PILLAI COLLEGE OF ENGINEERING A+ NAAC

Python Workshop

Presented by Vivek Vaishya

Why Python is even a thing?

Web and Internet Development

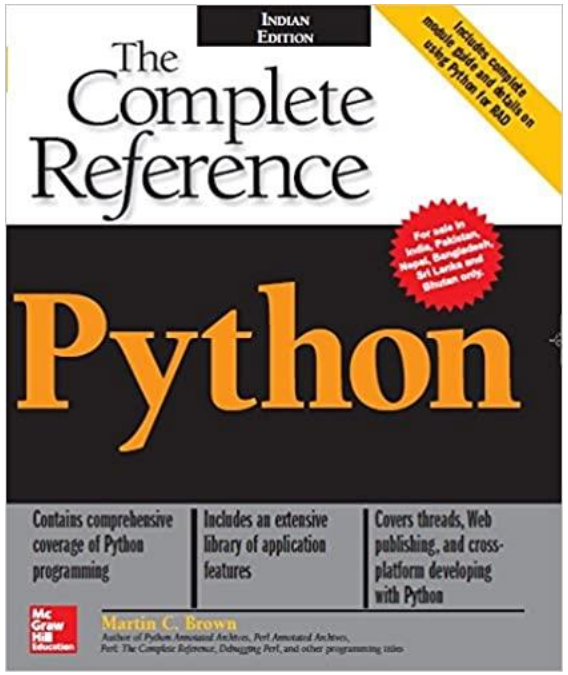
PYTORCH

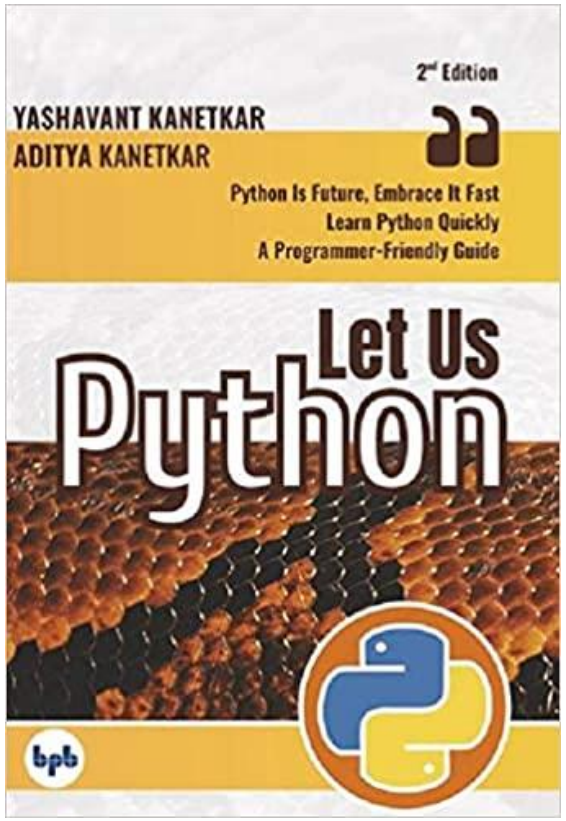
Pandas

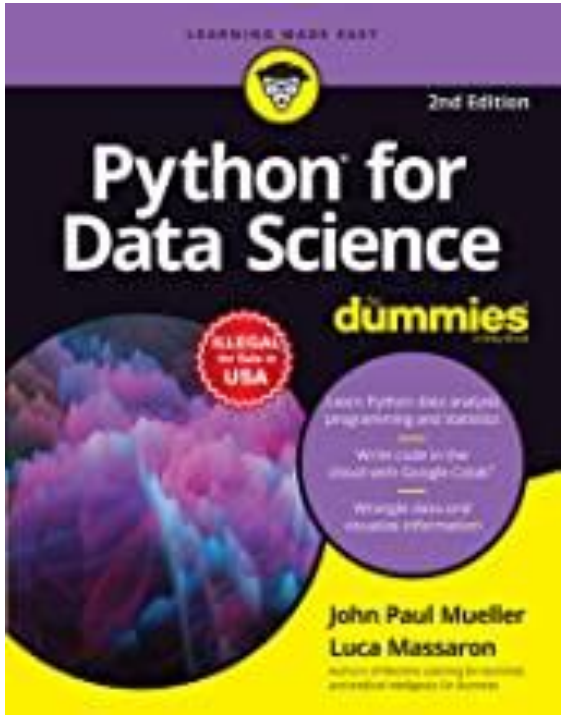
TensorFlow

SciPy

Scientific and Machine Learning

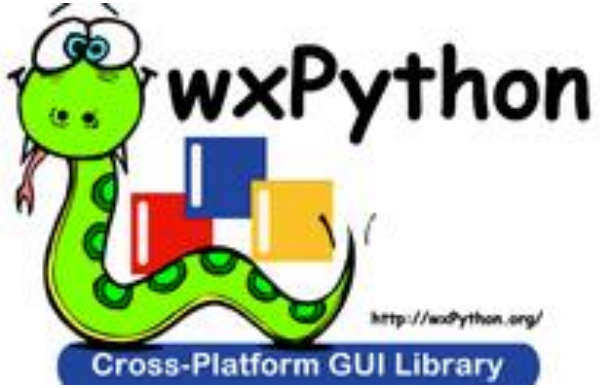

Indian Edition
The Complete Reference
Python
Contains comprehensive coverage of Python programming
Includes an extensive library of application features
Covers threads, Web publishing, and cross-platform developing with Python
Martin C. Brown


2nd Edition
YASHAVANT KANETKAR
ADITYA KANETKAR
Python Is Future, Embrace It Fast
Learn Python Quickly
A Programmer-Friendly Guide
Let Us Python
bpb

