the test dataset with the current parameters (Different results may occur if you do not have the random_state parameter set).

Now that we have a baseline accuracy for the test dataset, we can see how the Bagging Classifier out performs a single Decision Tree Classifier.

Creating a Bagging Classifier
For bagging we need to set the parameter n_estimators, this is the number of base classifiers that our model is going to aggregate together.

For this sample dataset the number of estimators is relatively low, it is often the case that much larger ranges are explored. Hyperparameter tuning is usually done with a grid search, but for now we will use a select set of values for the number of estimators.

We start by importing the necessary model.

```
from sklearn.ensemble import BaggingClassifier
```

Now lets create a range of values that represent the number of estimators we want to use in each ensemble.

```
estimator_range = [2,4,6,8,10,12,14,16]
```

To see how the Bagging Classifier performs with differing values of n_estimators we need a way to iterate over the range of values and store the results from each ensemble. To do this we will create a for loop, storing the models and scores in separate lists for later vizualizations.

Note: The default parameter for the base classifier in BaggingClassifier is the DicisionTreeClassifier therefore we do not need to set it when instantiating the bagging model.

```
models = []
scores = []

for n_estimators in estimator_range:

    # Create bagging classifier
    clf = BaggingClassifier(n_estimators = n_estimators, random_state = 22)

    # Fit the model
    clf.fit(X_train, y_train)

    # Append the model and score to their respective list
    models.append(clf)
    scores.append(accuracy_score(y_true = y_test, y_pred = clf.predict(X_test)))
```

With the models and scores stored, we can now visualize the improvement in model performance.

```
import matplotlib.pyplot as plt

# Generate the plot of scores against number of estimators
plt.figure(figsize=(9,6))
plt.plot(estimator_range, scores)

# Adjust labels and font (to make visable)
plt.xlabel("n_estimators", fontsize = 18)
plt.ylabel("score", fontsize = 18)
plt.tick_params(labelsize = 16)

# Visualize plot
plt.show()
```

Example
Import the necessary data and evaluate the BaggingClassifier performance.

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import BaggingClassifier

data = datasets.load_wine(as_frame = True)

X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 22)
```

```
estimator_range = [2,4,6,8,10,12,14,16]

models = []
scores = []

for n_estimators in estimator_range:

    # Create bagging classifier
    clf = BaggingClassifier(n_estimators = n_estimators, random_state = 22)

    # Fit the model
    clf.fit(X_train, y_train)

    # Append the model and score to their respective list
    models.append(clf)
    scores.append(accuracy_score(y_true = y_test, y_pred = clf.predict(X_test)))

# Generate the plot of scores against number of estimators
plt.figure(figsize=(9,6))
plt.plot(estimator_range, scores)

# Adjust labels and font (to make visable)
plt.xlabel("n_estimators", fontsize = 18)
plt.ylabel("score", fontsize = 18)
plt.tick_params(labelsize = 16)

# Visualize plot
plt.show()
```

Result

Results Explained

By iterating through different values for the number of estimators we can see an increase in model performance from 82.2% to 95.5%. After 14 estimators the accuracy begins to drop, again if you set a different random_state the values you see will vary. That is why it is best practice to use cross validation to ensure stable results.

In this case, we see a 13.3% increase in accuracy when it comes to identifying the type of the wine.

Another Form of Evaluation

As bootstrapping chooses random subsets of observations to create classifiers, there are observations that are left out in the selection process. These "out-of-bag" observations can then be used to evaluate the model, similarly to that of a test set. Keep in mind, that out-of-bag estimation can overestimate error in binary classification problems and should only be used as a compliment to other metrics.

We saw in the last exercise that 12 estimators yielded the highest accuracy, so we will use that to create our model. This time setting the parameter oob_score to true to evaluate the model with out-of-bag score.

Example

Create a model with out-of-bag metric.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier

data = datasets.load_wine(as_frame = True)

X = data.data
y = data.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 22)

oob_model = BaggingClassifier(n_estimators = 12, oob_score = True,random_state =
22)

oob_model.fit(X_train, y_train)

print(oob_model.oob_score_)
```

Since the samples used in OOB and the test set are different, and the dataset is relatively small, there is a difference in the accuracy. It is rare that they would be exactly the same, again OOB should be used quick means for estimating error, but is not the only evaluation metric.

Generating Decision Trees from Bagging Classifier
As was seen in the Decision Tree lesson, it is possible to graph the decision tree the model created. It is also possible to see the individual decision trees that went into the aggregated classifier. This helps us to gain a more intuitive understanding on how the bagging model arrives at its predictions.

Note: This is only functional with smaller datasets, where the trees are relatively shallow and narrow making them easy to visualize.

We will need to import plot_tree function from sklearn.tree. The different trees can be graphed by changing the estimator you wish to visualize.

Example
Generate Decision Trees from Bagging Classifier

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import plot_tree

X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 22)

clf = BaggingClassifier(n_estimators = 12, oob_score = True,random_state = 22)

clf.fit(X_train, y_train)

plt.figure(figsize=(30, 20))

plot_tree(clf.estimators_[0], feature_names = X.columns)
```

Result

Here we can see just the first decision tree that was used to vote on the final prediction. Again, by changing the index of the classifier you can see each of the trees that have been aggregated.

Machine Learning - Cross Validation
On this page, W3schools.com collaborates with NYC Data Science Academy, to deliver digital training content to our students.

Cross Validation
When adjusting models we are aiming to increase overall model performance on unseen data. Hyperparameter tuning can lead to much better performance on test

sets. However, optimizing parameters to the test set can lead information leakage causing the model to preform worse on unseen data. To correct for this we can perform cross validation.

To better understand CV, we will be performing different methods on the iris dataset. Let us first load in and separate the data.

```
from sklearn import datasets

X, y = datasets.load_iris(return_X_y=True)
```

There are many methods to cross validation, we will start by looking at k-fold cross validation.

K-Fold
The training data used in the model is split, into k number of smaller sets, to be used to validate the model. The model is then trained on k-1 folds of training set. The remaining fold is then used as a validation set to evaluate the model.

As we will be trying to classify different species of iris flowers we will need to import a classifier model, for this exercise we will be using a DecisionTreeClassifier. We will also need to import CV modules from sklearn.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold, cross_val_score
```

With the data loaded we can now create and fit a model for evaluation.

```
clf = DecisionTreeClassifier(random_state=42)
```

Now let's evaluate our model and see how it performs on each k-fold.

```
k_folds = KFold(n_splits = 5)

scores = cross_val_score(clf, X, y, cv = k_folds)
```

It is also good pratice to see how CV performed overall by averaging the scores for all folds.

ExampleGet your own Python Server
Run k-fold CV:

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold, cross_val_score

X, y = datasets.load_iris(return_X_y=True)

clf = DecisionTreeClassifier(random_state=42)

k_folds = KFold(n_splits = 5)

scores = cross_val_score(clf, X, y, cv = k_folds)

print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```


Stratified K-Fold
In cases where classes are imbalanced we need a way to account for the imbalance in both the train and validation sets. To do so we can stratify the target

classes, meaning that both sets will have an equal proportion of all classes.

Example

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import StratifiedKFold, cross_val_score

X, y = datasets.load_iris(return_X_y=True)

clf = DecisionTreeClassifier(random_state=42)

sk_folds = StratifiedKFold(n_splits = 5)

scores = cross_val_score(clf, X, y, cv = sk_folds)

print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

While the number of folds is the same, the average CV increases from the basic k-fold when making sure there is stratified classes.

Leave-One-Out (LOO)

Instead of selecting the number of splits in the training data set like k-fold LeaveOneOut, utilize 1 observation to validate and n-1 observations to train. This method is an exaustive technique.

Example

Run LOO CV:

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import LeaveOneOut, cross_val_score

X, y = datasets.load_iris(return_X_y=True)

clf = DecisionTreeClassifier(random_state=42)

loo = LeaveOneOut()

scores = cross_val_score(clf, X, y, cv = loo)

print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

We can observe that the number of cross validation scores performed is equal to the number of observations in the dataset. In this case there are 150 observations in the iris dataset.

The average CV score is 94%.

Leave-P-Out (LPO)

Leave-P-Out is simply a nuanced diffence to the Leave-One-Out idea, in that we can select the number of p to use in our validation set.

Example

Run LPO CV:

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import LeavePOut, cross_val_score

X, y = datasets.load_iris(return_X_y=True)

clf = DecisionTreeClassifier(random_state=42)
```

```
lpo = LeavePOut(p=2)

scores = cross_val_score(clf, X, y, cv = lpo)

print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

As we can see this is an exhaustive method we many more scores being calculated than Leave-One-Out, even with a p = 2, yet it achieves roughly the same average CV score.

Shuffle Split

Unlike KFold, ShuffleSplit leaves out a percentage of the data, not to be used in the train or validation sets. To do so we must decide what the train and test sizes are, as well as the number of splits.

