Python

Dr. Panem Charanarur

Overview

- History

- Installing & Running Python

Brief History of Python

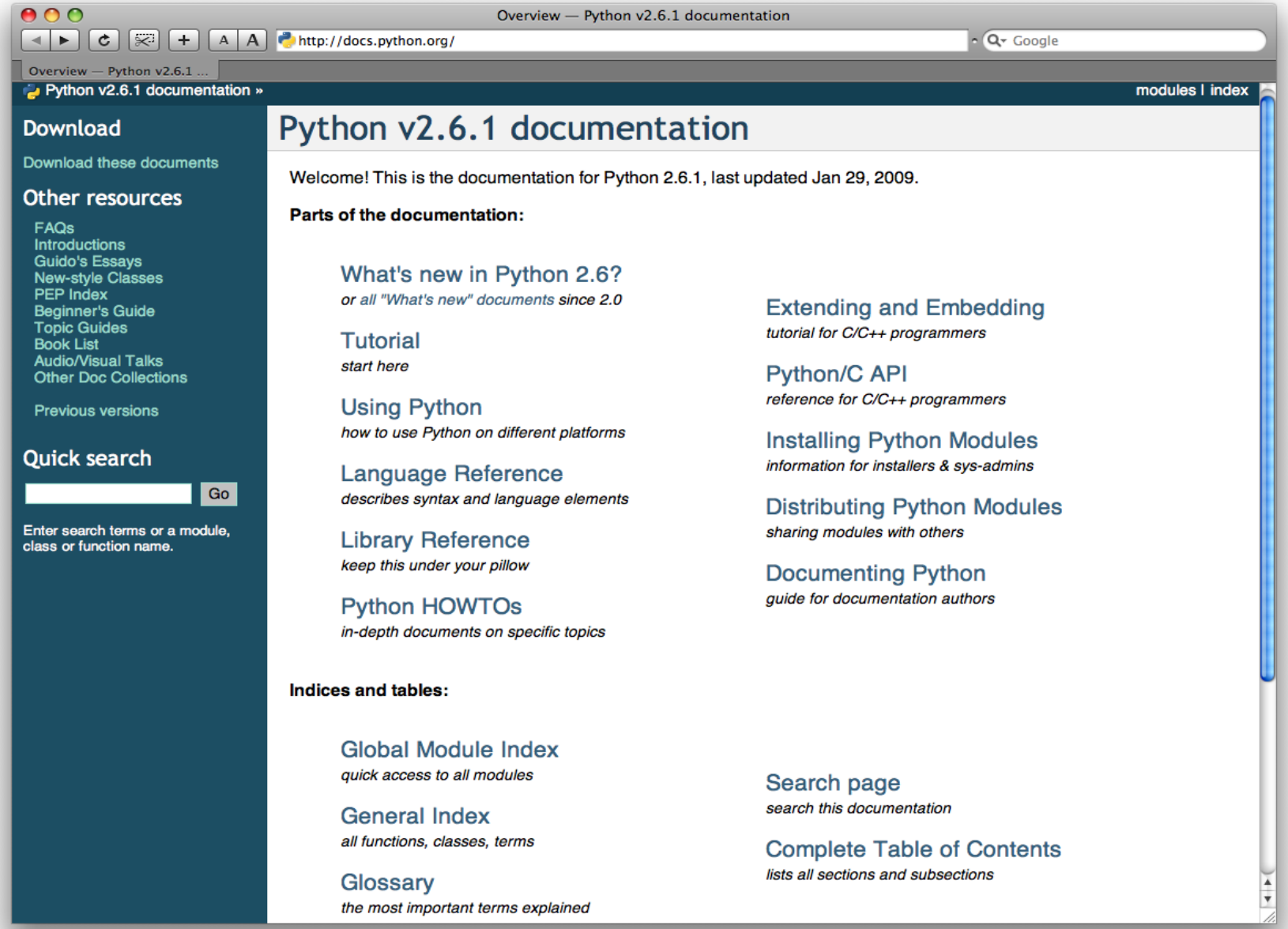
- Invented in the Netherlands, early 90s by Guido van Rossum
- Named after Monty Python
- Open sourced from the beginning
- Considered a scripting language, but is much more
- Scalable, object oriented and functional from the beginning
- Used by Google from the beginning
- Increasingly popular