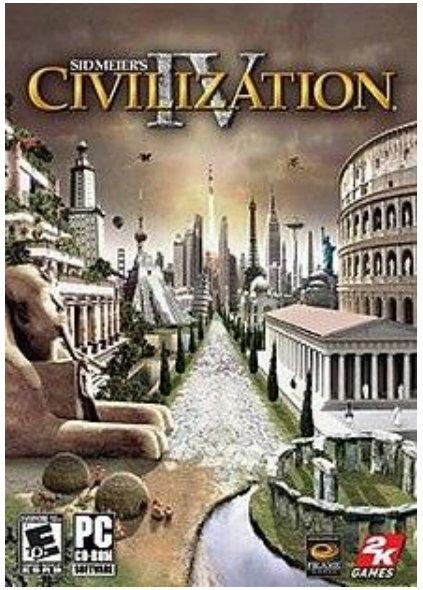

2nd Edition
Python for Data Science
dummies
John Paul Mueller
Luca Massaron


THIRD EDITION
PYTHON PROGRAMMING:
AN INTRODUCTION TO COMPUTER SCIENCE
JOHN ZELLE

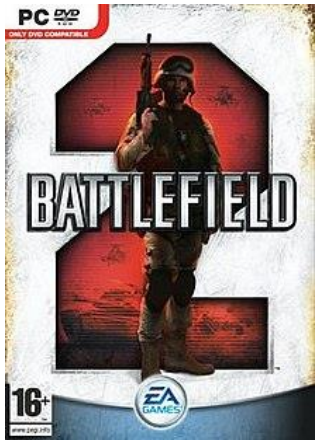
Education



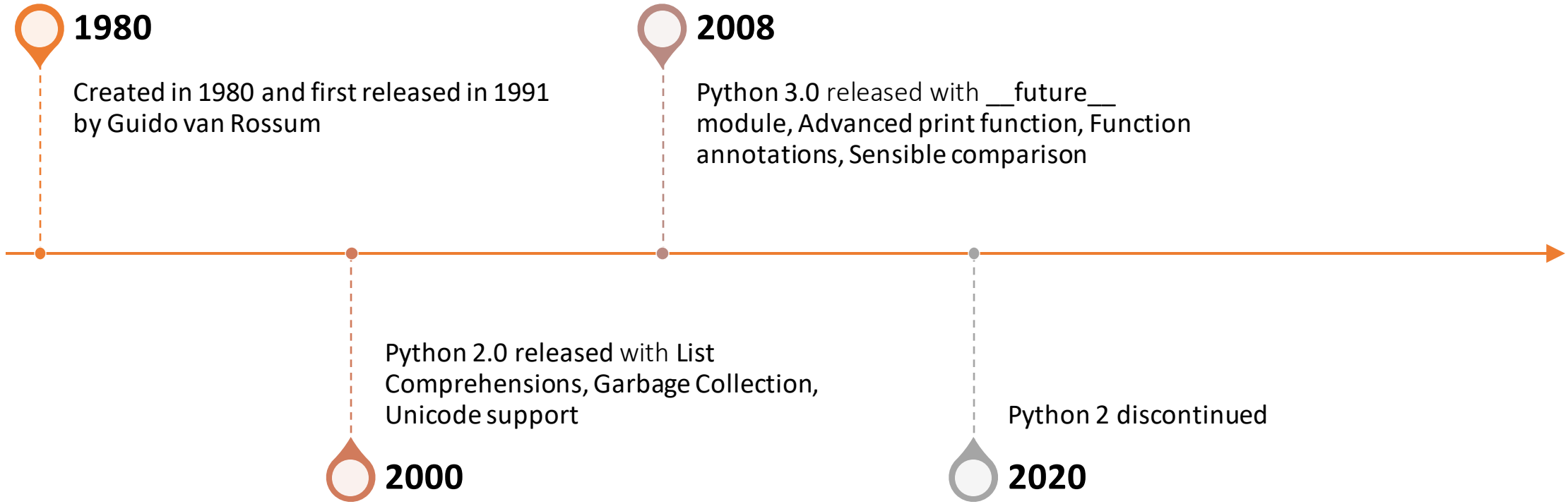
Desktop GUIs



Software and Game Development



The tail of Python


1980
Created in 1980 and first released in 1991 by Guido van Rossum
2000
Python 2.0 released with List Comprehensions, Garbage Collection, Unicode support
2008
Python 3.0 released with __future__ module, Advanced print function, Function annotations, Sensible comparison
2020
Python 2 discontinued

Implementations of Python

- Cpython (Default Python written in C)
- IronPython (Python running on .NET)
- Jython (Python running on the Java Virtual Machine)
- PyPy (A fast python implementation with a JIT compiler)
- Stackless Python (Branch of CPython supporting microthreads)
- MicroPython (Python running on micro controllers)
- PythonAnywhere (freemium hosted Python installation which lets you run Python in the browser)

Let's get started

Python Installation

- Windows: Download the setup file (for Python3) from the link and install it https://www.python.org/downloads/
- Linux: Use your package manager to install Python3
  - Ubuntu 20.04 and later ships with Python 3 pre-installed
  - For Arch Linux, use `$ sudo pacman -Sy python`
  - For Fedora, use `$ sudo yum install python`

Install VS Code IDE

- On Windows, Ubuntu and Fedora, you can download Visual Studio Code from this link https://code.visualstudio.com/Download
- On Arch Linux, install it from AUR or use Snap Store

Start with Basics