Example

Run Shuffle Split CV:

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import ShuffleSplit, cross_val_score

X, y = datasets.load_iris(return_X_y=True)

clf = DecisionTreeClassifier(random_state=42)

ss = ShuffleSplit(train_size=0.6, test_size=0.3, n_splits = 5)

scores = cross_val_score(clf, X, y, cv = ss)

print("Cross Validation Scores: ", scores)
print("Average CV Score: ", scores.mean())
print("Number of CV Scores used in Average: ", len(scores))
```

Ending Notes

These are just a few of the CV methods that can be applied to models. There are many more cross validation classes, with most models having their own class. Check out sklearns cross validation for more CV options.

Machine Learning - AUC - ROC Curve

On this page, W3schools.com collaborates with NYC Data Science Academy, to deliver digital training content to our students.

AUC - ROC Curve

In classification, there are many different evaluation metrics. The most popular is accuracy, which measures how often the model is correct. This is a great metric because it is easy to understand and getting the most correct guesses is often desired. There are some cases where you might consider using another evaluation metric.

Another common metric is AUC, area under the receiver operating characteristic (ROC) curve. The Reciever operating characteristic curve plots the true positive (TP) rate versus the false positive (FP) rate at different classification thresholds. The thresholds are different probability cutoffs that separate the two classes in binary classification. It uses probability to tell us how well a model separates the classes.

Imbalanced Data

Suppose we have an imbalanced data set where the majority of our data is of one value. We can obtain high accuracy for the model by predicting the majority class.

ExampleGet your own Python Server

```
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score,
roc_curve

n = 10000
ratio = .95
n_0 = int((1-ratio) * n)
n_1 = int(ratio * n)

y = np.array([0] * n_0 + [1] * n_1)
# below are the probabilities obtained from a hypothetical model that always
predicts the majority class
# probability of predicting class 1 is going to be 100%
y_proba = np.array([1]*n)
y_pred = y_proba > .5

print(f'accuracy score: {accuracy_score(y, y_pred)}')
cf_mat = confusion_matrix(y, y_pred)
print('Confusion matrix')
print(cf_mat)
print(f'class 0 accuracy: {cf_mat[0][0]/n_0}')
print(f'class 1 accuracy: {cf_mat[1][1]/n_1}')
```


Although we obtain a very high accuracy, the model provided no information about the data so it's not useful. We accurately predict class 1 100% of the time while inaccurately predict class 0 0% of the time. At the expense of accuracy, it might be better to have a model that can somewhat separate the two classes.

Example

```
# below are the probabilities obtained from a hypothetical model that doesn't
always predict the mode
y_proba_2 = np.array(
    np.random.uniform(0, .7, n_0).tolist() +
    np.random.uniform(.3, 1, n_1).tolist()
)
y_pred_2 = y_proba_2 > .5

print(f'accuracy score: {accuracy_score(y, y_pred_2)}')
cf_mat = confusion_matrix(y, y_pred_2)
print('Confusion matrix')
print(cf_mat)
print(f'class 0 accuracy: {cf_mat[0][0]/n_0}')
print(f'class 1 accuracy: {cf_mat[1][1]/n_1}')
```

For the second set of predictions, we do not have as high of an accuracy score as the first but the accuracy for each class is more balanced. Using accuracy as an evaluation metric we would rate the first model higher than the second even though it doesn't tell us anything about the data.

In cases like this, using another evaluation metric like AUC would be preferred.

```
import matplotlib.pyplot as plt

def plot_roc_curve(true_y, y_prob):
    """
    plots the roc curve based of the probabilities
    """

    fpr, tpr, thresholds = roc_curve(true_y, y_prob)
    plt.plot(fpr, tpr)
    plt.xlabel('False Positive Rate')
```

```
    plt.ylabel('True Positive Rate')
```

Example
Model 1:

```
plot_roc_curve(y, y_proba)
print(f'model 1 AUC score: {roc_auc_score(y, y_proba)}')
```

Result

```
model 1 AUC score: 0.5
```

Example
Model 2:

```
plot_roc_curve(y, y_proba_2)
print(f'model 2 AUC score: {roc_auc_score(y, y_proba_2)}')
```

Result

```
model 2 AUC score: 0.8270551578947367
```

An AUC score of around .5 would mean that the model is unable to make a distinction between the two classes and the curve would look like a line with a slope of 1. An AUC score closer to 1 means that the model has the ability to separate the two classes and the curve would come closer to the top left corner of the graph.

Probabilities
Because AUC is a metric that utilizes probabilities of the class predictions, we can be more confident in a model that has a higher AUC score than one with a lower score even if they have similar accuracies.

In the data below, we have two sets of probabilites from hypothetical models. The first has probabilities that are not as "confident" when predicting the two classes (the probabilities are close to .5). The second has probabilities that are more "confident" when predicting the two classes (the probabilities are close to the extremes of 0 or 1).

Example

```
import numpy as np

n = 10000
y = np.array([0] * n + [1] * n)
#
y_prob_1 = np.array(
    np.random.uniform(.25, .5, n//2).tolist() +
    np.random.uniform(.3, .7, n).tolist() +
    np.random.uniform(.5, .75, n//2).tolist()
)
y_prob_2 = np.array(
    np.random.uniform(0, .4, n//2).tolist() +
    np.random.uniform(.3, .7, n).tolist() +
    np.random.uniform(.6, 1, n//2).tolist()
)

print(f'model 1 accuracy score: {accuracy_score(y, y_prob_1>.5)}')
print(f'model 2 accuracy score: {accuracy_score(y, y_prob_2>.5)}')

print(f'model 1 AUC score: {roc_auc_score(y, y_prob_1)}')
print(f'model 2 AUC score: {roc_auc_score(y, y_prob_2)}')
```

Example
Plot model 1:

```
plot_roc_curve(y, y_prob_1)
```

Result

Example
Plot model 2:

```
fpr, tpr, thresholds = roc_curve(y, y_prob_2)
plt.plot(fpr, tpr)
```
Result

Even though the accuracies for the two models are similar, the model with the higher AUC score will be more reliable because it takes into account the predicted probability. It is more likely to give you higher accuracy when predicting future data.

Machine Learning - K-nearest neighbors (KNN)
On this page, W3schools.com collaborates with NYC Data Science Academy, to deliver digital training content to our students.

KNN
KNN is a simple, supervised machine learning (ML) algorithm that can be used for classification or regression tasks - and is also frequently used in missing value imputation. It is based on the idea that the observations closest to a given data point are the most "similar" observations in a data set, and we can therefore classify unforeseen points based on the values of the closest existing points. By choosing K, the user can select the number of nearby observations to use in the algorithm.

Here, we will show you how to implement the KNN algorithm for classification, and show how different values of K affect the results.

How does it work?
K is the number of nearest neighbors to use. For classification, a majority vote is used to determined which class a new observation should fall into. Larger values of K are often more robust to outliers and produce more stable decision boundaries than very small values (K=3 would be better than K=1, which might produce undesirable results.

ExampleGet your own Python Server
Start by visualizing some data points:

```
import matplotlib.pyplot as plt

x = [4, 5, 10, 4, 3, 11, 14 , 8, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
classes = [0, 0, 1, 0, 0, 1, 1, 0, 1, 1]

plt.scatter(x, y, c=classes)
plt.show()
```
Result

Now we fit the KNN algorithm with K=1:

```
from sklearn.neighbors import KNeighborsClassifier

data = list(zip(x, y))
knn = KNeighborsClassifier(n_neighbors=1)
```

```
knn.fit(data, classes)
```

And use it to classify a new data point:

Example
```
new_x = 8
new_y = 21
new_point = [(new_x, new_y)]

prediction = knn.predict(new_point)

plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```
Result

Now we do the same thing, but with a higher K value which changes the prediction:

Example
```
knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(data, classes)

prediction = knn.predict(new_point)

plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```
Result

Example Explained
Import the modules you need.

You can learn about the Matplotlib module in our "Matplotlib Tutorial.

scikit-learn is a popular library for machine learning in Python.

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
```

Create arrays that resemble variables in a dataset. We have two input features (x and y) and then a target class (class). The input features that are pre-labeled with our target class will be used to predict the class of new data. Note that while we only use two input features here, this method will work with any number of variables:

```
x = [4, 5, 10, 4, 3, 11, 14 , 8, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
classes = [0, 0, 1, 0, 0, 1, 1, 0, 1, 1]
```

Turn the input features into a set of points:

```
data = list(zip(x, y))
print(data)
```

Result:
```
[(4, 21), (5, 19), (10, 24), (4, 17), (3, 16), (11, 25), (14, 24), (8, 22), (10, 21), (12, 21)]
```

Using the input features and target class, we fit a KNN model on the model using 1 nearest neighbor:

```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(data, classes)
```

Then, we can use the same KNN object to predict the class of new, unforeseen data points. First we create new x and y features, and then call knn.predict() on the new data point to get a class of 0 or 1:

```
new_x = 8
new_y = 21
new_point = [(new_x, new_y)]
prediction = knn.predict(new_point)
print(prediction)
```

Result:
[0]

When we plot all the data along with the new point and class, we can see it's been labeled blue with the 1 class. The text annotation is just to highlight the location of the new point:

```
plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```

Result:

However, when we changes the number of neighbors to 5, the number of points used to classify our new point changes. As a result, so does the classification of the new point:

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(data, classes)
prediction = knn.predict(new_point)
print(prediction)
```

Result:
[1]

When we plot the class of the new point along with the older points, we note that the color has changed based on the associated class label:

```
plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```

Result:

Python MySQL
Python can be used in database applications.

One of the most popular databases is MySQL.

MySQL Database
To be able to experiment with the code examples in this tutorial, you should have MySQL installed on your computer.

You can download a MySQL database at https://www.mysql.com/downloads/.

Install MySQL Driver
Python needs a MySQL driver to access the MySQL database.

In this tutorial we will use the driver "MySQL Connector".

We recommend that you use PIP to install "MySQL Connector".

PIP is most likely already installed in your Python environment.

Navigate your command line to the location of PIP, and type the following:

Download and install "MySQL Connector":

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>python -m pip install mysql-connector-python
```

Now you have downloaded and installed a MySQL driver.