Python's Benevolent Dictator For Life

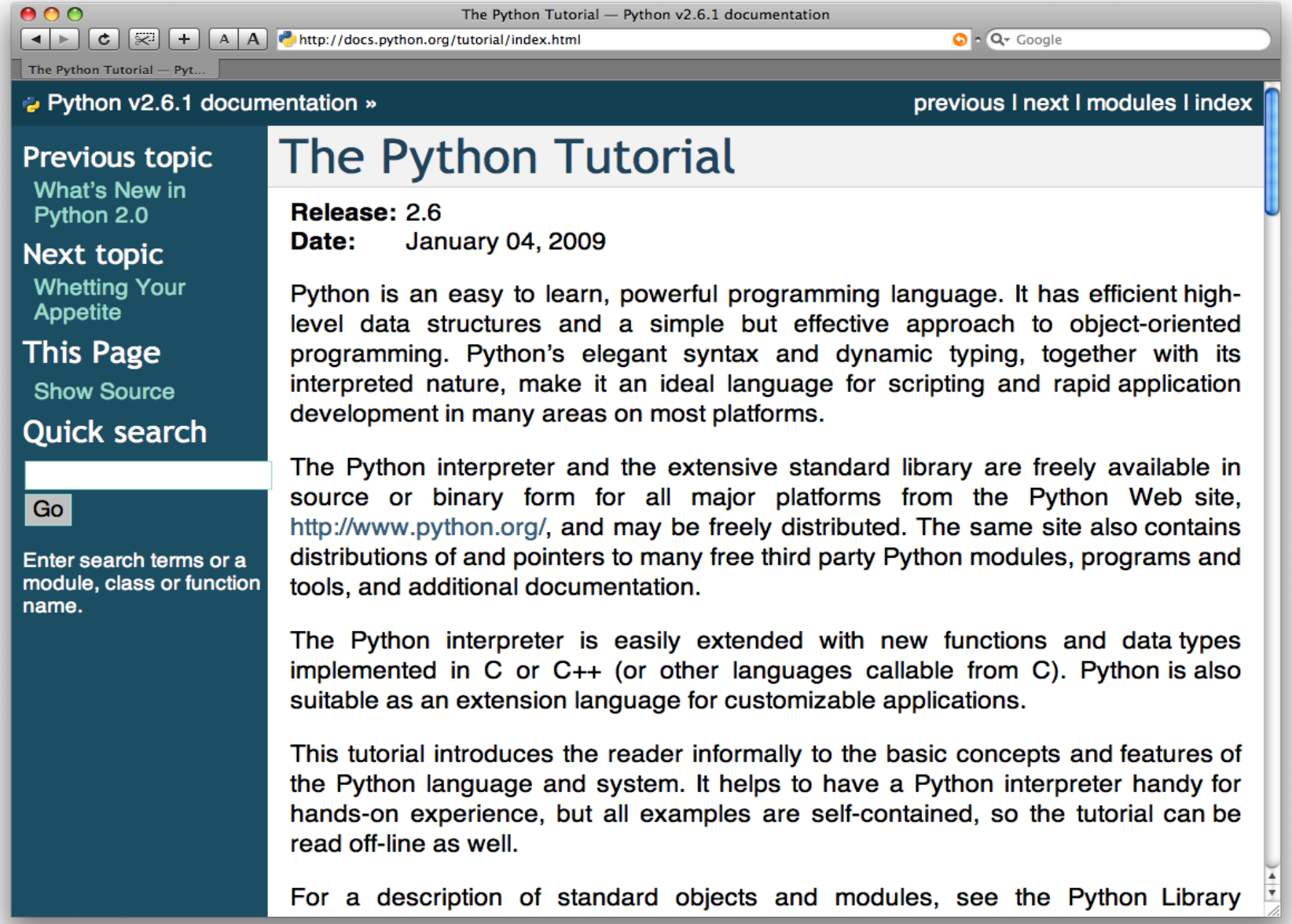
"Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered."