What are variables?

- Anything that one can name
- Technically, variable is a named memory location
- Example
  - x = 5
  - y = "Tenet"
- A variable can be a Number, a string, a List, a Tuple, a Set or an Object of Class

Input and Output

Use input(*prompt*) for input

- x = input() # no message will be displayed
- y = input("Enter number") # will display 'Enter number on terminal' and will wait for user input

Use print(*message*) for displaying output

- print("He Shot My Dog." + "\n" + "“You wanted me back...I’m back!”)
- print("2 + 2 = " + 4)

Conditional

if-else

```
if expression0:
    statement(s0)
else:
    statement(s1)
```

if-elif-else

```
if expression0:
    statement(s0)
elif expression1:
    statement(s1)
else:
    statement(s2)
```

Loops

While

```
while expression0:
    statement(s0)
```

For

```
for iterating_var0 in sequence0:
    statements(s0)
```

nested loop

```
for iterating_var0 in sequence0:
    for iterating_var1 in sequence1:
        statements(s1)
    statements(s0)
while expression0:
    while expression1:
        statement(s1)
    statement(s0)
```

Loop Controls

break

- Break any further execution

continue

- Skip current iteration

pass

- Use when you don't have anything to mention

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
  - *def function_name():*
    block_of_statements…
- Calling a function
  - *function_name()*
- Function with arguments
  - *def function_name(**arg0, arg2**):*
    process(arg1)
    process(arg2)
    return val

    function_name(some1, some2)
    function_name(some3, some4)
- Other features to discuss
  - Default parameter
  - Arbitrary Parameter
  - Passing
  - Recursion