Test MySQL Connector
To test if the installation was successful, or if you already have "MySQL Connector" installed, create a Python page with the following content:

demo_mysql_test.py:

```
import mysql.connector
```

If the above code was executed with no errors, "MySQL Connector" is installed and ready to be used.

ADVERTISEMENT

Create Connection
Start by creating a connection to the database.

Use the username and password from your MySQL database:

demo_mysql_connection.py:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)

print(mydb)
```

Now you can start querying the database using SQL statements.

Python MySQL Create Database
Creating a Database
To create a database in MySQL, use the "CREATE DATABASE" statement:

ExampleGet your own Python Server
create a database named "mydatabase":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)

mycursor = mydb.cursor()
```

```
mycursor.execute("CREATE DATABASE mydatabase")
```

If the above code was executed with no errors, you have successfully created a database.

Check if Database Exists

You can check if a database exist by listing all databases in your system by using the "SHOW DATABASES" statement:

Example

Return a list of your system's databases:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)

mycursor = mydb.cursor()

mycursor.execute("SHOW DATABASES")

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

Or you can try to access the database when making the connection:

Example

Try connecting to the database "mydatabase":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
```

If the database does not exist, you will get an error.

Python MySQL Create Table

Creating a Table

To create a table in MySQL, use the "CREATE TABLE" statement.

Make sure you define the name of the database when you create the connection

ExampleGet your own Python Server

Create a table named "customers":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE customers (name VARCHAR(255), address
VARCHAR(255))")
```

If the above code was executed with no errors, you have now successfully created a table.

Check if Table Exists

You can check if a table exist by listing all tables in your database with the "SHOW TABLES" statement:

Example

Return a list of your system's databases:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("SHOW TABLES")

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


Primary Key

When creating a table, you should also create a column with a unique key for each record.

This can be done by defining a PRIMARY KEY.

We use the statement "INT AUTO_INCREMENT PRIMARY KEY" which will insert a unique number for each record. Starting at 1, and increased by one for each record.

Example

Create primary key when creating the table:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE customers (id INT AUTO_INCREMENT PRIMARY KEY, 
name VARCHAR(255), address VARCHAR(255))")
```

If the table already exists, use the ALTER TABLE keyword:

Example

Create primary key on an existing table:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
```

```
)

mycursor = mydb.cursor()

mycursor.execute("ALTER TABLE customers ADD COLUMN id INT AUTO_INCREMENT PRIMARY
KEY")
```

Python MySQL Insert Into Table
Insert Into Table
To fill a table in MySQL, use the "INSERT INTO" statement.

ExampleGet your own Python Server
Insert a record in the "customers" table:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = ("John", "Highway 21")
mycursor.execute(sql, val)

mydb.commit()

print(mycursor.rowcount, "record inserted.")
```

Important!: Notice the statement: mydb.commit(). It is required to make the changes, otherwise no changes are made to the table.

Insert Multiple Rows
To insert multiple rows into a table, use the executemany() method.

The second parameter of the executemany() method is a list of tuples, containing the data you want to insert:

Example
Fill the "customers" table with data:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = [
  ('Peter', 'Lowstreet 4'),
  ('Amy', 'Apple st 652'),
  ('Hannah', 'Mountain 21'),
  ('Michael', 'Valley 345'),
  ('Sandy', 'Ocean blvd 2'),
```

```
  ('Betty', 'Green Grass 1'),
  ('Richard', 'Sky st 331'),
  ('Susan', 'One way 98'),
  ('Vicky', 'Yellow Garden 2'),
  ('Ben', 'Park Lane 38'),
  ('William', 'Central st 954'),
  ('Chuck', 'Main Road 989'),
  ('Viola', 'Sideway 1633')
]

mycursor.executemany(sql, val)

mydb.commit()

print(mycursor.rowcount, "was inserted.")
```


Get Inserted ID
You can get the id of the row you just inserted by asking the cursor object.

Note: If you insert more than one row, the id of the last inserted row is returned.

Example
Insert one row, and return the ID:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = ("Michelle", "Blue Village")
mycursor.execute(sql, val)

mydb.commit()

print("1 record inserted, ID:", mycursor.lastrowid)
```

Python MySQL Select From
Select From a Table
To select from a table in MySQL, use the "SELECT" statement:

ExampleGet your own Python Server
Select all records from the "customers" table, and display the result:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()
```

```
mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchall()

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

Note: We use the fetchall() method, which fetches all rows from the last executed statement.

Selecting Columns

To select only some of the columns in a table, use the "SELECT" statement followed by the column name(s):

Example

Select only the name and address columns:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("SELECT name, address FROM customers")

myresult = mycursor.fetchall()

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


Using the fetchone() Method

If you are only interested in one row, you can use the fetchone() method.

The fetchone() method will return the first row of the result:

Example

Fetch only one row:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchone()

print(myresult)
```

Python MySQL Where

Select With a Filter
When selecting records from a table, you can filter the selection by using the "WHERE" statement:

ExampleGet your own Python Server
Select record(s) where the address is "Park Lane 38": result:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address ='Park Lane 38'"

mycursor.execute(sql)

myresult = mycursor.fetchall()

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

Wildcard Characters
You can also select the records that starts, includes, or ends with a given letter or phrase.

Use the % to represent wildcard characters:

Example
Select records where the address contains the word "way":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address LIKE '%way%'"

mycursor.execute(sql)

myresult = mycursor.fetchall()

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


Prevent SQL Injection
When query values are provided by the user, you should escape the values.

This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.

The mysql.connector module has methods to escape query values:

Example
Escape query values by using the placholder %s method:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address = %s"
adr = ("Yellow Garden 2", )

mycursor.execute(sql, adr)

myresult = mycursor.fetchall()

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

Python MySQL Order By
Sort the Result
Use the ORDER BY statement to sort the result in ascending or descending order.

The ORDER BY keyword sorts the result ascending by default. To sort the result in descending order, use the DESC keyword.

ExampleGet your own Python Server
Sort the result alphabetically by name: result:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers ORDER BY name"

mycursor.execute(sql)

myresult = mycursor.fetchall()

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

ORDER BY DESC
Use the DESC keyword to sort the result in a descending order.

Example
Sort the result reverse alphabetically by name:

```
import mysql.connector

mydb = mysql.connector.connect(
```

```
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers ORDER BY name DESC"

mycursor.execute(sql)

myresult = mycursor.fetchall()

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

Python MySQL Delete From By
Delete Record
You can delete records from an existing table by using the "DELETE FROM" statement:

ExampleGet your own Python Server
Delete any record where the address is "Mountain 21":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "DELETE FROM customers WHERE address = 'Mountain 21'"

mycursor.execute(sql)

mydb.commit()

print(mycursor.rowcount, "record(s) deleted")
```

Important!: Notice the statement: mydb.commit(). It is required to make the changes, otherwise no changes are made to the table.

Notice the WHERE clause in the DELETE syntax: The WHERE clause specifies which record(s) that should be deleted. If you omit the WHERE clause, all records will be deleted!

Prevent SQL Injection
It is considered a good practice to escape the values of any query, also in delete statements.

This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.

The mysql.connector module uses the placeholder %s to escape values in the delete statement:

Example
Escape values by using the placeholder %s method:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "DELETE FROM customers WHERE address = %s"
adr = ("Yellow Garden 2", )

mycursor.execute(sql, adr)

mydb.commit()

print(mycursor.rowcount, "record(s) deleted")
```

Python MySQL Drop Table
Delete a Table
You can delete an existing table by using the "DROP TABLE" statement:

ExampleGet your own Python Server
Delete the table "customers":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "DROP TABLE customers"

mycursor.execute(sql)
```

Drop Only if Exist
If the table you want to delete is already deleted, or for any other reason does not exist, you can use the IF EXISTS keyword to avoid getting an error.