- Guido van Rossum

http://docs.python.org/



The Python tutorial is good!



Warmup

Windows:

- **Open anaconda prompt**
- **Type `conda -V`**
- **If you get an error, install Anaconda:** https://docs.anaconda.com/anaconda/install/windows/
  - #8 is important: **DO NOT** add to your path
- **If no error, consider upgrading conda:**

```
conda update conda
```

- **Clone** https://github.com/Harvard-IACS/2019-CS109B **(or pull the latest if you've already cloned)**

Mac:

- **Open a terminal**
- **Type `conda -V`**
- **If you get an error, install Anaconda:**
- https://docs.anaconda.com/anaconda/install/mac-os/
- **If no error, consider upgrading conda:**

```
conda update conda
```

- **Clone** https://github.com/Harvard-IACS/2019-CS109B **(or pull the latest if you've already cloned)**

Goals

- **Set up the tools you'll need for CS109b**
  - In a way that won't mess up your other classes
- **Teach a workflow that will *keep* your installs tidy**
- **User-level understanding of why 'environments' are helpful**
- ***Stretch*: Ability to produce conda environments for future projects**

Basic Python Commands

1. **Comments**: # symbol is being used for comments in python. For multiline comments, you have to use “”” symbols or enclosing the comment in the “”” symbol.

- **Example:**
  print “Hello World” **#** this is the comment section.
- **Example:**
  “”” This is Hello world project.”””

2. **Type function**: These Python Commands are used to check the type of variable and used inbuilt functions to check.

- **Example:**
  type (20), its type is int.
  >>> type (20) < type ‘int’ >
- **Example:**
  type (-1 +j), its type is complex
  >>> type (-1+j)
  < type ‘complex’ >

3. **Strings:** It is mainly enclosed in double-quotes.

- **Example:**
  type (“hello World”), type is string
  >>> type (“hello World”)
  < type ‘str’ >

4. Lists: Lists are mainly enclosed in square bracket. [ ]

- **Example**:
  type ([1, 2]), type is list
  >>> type ([1, 2, 3])
  < type 'List' >

5.Tuple: Tuple are mainly enclosed in parenthesis. ( )

- **Example:**
  type (1, 2, 3), a type is a tuple.
  >>> type ((1, 2, 3))
  <type 'tuple' >

6. Range: This function is used to create the list of integers. Range(10)

- **Example:**
  >>> range (10)
  Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- **Example:** **range(1,10)**
  >>> range (1,10)
  Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]

7. Boolean values: This data type helps in retrieving the data in True or false form.

- **Example:**
  >>> True
  True
  >>> type (True)
  < type 'bool' >
- **Example:**
  >>> False
  False
  >>> type (False)
  < type 'bool' >

8. Operator: Different operator is used for the different functions like division, multiply, addition and subtraction.

- **Example:**
  >>> 16/2 16/2=8
  8
  Example:
  >>> 2 * * ½ 2 **, -, +
  1

9. Variable and assignment: The assignment statement has variable = expression. Single '=' is used for assignment, and double '= =' is used to test for quality.

- **Example:**
  >>> X= 235
  >>> print X
  235
  >>> Z= 2* X
  >>> print Z
  470

10. Comparison operators: To compare the two values and the result of the comparison is always boolean value.

- **Example:**
  >>> 2 < 3
  True

11. Conditional/ decisions: It is used to make out the decision between two or more values like if-else

- **Example:**
  if x=0:
  Print “Hello, world.”
  Else:
  Print “Hello, world in Else.”