Modules

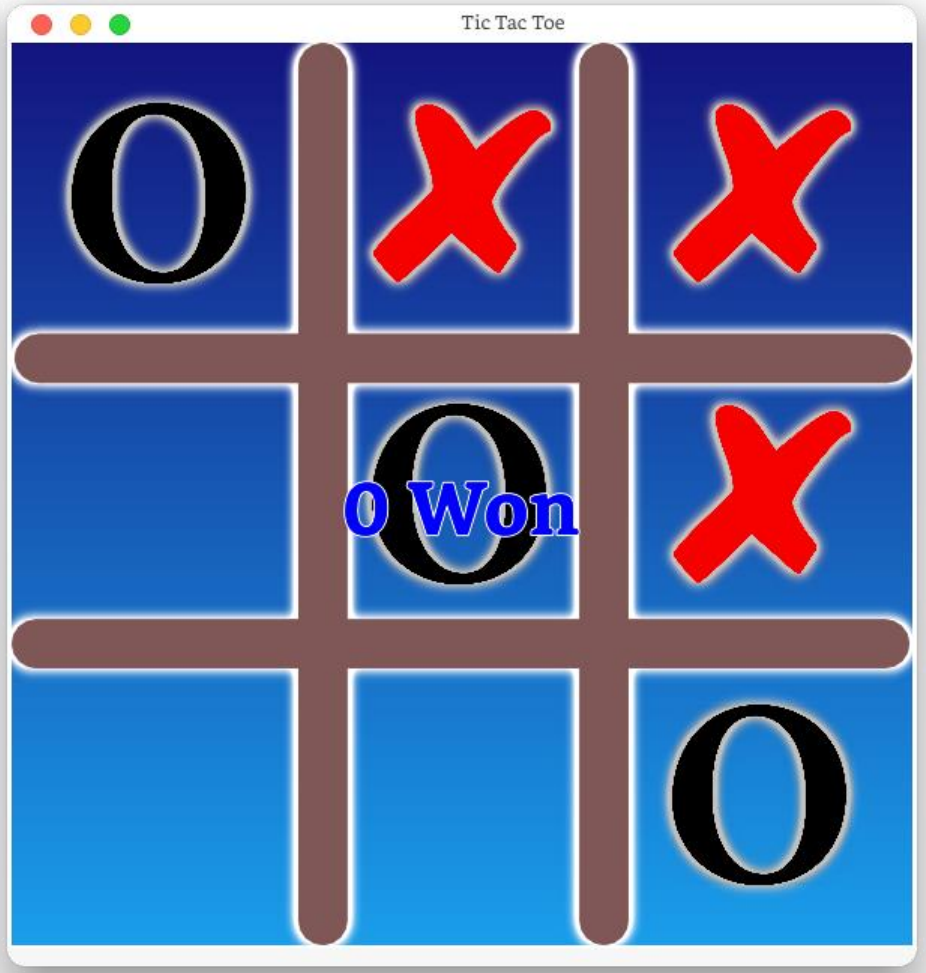
- A file containing a set of functions that you can include in other files to have a hierarchical structure
- A module can be Built-in or User-defined
- Some Built-in modules are
  - platform
  - cmath
  - copy
  - csv
  - datetime
  - os

Qt and PySide2

- Qt is a widget toolkit for creating graphical user interfaces as well as cross-platform applications that run on various software and hardware platforms such as Linux, Windows, macOS, Android or embedded systems with little or no change in the underlying codebase while still being a native application with native capabilities and speed.
- PySide2 is the official Python module from the Qt for Python project, which provides access to the complete Qt 5.12+ framework.
- `To install Pyside, simply use this command`
  - `pip install PySide2`

Goal

- A two player Tic Tac Toe game with logic written in Python and GUI designed in QML
- Link: https://github.com/vaishyavivek/TicTacToePySide



Lists

- More like Arrays
- Used to store multiple items in a single variable.
- Indexed
- Ordered
- Mutable
- Allows duplicate values
- Example
  - *laptops = ["Dell", "Lenovo", "HP", "Acer", "Asus"]*

Lists (contd.)

- List Items - Data Types: Supports all the types
- List Length: *len(list)*
- The list() Constructor: *list([1, 2, 3])*
- Access Items: *list[index]*
  - Positive, Negative and Range Indexing
- Change Item Value: *list[index] = newval*
  - Alter values in range
- Add Items
  - Insert new item at Index: *list.insert(index, item)*
  - Append to the end: *list.append(item)*
  - Merge two list: *list0.extend(list1)*
- Remove Items
  - Remove specific item: *list.remove(item)*
  - Remove at specified index: *list.pop(item)*
  - Empty the list: *list.clear()*
  - Delete the list completely: *del list*

Lists (contd.)