Example
Delete the table "customers" if it exists:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()
```

```
sql = "DROP TABLE IF EXISTS customers"

mycursor.execute(sql)
```

Python MySQL Update Table
Update Table
You can update existing records in a table by using the "UPDATE" statement:

ExampleGet your own Python Server
Overwrite the address column from "Valley 345" to "Canyon 123":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'"

mycursor.execute(sql)

mydb.commit()

print(mycursor.rowcount, "record(s) affected")
```

Important!: Notice the statement: mydb.commit(). It is required to make the changes, otherwise no changes are made to the table.

Notice the WHERE clause in the UPDATE syntax: The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Prevent SQL Injection
It is considered a good practice to escape the values of any query, also in update statements.

This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.

The mysql.connector module uses the placeholder %s to escape values in the delete statement:

Example
Escape values by using the placeholder %s method:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()
```

```
sql = "UPDATE customers SET address = %s WHERE address = %s"
val = ("Valley 345", "Canyon 123")

mycursor.execute(sql, val)

mydb.commit()

print(mycursor.rowcount, "record(s) affected")
```

Python MySQL Limit
Limit the Result
You can limit the number of records returned from the query, by using the "LIMIT" statement:

ExampleGet your own Python Server
Select the 5 first records in the "customers" table:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers LIMIT 5")

myresult = mycursor.fetchall()

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

Start From Another Position
If you want to return five records, starting from the third record, you can use the "OFFSET" keyword:

Example
Start from position 3, and return 5 records:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers LIMIT 5 OFFSET 2")

myresult = mycursor.fetchall()

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

Python MySQL Join

Join Two or More Tables
You can combine rows from two or more tables, based on a related column between them, by using a JOIN statement.

Consider you have a "users" table and a "products" table:

usersGet your own Python Server
```
{ id: 1, name: 'John', fav: 154},
{ id: 2, name: 'Peter', fav: 154},
{ id: 3, name: 'Amy', fav: 155},
{ id: 4, name: 'Hannah', fav:},
{ id: 5, name: 'Michael', fav:}
```
products
```
{ id: 154, name: 'Chocolate Heaven' },
{ id: 155, name: 'Tasty Lemons' },
{ id: 156, name: 'Vanilla Dreams' }
```
These two tables can be combined by using users' fav field and products' id field.

Example
Join users and products to see the name of the users favorite product:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "SELECT \
  users.name AS user, \
  products.name AS favorite \
  FROM users \
  INNER JOIN products ON users.fav = products.id"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```
Note: You can use JOIN instead of INNER JOIN. They will both give you the same result.

LEFT JOIN
In the example above, Hannah, and Michael were excluded from the result, that is because INNER JOIN only shows the records where there is a match.

If you want to show all users, even if they do not have a favorite product, use the LEFT JOIN statement:

Example
Select all users and their favorite product:

```
sql = "SELECT \
  users.name AS user, \
  products.name AS favorite \
  FROM users \
```

```
  LEFT JOIN products ON users.fav = products.id"
```
RIGHT JOIN
If you want to return all products, and the users who have them as their favorite, even if no user have them as their favorite, use the RIGHT JOIN statement:

Example
Select all products, and the user(s) who have them as their favorite:

```
sql = "SELECT \
  users.name AS user, \
  products.name AS favorite \
  FROM users \
  RIGHT JOIN products ON users.fav = products.id"
```
Note: Hannah and Michael, who have no favorite product, are not included in the result.

Python MongoDB
Python can be used in database applications.

One of the most popular NoSQL database is MongoDB.

MongoDB
MongoDB stores data in JSON-like documents, which makes the database very flexible and scalable.

To be able to experiment with the code examples in this tutorial, you will need access to a MongoDB database.

You can download a free MongoDB database at https://www.mongodb.com.

Or get started right away with a MongoDB cloud service at https://www.mongodb.com/cloud/atlas.

PyMongo
Python needs a MongoDB driver to access the MongoDB database.

In this tutorial we will use the MongoDB driver "PyMongo".

We recommend that you use PIP to install "PyMongo".

PIP is most likely already installed in your Python environment.

Navigate your command line to the location of PIP, and type the following:

Download and install "PyMongo":

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>python -m pip install pymongo
```
Now you have downloaded and installed a mongoDB driver.

Test PyMongo
To test if the installation was successful, or if you already have "pymongo" installed, create a Python page with the following content:

demo_mongodb_test.py:

```
import pymongo
```
If the above code was executed with no errors, "pymongo" is installed and ready to be used.

Python MongoDB Create Database
Creating a Database
To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database you want to create.

MongoDB will create the database if it does not exist, and make a connection to it.

ExampleGet your own Python Server
Create a database called "mydatabase":

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")

mydb = myclient["mydatabase"]
```
Important: In MongoDB, a database is not created until it gets content!

MongoDB waits until you have created a collection (table), with at least one document (record) before it actually creates the database (and collection).

Check if Database Exists
Remember: In MongoDB, a database is not created until it gets content, so if this is your first time creating a database, you should complete the next two chapters (create collection and create document) before you check if the database exists!

You can check if a database exist by listing all databases in you system:

Example
Return a list of your system's databases:

```
print(myclient.list_database_names())
```
Or you can check a specific database by name:

Example
Check if "mydatabase" exists:

```
dblist = myclient.list_database_names()
if "mydatabase" in dblist:
  print("The database exists.")
```

Python MongoDB Create Collection
A collection in MongoDB is the same as a table in SQL databases.

Creating a Collection
To create a collection in MongoDB, use database object and specify the name of the collection you want to create.

MongoDB will create the collection if it does not exist.

ExampleGet your own Python Server
Create a collection called "customers":

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]

mycol = mydb["customers"]
```

Important: In MongoDB, a collection is not created until it gets content!

MongoDB waits until you have inserted a document before it actually creates the collection.

Check if Collection Exists
Remember: In MongoDB, a collection is not created until it gets content, so if this is your first time creating a collection, you should complete the next chapter (create document) before you check if the collection exists!

You can check if a collection exist in a database by listing all collections:

Example
Return a list of all collections in your database:

```
print(mydb.list_collection_names())
```

Or you can check a specific collection by name:

Example
Check if the "customers" collection exists:

```
collist = mydb.list_collection_names()
if "customers" in collist:
  print("The collection exists.")
```

Python MongoDB Insert Document
A document in MongoDB is the same as a record in SQL databases.

Insert Into Collection
To insert a record, or document as it is called in MongoDB, into a collection, we use the insert_one() method.

The first parameter of the insert_one() method is a dictionary containing the name(s) and value(s) of each field in the document you want to insert.

ExampleGet your own Python Server
Insert a record in the "customers" collection:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mydict = { "name": "John", "address": "Highway 37" }

x = mycol.insert_one(mydict)
```

Return the _id Field
The insert_one() method returns a InsertOneResult object, which has a property, inserted_id, that holds the id of the inserted document.

Example
Insert another record in the "customers" collection, and return the value of the _id field:

```
mydict = { "name": "Peter", "address": "Lowstreet 27" }

x = mycol.insert_one(mydict)

print(x.inserted_id)
```

If you do not specify an _id field, then MongoDB will add one for you and assign a unique id for each document.

In the example above no _id field was specified, so MongoDB assigned a unique _id for the record (document).

Insert Multiple Documents
To insert multiple documents into a collection in MongoDB, we use the insert_many() method.

The first parameter of the insert_many() method is a list containing dictionaries with the data you want to insert:

Example
```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mylist = [
  { "name": "Amy", "address": "Apple st 652"},
  { "name": "Hannah", "address": "Mountain 21"},
  { "name": "Michael", "address": "Valley 345"},
  { "name": "Sandy", "address": "Ocean blvd 2"},
  { "name": "Betty", "address": "Green Grass 1"},
  { "name": "Richard", "address": "Sky st 331"},
  { "name": "Susan", "address": "One way 98"},
  { "name": "Vicky", "address": "Yellow Garden 2"},
  { "name": "Ben", "address": "Park Lane 38"},
  { "name": "William", "address": "Central st 954"},
  { "name": "Chuck", "address": "Main Road 989"},
  { "name": "Viola", "address": "Sideway 1633"}
]

x = mycol.insert_many(mylist)

#print list of the _id values of the inserted documents:
print(x.inserted_ids)
```
The insert_many() method returns a InsertManyResult object, which has a property, inserted_ids, that holds the ids of the inserted documents.

Insert Multiple Documents, with Specified IDs
If you do not want MongoDB to assign unique ids for you document, you can specify the _id field when you insert the document(s).

Remember that the values has to be unique. Two documents cannot have the same _id.

Example
```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mylist = [
  { "_id": 1, "name": "John", "address": "Highway 37"},
  { "_id": 2, "name": "Peter", "address": "Lowstreet 27"},
  { "_id": 3, "name": "Amy", "address": "Apple st 652"},
  { "_id": 4, "name": "Hannah", "address": "Mountain 21"},
  { "_id": 5, "name": "Michael", "address": "Valley 345"},
  { "_id": 6, "name": "Sandy", "address": "Ocean blvd 2"},
  { "_id": 7, "name": "Betty", "address": "Green Grass 1"},
```

```
  { "_id": 8, "name": "Richard", "address": "Sky st 331"},
  { "_id": 9, "name": "Susan", "address": "One way 98"},
  { "_id": 10, "name": "Vicky", "address": "Yellow Garden 2"},
  { "_id": 11, "name": "Ben", "address": "Park Lane 38"},
  { "_id": 12, "name": "William", "address": "Central st 954"},
  { "_id": 13, "name": "Chuck", "address": "Main Road 989"},
  { "_id": 14, "name": "Viola", "address": "Sideway 1633"}
]

x = mycol.insert_many(mylist)

#print list of the _id values of the inserted documents:
print(x.inserted_ids)
```

Python MongoDB Find
In MongoDB we use the find() and find_one() methods to find data in a collection.

Just like the SELECT statement is used to find data in a table in a MySQL database.

Find One
To select data from a collection in MongoDB, we can use the find_one() method.

The find_one() method returns the first occurrence in the selection.

ExampleGet your own Python Server
Find the first document in the customers collection:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

x = mycol.find_one()

print(x)
```

Find All
To select data from a table in MongoDB, we can also use the find() method.