12. For Loop: This Python command is used when iteration and action consist of the same elements.

- **Example:**
  for x in [1, 2, 3, 4, 5, 6]:
  Print x;

13. While loop: While loop will never be executed if the condition evaluates to false for the first time.

- **Example:**
  x =0
  while x<10:
  Print x,
  X= x+2

14. Else in loop: Loop have optional else for execution.

- **Example:**
  for x in [1, 3, 5, 7, 9, 11]:
  Print x
  Else:
  Print “In Else”

15. Break, continue statement: break statement is used to exit out the loop when particular output is achieved; continue is used to continue with the next iteration of a loop.

- **Example:**
  if x==0:
  Print “X is 0”
  Break
  Else:
  Print “X is greater than 0.”

16. Lists: It is the finite number of items, and by assigning a value to list the list value will get changed.

- **Example:**
  >>> a = [1, “JAY”, 34] >>> a [0] 1
  >>> a [0] = 101
  >>> a
  [101, “JAY”, 34]

17. Length of list: To know the length if list.

- **Example:**
  >>> X = [1, 4, 67,9] >>> len (X)
  4

18. Sublists: It will give you the values between the mentioned start index and the end index.

- **Example:**
  x [start : end] >>> X [1, 3, 4,6, 7, 8, 9, 0, 2] >>> X [2:5] [4, 6, 7]

19. Joining two list: + operator is being used to concatenate 2 lists.

- **Example:**
  >>> [2, 1, 5] + [0, 4,6,7] [2, 1, 5, 0, 4, 6,7]

20. Strings: It is used to check the index to know the character written in string.

- **Example:**
  >>> x= “hello, world” “”
  >>> x [2] ‘l’
  >>> x [5] ‘o’

21. List methods: The different methods available to in list to perform the function.

- **Example:**
  X [1, 2,3,4,5]

>>> X.append (7)

>>> x
[1, 2, 3, 4, 5, 7]

>>>X.insert (0, 0)

>>> x
[0, 1, 2, 3, 4, 5]

>>> X.remove (2)

>>> x
[0, 1, 3, 4, 5]

>>> X.pop (1)

>>> x
[2,3,4,5]

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.

- Example If statement:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

- Elif: The elif keyword is pythons 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")
```

- 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")
```

- 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")
```

- 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.
- Example

Print each fruit in a fruit list:

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

- 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)
```

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

Create a List:

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

- 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]
```

- 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.
- Dictionaries are written with curly brackets, and have keys and values
- Example

Create and print a dictionary:

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

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

```
print(len(thisdict))
```

- Dictionary Items - Data Types
- ExampleString, int, boolean, and list data types:

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

Network: The standard library of Python has full support for network protocols, encoding, and decoding of data and other networking concepts, and it is simpler to write network programs in Python

- There are two levels of network service access in Python. These are:
- Low-Level Access
- High-Level Access
- In the first case, programmers can use and access the basic socket support for the operating system using Python's libraries, and programmers can implement both connection-less and connection-oriented protocols for programming.
- Sockets use protocols for determining the connection type for port-to-port communication between client and server machines. The protocols are used for:
- Domain Name Servers (DNS)
- IP addressing
- E-mail
- FTP (File Transfer Protocol) etc.

- Python has a socket method that let programmers' set-up different types of socket virtually. The syntax for the socket method is:
- Syntax:

g = socket.socket (socket_family, type_of_socket, protocol=value)

For example, if we want to establish a TCP socket, we can write the following code snippet:

- Example:

```
# imports everything from 'socket'
from socket import *
# use socket.socket() - function
tcp1=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

you defined the socket, you can use several methods to manage the connections. Some of the important server socket methods are:

- **listen()**: is used to establish and start TCP listener.
- **bind()**: is used to bind-address (host-name, port number) to the socket.
- **accept()**: is used to TCP client connection until the connection arrives.
- **connect()**: is used to initiate TCP server connection.
- **send()**: is used to send TCP messages.
- **recv()**: is used to receive TCP messages.
- **sendto()**: is used to send UDP messages
- **close()**: is used to close a socket.