- Loop Through a List
  - *for x in list:*
    statement(x)
  - *for index in range(len(list)):*
    statement(list[index])
- Looping Using List Comprehension
  - *[statement(item) for item in list]*
- List Comprehension
  - *newlist = [expression for item in iterable if condition == True]*
- Sort the list
  - *list.sort()*
- Join two list
  - *list2 = list1 + list0*
- Reverse the order of list
  - list.reverse()

Sets

- Used to store multiple items in a single variable
- Unindexed
- Unordered
- Immutable
- No duplicate values allowed
- Example
  - *laptops = {"Dell", "Lenovo", "HP", "Acer", "Asus"}*

Sets (contd.)

- Set Items - Data Types: Supports all the types
- Set Length: *len(set)*
- The set() Constructor: set*((1, 2, 3))*
- Access Items
  - Only through loops
- Change Item Value
  - Not possible!
- Add Items
  - Add item: *set.add(item)*
  - Merge two sets or another list: *set0.extend(list1)*
- Remove Items
  - Remove specific item
    - *set.remove(item)* # will raise error if item doesn't exist
    - *Set.discard(item)* # will simply ignore if item doesn't exist
    - *Set.pop(item)* # remove random item
  - Empty the list: set.*clear()*
  - Delete the list completely: *del set*

Sets (contd.)

- Loop Through a Set
  - *for x in set:*
    statement(x)
- Join two sets
  - *set2 = set0.union(set1)*
  - *set0.update(set1)*
- Intersection of two sets
  - *set2 = set0.intersection(set1)*
  - *set0.intersection_update(set1)*
- Symmetric Difference of two sets
  - *set2 = set0.symmetric_difference(set1)*
  - *set0.symmetric_difference(set1)*

Tuples

- Used to store multiple items in a single variable
- Indexed
- Ordered
- Immutable
- Duplicates are allowed
- Example
  - *laptops = ("Dell", "Lenovo", "HP", "Acer", "Asus")*

Tuples (contd.)

- TupleItems - Data Types: Supports all the types
- TupleLength: *len(tuple)*
- The set() Constructor: *tuple((1, 2, 3))*
- Tuple with single item: tuple0 = (item0,)
- Access Items: *list[index]*
  - Positive, Negative and Range Indexing
- Change Item Value: *list[index] = newval*
  - Alter values in range
- Add Items
  - Insert new item at Index: *list.insert(index, item)*
  - Append to the end: *list.append(item)*
  - Merge two list: *list0.extend(list1)*
- Remove Items
  - Remove specific item: *list.remove(item)*
  - Remove at specified index: *list.pop(item)*
  - Empty the list: *list.clear()*
  - Delete the list completely: *del list*

Tuples (contd.)

- Loop Through a Set
  - *for x in set:*
    statement(x)
  - *for index in range(len(set)):*
    statement(list[index])
- Unpack tuple
  - *(val0, val1, val2) = tuple0*
- Join two tuple
  - tuple*2 = tuple1 + tuple0*

Dictionary

- Used to store data values in key:value pairs
- Ordered
- Mutable
- No duplicate values allowed
- Example
  - *thisdict = {*
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
    }

Dictionary

(contd.)

- Dictionary Items - Data Types: Supports all the types
- DictionaryLength: *len(dictionary)*
- Access Items
  - Through key: *dict0[key0] = item1*
  - Through get method: *dict0.get(key0)*
- Get all the keys: *dict0.keys()*
- Get all the values: *dict0.values()*
- Change items
  - Default way: *dict0[key0] = newval0*
  - Using Update: dict0.update({key0: val0})
- Add Items: same as change
- Remove Items
  - dict0.pop(key0)
  - Empty the list: dict0.*clear()*

Dictionary

(contd.)

- Loop Through a Dictionary
  - *for x in dict0:*
    statement(x) #perform operation on keys
  - *for x in dict0.keys():*
    statement(x) #perform operation on keys
  - *for x in dict0:*
    statement(dict0[x]) #perform operation on values
  - *for x in dict0.values():*
    statement(x) #perform operation on values
  - *for x, y in dict0.items():*
    statement(x, y) #perform operation on key:value
- Make copy of Dictionary
  - *newdict = olddict* # will make shallow copy
  - *newdict = olddict.copy()* # will make deep copy
- Nested Dictionary
  - myfamily = {
    "child1" : {
    "name" : "kushal",
    "year" : 2004
    },
    "address" : "mumbai"
    }