The find() method returns all occurrences in the selection.

The first parameter of the find() method is a query object. In this example we use an empty query object, which selects all documents in the collection.

No parameters in the find() method gives you the same result as SELECT * in MySQL.

Example
Return all documents in the "customers" collection, and print each document:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

for x in mycol.find():
  print(x)
```

Return Only Some Fields
The second parameter of the find() method is an object describing which fields to include in the result.

This parameter is optional, and if omitted, all fields will be included in the result.

Example
Return only the names and addresses, not the _ids:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

for x in mycol.find({},{ "_id": 0, "name": 1, "address": 1 }):
  print(x)
```

You are not allowed to specify both 0 and 1 values in the same object (except if one of the fields is the _id field). If you specify a field with the value 0, all other fields get the value 1, and vice versa:

Example
This example will exclude "address" from the result:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

for x in mycol.find({},{ "address": 0 }):
  print(x)
```

Example
You get an error if you specify both 0 and 1 values in the same object (except if one of the fields is the _id field):

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

for x in mycol.find({},{ "name": 1, "address": 0 }):
  print(x)
```

Python MongoDB Query
Filter the Result
When finding documents in a collection, you can filter the result by using a query object.

The first argument of the find() method is a query object, and is used to limit the search.

ExampleGet your own Python Server
Find document(s) with the address "Park Lane 38":

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
```

```
myquery = { "address": "Park Lane 38" }

mydoc = mycol.find(myquery)

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

Advanced Query

To make advanced queries you can use modifiers as values in the query object.

E.g. to find the documents where the "address" field starts with the letter "S" or higher (alphabetically), use the greater than modifier: {"$gt": "S"}:

Example

Find documents where the address starts with the letter "S" or higher:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": { "$gt": "S" } }

mydoc = mycol.find(myquery)

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

Filter With Regular Expressions

You can also use regular expressions as a modifier.

Regular expressions can only be used to query strings.

To find only the documents where the "address" field starts with the letter "S", use the regular expression {"$regex": "^S"}:

Example

Find documents where the address starts with the letter "S":

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": { "$regex": "^S" } }

mydoc = mycol.find(myquery)

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

Python MongoDB Sort

Sort the Result

Use the sort() method to sort the result in ascending or descending order.

The sort() method takes one parameter for "fieldname" and one parameter for "direction" (ascending is the default direction).

ExampleGet your own Python Server

Sort the result alphabetically by name:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mydoc = mycol.find().sort("name")

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

Sort Descending
Use the value -1 as the second parameter to sort descending.

```
sort("name", 1) #ascending
sort("name", -1) #descending
```

Example
Sort the result reverse alphabetically by name:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mydoc = mycol.find().sort("name", -1)

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

Python MongoDB Delete Document
Delete Document
To delete one document, we use the delete_one() method.

The first parameter of the delete_one() method is a query object defining which document to delete.

Note: If the query finds more than one document, only the first occurrence is deleted.

ExampleGet your own Python Server
Delete the document with the address "Mountain 21":

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": "Mountain 21" }

mycol.delete_one(myquery)
```

Delete Many Documents
To delete more than one document, use the delete_many() method.

The first parameter of the delete_many() method is a query object defining which documents to delete.

Example
Delete all documents were the address starts with the letter S:

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": {"$regex": "^S"} }

x = mycol.delete_many(myquery)

print(x.deleted_count, " documents deleted.")
```

Delete All Documents in a Collection
To delete all documents in a collection, pass an empty query object to the delete_many() method:

Example
Delete all documents in the "customers" collection:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

x = mycol.delete_many({})

print(x.deleted_count, " documents deleted.")
```

Python MongoDB Drop Collection
Delete Collection
You can delete a table, or collection as it is called in MongoDB, by using the drop() method.

ExampleGet your own Python Server
Delete the "customers" collection:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mycol.drop()
```

The drop() method returns true if the collection was dropped successfully, and false if the collection does not exist.

Python MongoDB Update
Update Collection
You can update a record, or document as it is called in MongoDB, by using the update_one() method.

The first parameter of the update_one() method is a query object defining which document to update.

Note: If the query finds more than one record, only the first occurrence is updated.

The second parameter is an object defining the new values of the document.

ExampleGet your own Python Server
Change the address from "Valley 345" to "Canyon 123":

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": "Valley 345" }
newvalues = { "$set": { "address": "Canyon 123" } }

mycol.update_one(myquery, newvalues)

#print "customers" after the update:
for x in mycol.find():
  print(x)
```

Update Many
To update all documents that meets the criteria of the query, use the update_many() method.

Example
Update all documents where the address starts with the letter "S":

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": { "$regex": "^S" } }
newvalues = { "$set": { "name": "Minnie" } }

x = mycol.update_many(myquery, newvalues)

print(x.modified_count, "documents updated.")
```

Python MongoDB Limit
Limit the Result
To limit the result in MongoDB, we use the limit() method.

The limit() method takes one parameter, a number defining how many documents to return.

Consider you have a "customers" collection:

CustomersGet your own Python Server

```
{'_id': 1, 'name': 'John', 'address': 'Highway37'}
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}
```

Example
Limit the result to only return 5 documents:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myresult = mycol.find().limit(5)