- **Selection:**
- Python's select() function is a direct interface to the underlying operating system implementation.
- It monitors sockets, open files, and pipes (anything with a fileno() method that returns a valid file descriptor) until they become readable or writable, or a communication error occurs. select() makes it easier to monitor multiple connections at the same time, and is more efficient than writing a polling loop in Python using socket timeouts, because the monitoring happens in the operating system network layer, instead of the interpreter.
- **Note:**Using Python's file objects with select() works for Unix, but is not supported under Windows.
- **Example:**

```
import select
import socket
 import sys
import Queue
# Create a TCP/IP socket
 server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setblocking(0)
# Bind the socket to the port
server_address = ('localhost', 10000)
print >>sys.stderr, 'starting up on %s port %s' %
server_address server.bind(server_address)
# Listen for incoming connections
server.listen(5)
```

- **Functions**: In Python, a defined function's declaration begins with the keyword def and followed by the function name.
- The function may take arguments(s) as input within the opening and closing parentheses, just after the function name followed by a colon.
- After defining the function name and arguments(s) a block of program statement(s) start at the next line and these statement(s) must be indented.
- Here is the syntax of a user defined function.
- **Syntax:**

def function_name(argument1, argument2, ...) :

statement_1 statement_2 ....

Example:

```
def nsquare(x, y):
return (x*x + 2*x*y + y*y)
print("The square of the sum of 2 and 3 is : ", nsquare(2, 3))
```

- **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__()

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))
```

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 Classes:

To create a class, use the keyword class.

Example1:Create a class named MyClass, with a property named x:

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

Example 2: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)
```

Python Objects:

Example1:Create an object named p1, and print the value of x:

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

Example 2: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()
```

- **Introduction of Anaconda:**
- Anaconda is a data science platform for data scientists, IT professionals, and business leaders. It is a distribution of Python, R, etc. With more than 300 packages for data science, it quickly became one of the best platforms for any project
- https://docs.anaconda.com/anaconda/install/windows/

NumPy:

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.

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++.

- Example

```
import numpy
arr = numpy.array([1, 2, 3, 4, 5])
print(arr)
```

- Example

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

- Example

```
import numpy as np
print(np.__version__)
```

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.

Example

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

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.

Example

Get the first element from the following array:

```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[0])
```

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

Example
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])
```

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

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.

Example
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)
```

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

Example
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)
```

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.

Example

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)
```

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.

Example

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)
```

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.
Example
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)
```

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.
Example
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)
```

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.

Example

Sort the array:

```
import numpy as np

arr = np.array([3, 2, 0, 1])

print(np.sort(arr))
```

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.

Example

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)
```

Pandas

Pandas is a Python library.

Pandas is used to analyze data.

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.

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

Example

```
import pandas

mydataset = {
  'cars':
["BMW", "Volvo", "F
ord"],
  'passings': [3, 7, 2]
}

myvar =
pandas.DataFrame(m
ydataset)

print(myvar)
```

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.

Example

Create a simple Pandas Series from a list:

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```

Create Labels

With the index argument, you can name your own labels.

Example

Create you own labels:

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
```

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:

```
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories)

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:

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

myvar = pd.DataFrame(data)

print(myvar)
```

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.

Example
Create a simple Pandas DataFrame:

```
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)
```

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])
```

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)
```

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"])
```

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)
```

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'.

Example

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)
```

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'.

Example

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)
```

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.

Example

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()
```

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()
```

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')
```

Matplotlib

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.

Matplotlib Pyplot

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

Now the Pyplot package can be referred to as plt.

Example

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()
```

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.

Example

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()
```

Markers

You can use the keyword argument marker to emphasize each point with a specified marker:

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, marker = 'o')
plt.show()
```

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

Line Syntax	Description
'-'	Solid line
':'	Dotted line
'--'	Dashed line
'-.'	Dashed/dotted line

Color Syntax	Description
'r'	Red
'g'	Green
'b'	Blue
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

Matplotlib GridMatplotlib SubplotsMatplotlib ScatterMatplotlib BarsMatplotlib HistogramsMatplotlib Pie Charts