Files

- A **file** is an object on a computer that stores data, information, settings, or commands used with a computer program.
- Open a file: *f = open(filename, mode)*
  - `"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
  - `"t"` - Text - Default value. Text mode
  - `"b"` - Binary - Binary mode (e.g. images)
- Read the contents of file
  - *print(f.read())*
  - *print(f.readline())*
- Close after performing operation
  - *f.close()*
- Write or create new file
  - *f.write('some data')*
- Delete an existing file
  - *import os*
    if os.path.exists(filename):
    os.remove(filename)

Error 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:
  - The try block lets you test a block of code for errors.
  - The except block lets you handle the error.
  - The finally block lets you execute code, regardless of the result of the try- and except blocks.
- Example
  - ```
    try:
      print(x)
    except:
      print("Something went wrong")
    else:
      print("Nothing went wrong")
    finally:
      print("The 'try except' is finished")
    ```
    ```

Map

- The `map( )` function executes a specified function for each item in an iterable. The item is sent to the function as a parameter.
- *result = map(function, iterables)*
- *Example*
  - *def myfunc(a, b):*
    return a + b

    x = map(myfunc, ('apple', 'banana', 'cherry'), ('orange', 'lemon', 'pineapple'))

Filter

- The `filter()` function returns an iterator were the items are filtered through a function to test if the item is accepted or not.
- *result = filter(function, iterable)*
- Example
  - *ages = [5, 12, 17, 18, 24, 32]*

```
def myFunc(x):
  if x < 18:
    return False
  else:
    return True

adults = filter(myFunc, ages)

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

Zip

- The `zip()` function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc.
- If the passed iterators have different lengths, the iterator with the least items decides the length of the new iterator.
- *result = zip(iterator1, iterator2, iterator3 ...)*
- Example
  - *a = ("John", "Charles", "Mike")*
    b = ("Jenny", "Christy", "Monica")

    x = zip(a, b)

Some useful Python functions

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	divmod() - Returns the quotient and the remainder when argument1 is divided by argument2	eval() - Evaluates and executes an expression
exec() - Executes the specified code (or object)	format() - Formats a specified value	hex() - Converts a number into a hexadecimal value	max() - Returns the largest item in an iterable	min() - Returns the smallest item in an iterable
oct() - Converts a number into an octal	pow() - Returns the value of x to the power of y	range() - Returns a sequence of numbers, starting from 0 and increments by 1 (by default)	sum() - Sums the items of an iterator	type() - Returns the type of an object

Let's dig deeper

Object Oriented Programming

- OOP is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods.
- A Class is a blueprint of real world scenario.
- An object is an entity of the Class.

Abstraction

- Refers to, providing only essential information to the outside world and hiding their background details.
- For example, a web server hides how it processes data it receives, the end user just hits the endpoints and gets the data back.

Encapsulation

- The process of binding data members (variables, properties) and member functions (methods) into a single unit.
- It is also a way of restricting access to certain properties or component. The best example for encapsulation is a class.

Inheritance

- The ability to create a new class from an existing class
- We can create a Child class from a Parent class such that it inherits the properties and methods of the parent class and can have its own additional properties and methods.
- For example, if we have a class Vehicle that has properties like Color, Price, etc, we can create 2 classes like Bike and Car from it that have those 2 properties and additional properties that are specialized for them like a car has numberOfWindows while a bike cannot. Same is applicable to methods.

Polymorphism

- Take many forms.
- Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.
- Python polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

OOP with Python

- Python is an object oriented programming language.
- Almost everything in Python is an object, with its properties and methods.
- Create a Class
  - Create a class named MyClass, with a property named x:
    - *class MyClass:*
      x = 5
- Create an Object
  - Create an object named p1, and print the value of x:
    - *p1 = MyClass()*
      print(p1.x)

OOP in Python (contd.)

Constructor and class methods

```
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()
```

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

Delete Object and/or its properties

```
del p1.age
del p1
```

class definitions cannot be empty but sometimes its needed to leave the definition without any statements.

```
class Person:
  pass
```

Time to get hands dirty

Thank you

Workshop Content

- Applications of Python (15-20m)
- How it all started and present (5m)
- Python Setup (30m)
- Python Basics (2h)
  - Numbers, Variables, String, Input, Output, List, Dictionary, Tuple, Set
  - Loops, File, List Comprehension
- Intermediate (2h)
  - Functions, Lambda, Map, Filter, Error Handling, OOP
- Advanced (45m)
  - GUI with QML
- Upcoming (10m)