#print the result:
for x in myresult:
  print(x)
```

Python Reference
This section contains a Python reference documentation.

Python Reference
Built-in Functions String Methods List Methods Dictionary Methods Tuple Methods Set Methods File Methods Keywords Exceptions Glossary

Module Reference
Random Module Requests Module Math Module CMath Module

Python Built in Functions
Python has a set of built-in functions.

Function	Description
abs() Returns the absolute value of a number
all() Returns True if all items in an iterable object are true
any() Returns True if any item in an iterable object is true
ascii()	Returns a readable version of an object. Replaces none-ascii characters with escape character
bin() Returns the binary version of a number
bool()	Returns the boolean value of the specified object
bytearray() Returns an array of bytes
bytes()	Returns a bytes object
callable()	Returns True if the specified object is callable, otherwise False
chr() Returns a character from the specified Unicode code.
classmethod()	Converts a method into a class method
compile()	Returns the specified source as an object, ready to be executed
complex()	Returns a complex number
delattr()	Deletes the specified attribute (property or method) from the specified object
dict()	Returns a dictionary (Array)
dir() Returns a list of the specified object's properties and methods
divmod()	Returns the quotient and the remainder when argument1 is divided by argument2
enumerate() Takes a collection (e.g. a tuple) and returns it as an enumerate object
eval()	Evaluates and executes an expression
exec()	Executes the specified code (or object)
filter()	Use a filter function to exclude items in an iterable object
float()	Returns a floating point number
format()	Formats a specified value
frozenset() Returns a frozenset object
getattr()	Returns the value of the specified attribute (property or method)
globals()	Returns the current global symbol table as a dictionary
hasattr()	Returns True if the specified object has the specified attribute (property/method)
hash()	Returns the hash value of a specified object
help()	Executes the built-in help system

hex() Converts a number into a hexadecimal value
id() Returns the id of an object
input() Allowing user input
int() Returns an integer number
isinstance() Returns True if a specified object is an instance of a specified object
issubclass() Returns True if a specified class is a subclass of a specified object
iter() Returns an iterator object
len() Returns the length of an object
list() Returns a list
locals() Returns an updated dictionary of the current local symbol table
map() Returns the specified iterator with the specified function applied to each item
max() Returns the largest item in an iterable
memoryview() Returns a memory view object
min() Returns the smallest item in an iterable
next() Returns the next item in an iterable
object() Returns a new object
oct() Converts a number into an octal
open() Opens a file and returns a file object
ord() Convert an integer representing the Unicode of the specified character
pow() Returns the value of x to the power of y
print() Prints to the standard output device
property() Gets, sets, deletes a property
range() Returns a sequence of numbers, starting from 0 and increments by 1 (by default)
repr() Returns a readable version of an object
reversed() Returns a reversed iterator
round() Rounds a numbers
set() Returns a new set object
setattr() Sets an attribute (property/method) of an object
slice() Returns a slice object
sorted() Returns a sorted list
staticmethod() Converts a method into a static method
str() Returns a string object
sum() Sums the items of an iterator
super() Returns an object that represents the parent class
tuple() Returns a tuple
type() Returns the type of an object
vars() Returns the __dict__ property of an object
zip() Returns an iterator, from two or more iterators

Python String Methods
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
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
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
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
Note: All string methods returns new values. They do not change the original string.

Learn more about strings in our Python Strings Tutorial.

Python List/Array Methods
Python has a set of built-in methods that you can use on lists/arrays.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list
Note: Python does not have built-in support for Arrays, but Python Lists can be used instead.

Learn more about lists in our Python Lists Tutorial.

Learn more about arrays in our Python Arrays Tutorial.

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

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary
Learn more about dictionaries in our Python Dictionaries Tutorial.

Python 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
Learn more about tuples in our Python Tuples Tutorial.

Python 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 or more 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
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 another set, or any other iterable
Learn more about sets in our Python Sets Tutorial.

Python File Methods
Python has a set of methods available for the file object.

Method	Description
close()	Closes the file
detach()	Returns the separated raw stream from the buffer
fileno()	Returns a number that represents the stream, from the operating system's perspective
flush()	Flushes the internal buffer
isatty()	Returns whether the file stream is interactive or not
read()	Returns the file content
readable()	Returns whether the file stream can be read or not
readline()	Returns one line from the file
readlines()	Returns a list of lines from the file
seek()	Change the file position
seekable()	Returns whether the file allows us to change the file position
tell()	Returns the current file position
truncate()	Resizes the file to a specified size
writable()	Returns whether the file can be written to or not
write()	Writes the specified string to the file
writelines()	Writes a list of strings to the file

Learn more about the file object in our Python File Handling Tutorial.

Python Keywords
Python has a set of keywords that are reserved words that cannot be used as variable names, function names, or any other identifiers:

Keyword	Description
and	A logical operator
as	To create an alias
assert	For debugging
break	To break out of a loop
class	To define a class
continue	To continue to the next iteration of a loop
def	To define a function
del	To delete an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements
except	Used with exceptions, what to do when an exception occurs
False	Boolean value, result of comparison operations
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
for	To create a for loop
from	To import specific parts of a module
global	To declare a global variable
if	To make a conditional statement
import	To import a module
in	To check if a value is present in a list, tuple, etc.
is	To test if two variables are equal
lambda	To create an anonymous function
None	Represents a null value
nonlocal	To declare a non-local variable
not	A logical operator
or	A logical operator
pass	A null statement, a statement that will do nothing
raise	To raise an exception
return	To exit a function and return a value
True	Boolean value, result of comparison operations
try	To make a try...except statement

while To create a while loop
with Used to simplify exception handling
yield To end a function, returns a generator

Python Built-in Exceptions
Built-in Exceptions
The table below shows built-in exceptions that are usually raised in Python:

Exception	Description
ArithmeticError	Raised when an error occurs in numeric calculations
AssertionError	Raised when an assert statement fails
AttributeError	Raised when attribute reference or assignment fails
Exception	Base class for all exceptions
EOFError	Raised when the input() method hits an "end of file" condition (EOF)
FloatingPointError	Raised when a floating point calculation fails
GeneratorExit	Raised when a generator is closed (with the close() method)
ImportError	Raised when an imported module does not exist
IndentationError	Raised when indentation is not correct
IndexError	Raised when an index of a sequence does not exist
KeyError	Raised when a key does not exist in a dictionary
KeyboardInterrupt	Raised when the user presses Ctrl+c, Ctrl+z or Delete
LookupError	Raised when errors raised cant be found
MemoryError	Raised when a program runs out of memory
NameError	Raised when a variable does not exist
NotImplementedError	Raised when an abstract method requires an inherited class to override the method
OSError	Raised when a system related operation causes an error
OverflowError	Raised when the result of a numeric calculation is too large
ReferenceError	Raised when a weak reference object does not exist
RuntimeError	Raised when an error occurs that do not belong to any specific exceptions
StopIteration	Raised when the next() method of an iterator has no further values
SyntaxError	Raised when a syntax error occurs
TabError	Raised when indentation consists of tabs or spaces
SystemError	Raised when a system error occurs
SystemExit	Raised when the sys.exit() function is called
TypeError	Raised when two different types are combined
UnboundLocalError	Raised when a local variable is referenced before assignment
UnicodeError	Raised when a unicode problem occurs
UnicodeEncodeError	Raised when a unicode encoding problem occurs
UnicodeDecodeError	Raised when a unicode decoding problem occurs
UnicodeTranslateError	Raised when a unicode translation problem occurs
ValueError	Raised when there is a wrong value in a specified data type
ZeroDivisionError	Raised when the second operator in a division is zero

-----

Python Glossary
This is a list of all the features explained in the Python Tutorial.

Feature	Description
Indentation	Indentation refers to the spaces at the beginning of a code line
Comments	Comments are code lines that will not be executed
Multiline Comments	How to insert comments on multiple lines
Creating Variables	Variables are containers for storing data values
Variable Names	How to name your variables
Assign Values to Multiple Variables	How to assign values to multiple variables
Output Variables	Use the print statement to output variables
String Concatenation	How to combine strings
Global Variables	Global variables are variables that belongs to the global

scope
Built-In Data Types Python has a set of built-in data types
Getting Data Type How to get the data type of an object
Setting Data Type How to set the data type of an object
Numbers There are three numeric types in Python
Int The integer number type
Float The floating number type
Complex The complex number type
Type Conversion How to convert from one number type to another
Random Number How to create a random number
Specify a Variable Type How to specify a certain data type for a variable
String Literals How to create string literals
Assigning a String to a Variable How to assign a string value to a variable
Multiline Strings How to create a multiline string
Strings are Arrays Strings in Python are arrays of bytes representing Unicode characters
Slicing a String How to slice a string
Negative Indexing on a String How to use negative indexing when accessing a string
String Length How to get the length of a string
Check In String How to check if a string contains a specified phrase
Format String How to combine two strings
Escape Characters How to use escape characters
Boolean Values True or False
Evaluate Booleans Evaluate a value or statement and return either True or False
Return Boolean Value Functions that return a Boolean value
Operators Use operator to perform operations in Python
Arithmetic Operators Arithmetic operator are used to perform common mathematical operations
Assignment Operators Assignment operators are use to assign values to variables
Comparison Operators Comparison operators are used to compare two values
Logical Operators Logical operators are used to combine conditional statements
Identity Operators Identity operators are used to see if two objects are in fact the same object
Membership Operators Membership operators are used to test is a sequence is present in an object
Bitwise Operators Bitwise operators are used to compare (binary) numbers
Lists A list is an ordered, and changeable, collection
Access List Items How to access items in a list
Change List Item How to change the value of a list item
Loop Through List Items How to loop through the items in a list
List Comprehension How use a list comprehensive
Check if List Item Exists How to check if a specified item is present in a list
List Length How to determine the length of a list
Add List Items How to add items to a list
Remove List Items How to remove list items
Copy a List How to copy a list
Join Two Lists How to join two lists
Tuple A tuple is an ordered, and unchangeable, collection
Access Tuple Items How to access items in a tuple
Change Tuple Item How to change the value of a tuple item
Loop List Items How to loop through the items in a tuple
Check if Tuple Item Exists How to check if a specified item is present in a tuple
Tuple Length How to determine the length of a tuple
Tuple With One Item How to create a tuple with only one item
Remove Tuple Items How to remove tuple items
Join Two Tuples How to join two tuples
Set A set is an unordered, and unchangeable, collection
Access Set Items How to access items in a set
Add Set Items How to add items to a set
Loop Set Items How to loop through the items in a set

Check if Set Item Exists	How to check if a item exists
Set Length	How to determine the length of a set
Remove Set Items	How to remove set items
Join Two Sets	How to join two sets
Dictionary	A dictionary is an unordered, and changeable, collection
Access Dictionary Items	How to access items in a dictionary
Change Dictionary Item	How to change the value of a dictionary item
Loop Dictionary Items	How to loop through the items in a tuple
Check if Dictionary Item Exists	How to check if a specified item is present in a dictionary
Dictionary Length	How to determine the length of a dictionary
Add Dictionary Item	How to add an item to a dictionary
Remove Dictionary Items	How to remove dictionary items
Copy Dictionary	How to copy a dictionary
Nested Dictionaries	A dictionary within a dictionary
If Statement	How to write an if statement
If Indentation	If statements in Python relies on indentation (whitespace at the beginning of a line)
Elif	elif is the same as "else if" in other programming languages
Else	How to write an if...else statement
Shorthand If	How to write an if statement in one line
Shorthand If Else	How to write an if...else statement in one line
If AND	Use the and keyword to combine if statements
If OR	Use the or keyword to combine if statements
If NOT	Use the not keyword to reverse the condition
Nested If	How to write an if statement inside an if statement
The pass Keyword in If	Use the pass keyword inside empty if statements
While	How to write a while loop
While Break	How to break a while loop
While Continue	How to stop the current iteration and continue wit the next
While Else	How to use an else statement in a while loop
For	How to write a for loop
Loop Through a String	How to loop through a string
For Break	How to break a for loop
For Continue	How to stop the current iteration and continue wit the next
Looping Through a range	How to loop through a range of values
For Else	How to use an else statement in a for loop
Nested Loops	How to write a loop inside a loop
For pass	Use the pass keyword inside empty for loops
Function	How to create a function in Python
Call a Function	How to call a function in Python
Function Arguments	How to use arguments in a function
*args	To deal with an unknown number of arguments in a function, use the * symbol before the parameter name
Keyword Arguments	How to use keyword arguments in a function
**kwargs	To deal with an unknown number of keyword arguments in a function, use the * symbol before the parameter name
Default Parameter Value	How to use a default parameter value
Passing a List as an Argument	How to pass a list as an argument
Function Return Value	How to return a value from a function
The pass Statement in Functions	Use the pass statement in empty functions
Function Recursion	Functions that can call itself is called recursive functions
Lambda Function	How to create anonymous functions in Python
Why Use Lambda Functions	Learn when to use a lambda function or not
Array	Lists can be used as Arrays
What is an Array	Arrays are variables that can hold more than one value
Access Arrays	How to access array items
Array Length	How to get the length of an array
Looping Array Elements	How to loop through array elements
Add Array Element	How to add elements from an array
Remove Array Element	How to remove elements from an array
Array Methods	Python has a set of Array/Lists methods
Class	A class is like an object constructor

Create Class	How to create a class
The Class __init__() Function	The __init__() function is executed when the class is initiated
Object Methods	Methods in objects are functions that belongs to the object
self	The self parameter refers to the current instance of the class
Modify Object Properties	How to modify properties of an object
Delete Object Properties	How to modify properties of an object
Delete Object	How to delete an object
Class pass Statement	Use the pass statement in empty classes
Create Parent Class	How to create a parent class
Create Child Class	How to create a child class
Create the __init__() Function	How to create the __init__() function
super Function	The super() function make the child class inherit the parent class
Add Class Properties	How to add a property to a class
Add Class Methods	How to add a method to a class
Iterators	An iterator is an object that contains a countable number of values
Iterator vs Iterable	What is the difference between an iterator and an iterable
Loop Through an Iterator	How to loop through the elements of an iterator
Create an Iterator	How to create an iterator
StopIteration	How to stop an iterator
Global Scope	When does a variable belong to the global scope?
Global Keyword	The global keyword makes the variable global
Create a Module	How to create a module
Variables in Modules	How to use variables in a module
Renaming a Module	How to rename a module
Built-in Modules	How to import built-in modules
Using the dir() Function	List all variable names and function names in a module
Import From Module	How to import only parts from a module
Datetime Module	How to work with dates in Python
Date Output	How to output a date
Create a Date Object	How to create a date object
The strftime Method	How to format a date object into a readable string
Date Format Codes	The datetime module has a set of legal format codes
JSON	How to work with JSON in Python
Parse JSON	How to parse JSON code in Python
Convert into JSON	How to convert a Python object in to JSON
Format JSON	How to format JSON output with indentations and line breaks
Sort JSON	How to sort JSON
RegEx Module	How to import the regex module
RegEx Functions	The re module has a set of functions
Metacharacters in RegEx	Metacharacters are characters with a special meaning
RegEx Special Sequences	A backslash followed by a a character has a special meaning
RegEx Sets	A set is a set of characters inside a pair of square brackets with a special meaning
RegEx Match Object	The Match Object is an object containing information about the search and the result
Install PIP	How to install PIP
PIP Packages	How to download and install a package with PIP
PIP Remove Package	How to remove a package with PIP
Error Handling	How to handle errors in Python
Handle Many Exceptions	How to handle more than one exception
Try Else	How to use the else keyword in a try statement
Try Finally	How to use the finally keyword in a try statement
raise	How to raise an exception in Python

Python Random Module
Python has a built-in module that you can use to make random numbers.

The random module has a set of methods:

Method	Description
seed()	Initialize the random number generator
getstate()	Returns the current internal state of the random number generator
setstate()	Restores the internal state of the random number generator
getrandbits()	Returns a number representing the random bits
randrange()	Returns a random number between the given range
randint()	Returns a random number between the given range
choice()	Returns a random element from the given sequence
choices()	Returns a list with a random selection from the given sequence
shuffle()	Takes a sequence and returns the sequence in a random order
sample()	Returns a given sample of a sequence
random()	Returns a random float number between 0 and 1
uniform()	Returns a random float number between two given parameters
triangular()	Returns a random float number between two given parameters, you can also set a mode parameter to specify the midpoint between the two other parameters
betavariate()	Returns a random float number between 0 and 1 based on the Beta distribution (used in statistics)
expovariate()	Returns a random float number based on the Exponential distribution (used in statistics)
gammavariate()	Returns a random float number based on the Gamma distribution (used in statistics)
gauss()	Returns a random float number based on the Gaussian distribution (used in probability theories)
lognormvariate()	Returns a random float number based on a log-normal distribution (used in probability theories)
normalvariate()	Returns a random float number based on the normal distribution (used in probability theories)
vonmisesvariate()	Returns a random float number based on the von Mises distribution (used in directional statistics)
paretovariate()	Returns a random float number based on the Pareto distribution (used in probability theories)
weibullvariate()	Returns a random float number based on the Weibull distribution (used in statistics)

Python Requests Module
ExampleGet your own Python Server
Make a request to a web page, and print the response text:

```
import requests

x = requests.get('https://w3schools.com/python/demopage.htm')

print(x.text)
```

Definition and Usage
The requests module allows you to send HTTP requests using Python.

The HTTP request returns a Response Object with all the response data (content, encoding, status, etc).

Download and Install the Requests Module
Navigate your command line to the location of PIP, and type the following:

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip install requests
```

Syntax
requests.methodname(params)
Methods

Method	Description
delete(url, args)	Sends a DELETE request to the specified url

get(url, params, args) Sends a GET request to the specified url
head(url, args) Sends a HEAD request to the specified url
patch(url, data, args) Sends a PATCH request to the specified url
post(url, data, json, args) Sends a POST request to the specified url
put(url, data, args) Sends a PUT request to the specified url
request(method, url, args) Sends a request of the specified method to the specified url

-----

Python statistics Module
Python statistics Module
Python has a built-in module that you can use to calculate mathematical statistics of numeric data.

The statistics module was new in Python 3.4.

Statistics Methods
Method Description
statistics.harmonic_mean() Calculates the harmonic mean (central location) of the given data
statistics.mean() Calculates the mean (average) of the given data
statistics.median() Calculates the median (middle value) of the given data
statistics.median_grouped() Calculates the median of grouped continuous data
statistics.median_high() Calculates the high median of the given data
statistics.median_low() Calculates the low median of the given data
statistics.mode() Calculates the mode (central tendency) of the given numeric or nominal data
statistics.pstdev() Calculates the standard deviation from an entire population
statistics.stdev() Calculates the standard deviation from a sample of data
statistics.pvariance() Calculates the variance of an entire population
statistics.variance() Calculates the variance from a sample of data

-----

Python math Module
Python math Module
Python has a built-in module that you can use for mathematical tasks.

The math module has a set of methods and constants.

Math Methods
Method Description
math.acos() Returns the arc cosine of a number
math.acosh() Returns the inverse hyperbolic cosine of a number
math.asin() Returns the arc sine of a number
math.asinh() Returns the inverse hyperbolic sine of a number
math.atan() Returns the arc tangent of a number in radians
math.atan2() Returns the arc tangent of y/x in radians
math.atanh() Returns the inverse hyperbolic tangent of a number
math.ceil() Rounds a number up to the nearest integer
math.comb() Returns the number of ways to choose k items from n items without repetition and order
math.copysign() Returns a float consisting of the value of the first parameter and the sign of the second parameter
math.cos() Returns the cosine of a number
math.cosh() Returns the hyperbolic cosine of a number
math.degrees() Converts an angle from radians to degrees
math.dist() Returns the Euclidean distance between two points (p and q), where p and q are the coordinates of that point
math.erf() Returns the error function of a number
math.erfc() Returns the complementary error function of a number
math.exp() Returns E raised to the power of x

math.expm1()	Returns Ex - 1
math.fabs() Returns the absolute value of a number
math.factorial()	Returns the factorial of a number
math.floor()	Rounds a number down to the nearest integer
math.fmod() Returns the remainder of x/y
math.frexp()	Returns the mantissa and the exponent, of a specified number
math.fsum() Returns the sum of all items in any iterable (tuples, arrays, lists, etc.)
math.gamma()	Returns the gamma function at x
math.gcd()	Returns the greatest common divisor of two integers
math.hypot()	Returns the Euclidean norm
math.isclose()	Checks whether two values are close to each other, or not
math.isfinite()	Checks whether a number is finite or not
math.isinf()	Checks whether a number is infinite or not
math.isnan()	Checks whether a value is NaN (not a number) or not
math.isqrt()	Rounds a square root number downwards to the nearest integer
math.ldexp()	Returns the inverse of math.frexp() which is x * (2**i) of the given numbers x and i
math.lgamma()	Returns the log gamma value of x
math.log()	Returns the natural logarithm of a number, or the logarithm of number to base
math.log10()	Returns the base-10 logarithm of x
math.log1p()	Returns the natural logarithm of 1+x
math.log2() Returns the base-2 logarithm of x
math.perm() Returns the number of ways to choose k items from n items with order and without repetition
math.pow()	Returns the value of x to the power of y
math.prod() Returns the product of all the elements in an iterable
math.radians()	Converts a degree value into radians
math.remainder()	Returns the closest value that can make numerator completely divisible by the denominator
math.sin()	Returns the sine of a number
math.sinh() Returns the hyperbolic sine of a number
math.sqrt() Returns the square root of a number
math.tan()	Returns the tangent of a number
math.tanh() Returns the hyperbolic tangent of a number
math.trunc()	Returns the truncated integer parts of a number

Math Constants
Constant	Description
math.e	Returns Euler's number (2.7182...)
math.inf	Returns a floating-point positive infinity
math.nan	Returns a floating-point NaN (Not a Number) value
math.pi	Returns PI (3.1415...)
math.tau	Returns tau (6.2831...)

Python cmath Module
Python cmath Module
Python has a built-in module that you can use for mathematical tasks for complex numbers.

The methods in this module accepts int, float, and complex numbers. It even accepts Python objects that has a __complex__() or __float__() method.

The methods in this module almost always return a complex number. If the return value can be expressed as a real number, the return value has an imaginary part of 0.

The cmath module has a set of methods and constants.

cMath Methods
Method	Description

```
cmath.acos(x)     Returns the arc cosine value of x
cmath.acosh(x)    Returns the hyperbolic arc cosine of x
cmath.asin(x)     Returns the arc sine of x
cmath.asinh(x)    Returns the hyperbolic arc sine of x
cmath.atan(x)     Returns the arc tangent value of x
cmath.atanh(x)    Returns the hyperbolic arctangent value of x
cmath.cos(x)      Returns the cosine of x
cmath.cosh(x)     Returns the hyperbolic cosine of x
cmath.exp(x)      Returns the value of Ex, where E is Euler's number
(approximately 2.718281...), and x is the number passed to it
cmath.isclose()   Checks whether two values are close, or not
cmath.isfinite(x) Checks whether x is a finite number
cmath.isinf(x)    Check whether x is a positive or negative infinty
cmath.isnan(x)    Checks whether x is NaN (not a number)
cmath.log(x[, base])    Returns the logarithm of x to the base
cmath.log10(x)    Returns the base-10 logarithm of x
cmath.phase()     Return the phase of a complex number
cmath.polar()     Convert a complex number to polar coordinates
cmath.rect()      Convert polar coordinates to rectangular form
cmath.sin(x)      Returns the sine of x
cmath.sinh(x)     Returns the hyperbolic sine of x
cmath.sqrt(x)     Returns the square root of x
cmath.tan(x)      Returns the tangent of x
cmath.tanh(x)     Returns the hyperbolic tangent of x

cMath Constants
Constant    Description
cmath.e     Returns Euler's number (2.7182...)
cmath.inf   Returns a floating-point positive infinity value
cmath.infj  Returns a complex infinity value
cmath.nan   Returns floating-point NaN (Not a Number) value
cmath.nanj  Returns coplext NaN (Not a Number) value
cmath.pi    Returns PI (3.1415...)
cmath.tau   Returns tau (6.2831...)

-----

How to Remove Duplicates From a Python List
Learn how to remove duplicates from a List in Python.

ExampleGet your own Python Server
Remove any duplicates from a List:

mylist = ["a", "b", "a", "c", "c"]
mylist = list(dict.fromkeys(mylist))
print(mylist)
Example Explained
First we have a List that contains duplicates:

A List with Duplicates
mylist = ["a", "b", "a", "c", "c"]
mylist = list(dict.fromkeys(mylist))
print(mylist)
Create a dictionary, using the List items as keys. This will automatically
remove any duplicates because dictionaries cannot have duplicate keys.

Create a Dictionary
mylist = ["a", "b", "a", "c", "c"]
mylist = list( dict.fromkeys(mylist) )
print(mylist)
Then, convert the dictionary back into a list:

Convert Into a List
mylist = ["a", "b", "a", "c", "c"]
```

```
mylist = list( dict.fromkeys(mylist) )
print(mylist)
```
Now we have a List without any duplicates, and it has the same order as the original List.

Print the List to demonstrate the result

Print the List
```
mylist = ["a", "b", "a", "c", "c"]
mylist = list(dict.fromkeys(mylist))
print(mylist)
```


Create a Function
If you like to have a function where you can send your lists, and get them back without duplicates, you can create a function and insert the code from the example above.

Example
```
def my_function(x):
  return list(dict.fromkeys(x))

mylist = my_function(["a", "b", "a", "c", "c"])

print(mylist)
```
Example Explained
Create a function that takes a List as an argument.

Create a Function
```
def my_function(x):
  return list(dict.fromkeys(x))

mylist = my_function(["a", "b", "a", "c", "c"])

print(mylist)
```
Create a dictionary, using this List items as keys.

Create a Dictionary
```
def my_function(x):
  return list( dict.fromkeys(x) )

mylist = my_function(["a", "b", "a", "c", "c"])

print(mylist)
```
Convert the dictionary into a list.

Convert Into a List
```
def my_function(x):
  return list( dict.fromkeys(x) )

mylist = my_function(["a", "b", "a", "c", "c"])

print(mylist)
```
Return the list

Return List
```
def my_function(x):
  return list(dict.fromkeys(x))

mylist = my_function(["a", "b", "a", "c", "c"])

print(mylist)
```
Call the function, with a list as a parameter:

Call the Function
```
def my_function(x):
  return list(dict.fromkeys(x))

mylist = my_function(["a", "b", "a", "c", "c"])

print(mylist)
```
Print the result:

Print the Result
```
def my_function(x):
  return list(dict.fromkeys(x))

mylist = my_function(["a", "b", "a", "c", "c"])

print(mylist)
```

How to Reverse a String in Python
Learn how to reverse a String in Python.

There is no built-in function to reverse a String in Python.

The fastest (and easiest?) way is to use a slice that steps backwards, -1.

ExampleGet your own Python Server
Reverse the string "Hello World":

```
txt = "Hello World"[::-1]
print(txt)
```
Example Explained
We have a string, "Hello World", which we want to reverse:

The String to Reverse
```
txt = "Hello World" [::-1]
print(txt)
```
Create a slice that starts at the end of the string, and moves backwards.

In this particular example, the slice statement [::-1] means start at the end of the string and end at position 0, move with the step -1, negative one, which means one step backwards.

Slice the String
```
txt = "Hello World" [::-1]
print(txt)
```
Now we have a string txt that reads "Hello World" backwards.

Print the String to demonstrate the result

Print the List
```
txt = "Hello World"[::-1]
print(txt)
```


Create a Function
If you like to have a function where you can send your strings, and return them backwards, you can create a function and insert the code from the example above.

Example
```
def my_function(x):
  return x[::-1]

mytxt = my_function("I wonder how this text looks like backwards")
```

```
print(mytxt)
```
Example Explained
Create a function that takes a String as an argument.

Create a Function
```
def my_function(x):
  return x[::-1]

mytxt = my_function("I wonder how this text looks like backwards")

print(mytxt)
```
Slice the string starting at the end of the string and move backwards.

Slice the String
```
def my_function(x):
  return x [::-1]

mytxt = my_function("I wonder how this text looks like backwards")

print(mytxt)
```
Return the backward String

Return the String
```
def my_function(x):
  return x[::-1]

mytxt = my_function("I wonder how this text looks like backwards")

print(mytxt )
```
Call the function, with a string as a parameter:

Call the Function
```
def my_function(x):
  return x[::-1]

mytxt = my_function("I wonder how this text looks like backwards")

print(mytxt)
```
Print the result:

Print the Result
```
def my_function(x):
  return x[::-1]

mytxt = my_function("I wonder how this text looks like backwards")

print(mytxt)
```

How to Add Two Numbers in Python
Learn how to add two numbers in Python.

Use the + operator to add two numbers:

ExampleGet your own Python Server
```
x = 5
y = 10
print(x + y)
```
Add Two Numbers with User Input
In this example, the user must input two numbers. Then we print the sum by calculating (adding) the two numbers:

Example

```
x = input("Type a number: ")
y = input("Type another number: ")

sum = int(x) + int(y)

print("The sum is: ", sum)
```
