Python Quickstart - PyQuick

Vihar Kurama

Co-founder, Caravel.AI

What

Python is a programming language that lets you work quickly and integrate systems more effectively.*

*Source: https://www.python.org

Who and When?

- Guido van Rossum
- Feb, 1991
- Python Software Foundation
- https://www.python.org

Why

Python is powerful… and fast;
plays well with others;
runs everywhere;
is friendly & easy to learn;
is Open.*

These are some of the reasons people who use Python would rather not use anything else.*

*Source: https://www.python.org

Who use it.

Google
NASA
OpenAI
Dropbox
Quora
YouTube
mozilla
UBER

Vihar Kurama, 2018

Applications

- Web and Internet Development
- Scientific and Numeric Computing
- Education
- Desktop GUIs
- Software Development
- Business Applications
- Machine Learning Systems and Algorithms
- and many more…

Vihar Kurama, 2018

This Lecture

Exciting Parts of Programming with Python

Programming Intuitions with Python

Rapid Prototyping with Python

Building Practical Powerful Software with Python

Suggested Readings

Python Quick, Vamsi Kurama

Python Programming: A Modern Approach, Vamsi Kurama

Learn Python the Hard Way, Zed Shaw

A Byte of Python, Ch Swaroop

Think Python, Allen B. Downey

Dive into Python, Mark Pilgrim

Python Programming Language

Python is a **general purpose**, **dynamically typed** and **interpreted** programming language.

Vihar Kurama, 2018

Python Versions

- 2.x.y
- 3.x.y
- This Lecture is based on 3.6.y

Running Python Programs

Two ways of Running Python Programs

- Running Python Interpreter

```
python

>>>
```

- Running Python Scripts

```
python hello.py
```

Hello World!

```
print("Hello world!")

print("Hello Python!")
```

Primer

- Storing Information
- Making Decisions
- Repeating Techniques
- Making Lists / Organising Data
- Building Instructions
- Avoiding Pit holes

Working with Data

Numbers

Text (Characters and Symbols)

Logic

Working with Data

Numbers

- int
- float

Text

- str

Logic

- bool

Python as a Calculator

```
>>> 5 + 3

>>> 5 + 83 * 3

>>> 89 + (4 * 2)

>>> 6 + 7 //(8**7)

>>> (6 * 7) / (9+10)*(8**7)
```

Vihar Kurama, 2018

Variables and Assignment

```
>>> a = 24

>>> b = 19

>>> a + b

43

>>> b = a

>>> greet = "Hello"

>>> who = " World!"

>>> greet + who

Hello World!
```

Vihar Kurama, 2018

Interpreted Type?

Use `type(variable)` Function.

```
>>> a = 9.0

>>> type(a)

<class 'float'>

>>> b = 9

>>> type(b)

<class 'int'>
```

Vihar Kurama, 2018

Math

Operators

+, - ,*, **, /, %, << , >>, &, l, ^,
< >, <=, >=, ==, !=

** The Beautiful Math Library is also your Treasure

Boolean logic

```
>>> a = True
>>> b = False
>>> type(a)
<class 'bool'>
```

Vihar Kurama, 2018

Boolean logic Expressions

```
>>> print(24 > 17)
>>> print(19 < 2)
>>> print(24 > 17 and 19 < 2)
>>> print(24 > 17 or 19 < 2)
```

Vihar Kurama, 2018

Primitive Types

- `int`
- `float`
- `str`
- `bool`

Input

How do you deal with an input from the user?

```
>>> a = 17
>>> name = "Python"
>>> a = input()
2
>>> print(a)
2
```

Strings

```
>>> x = "hello"
>>> y = 'world'
>>> x = """This is a multi-line string
written in
three lines."""
>>> y = '''multi-line strings can be written
using three single quote characters as well.
The string can contain 'single quotes' or
"double quotes"
in side it.'''
```

String Interpolation

```
name = "Bond"
print(name)
print("Hello, I am {}".format(name))

num = "007"
print("Hello, I am {} {}".format(name, num))
```

Input with Prompt

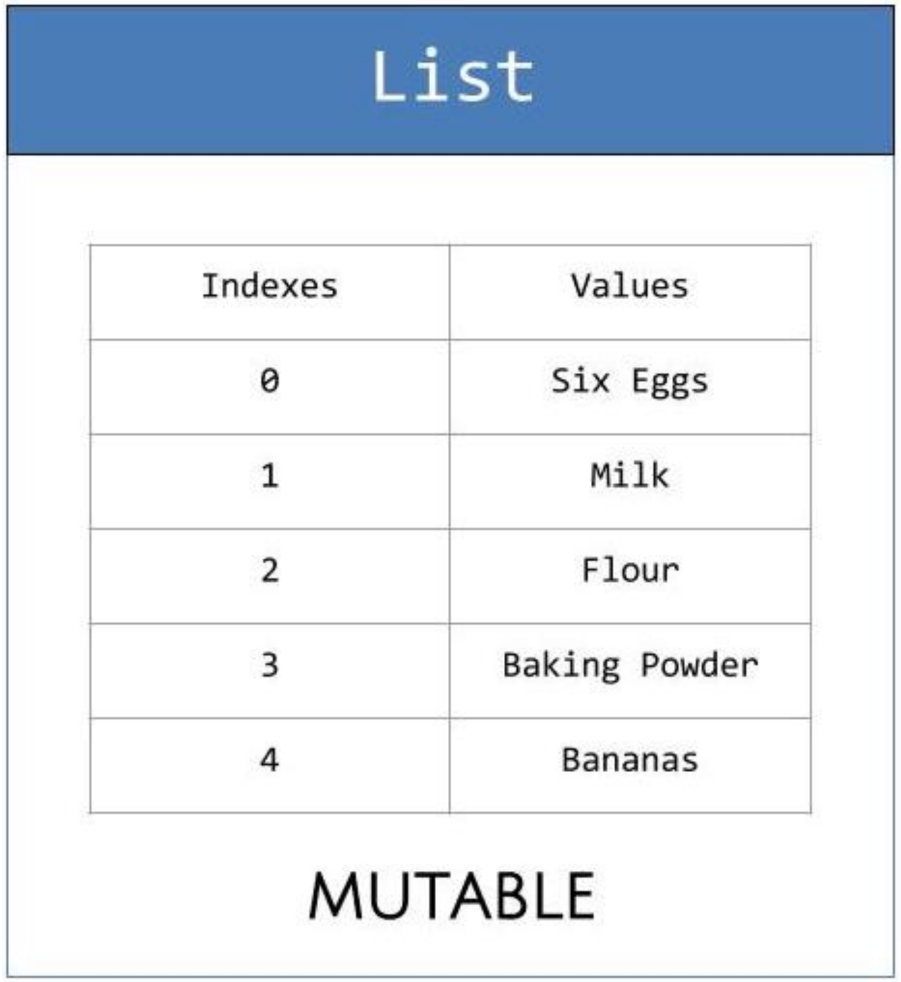
How do you deal with an input with a prompt?

```
>>> name = input("Enter your name: ")

Enter your name: Rossum

>>> print(name)

Rossum
```

String Methods

```
.strip
.spilt
.upper
.title
.capitalize
.startswith
.swapcase
.islower
```

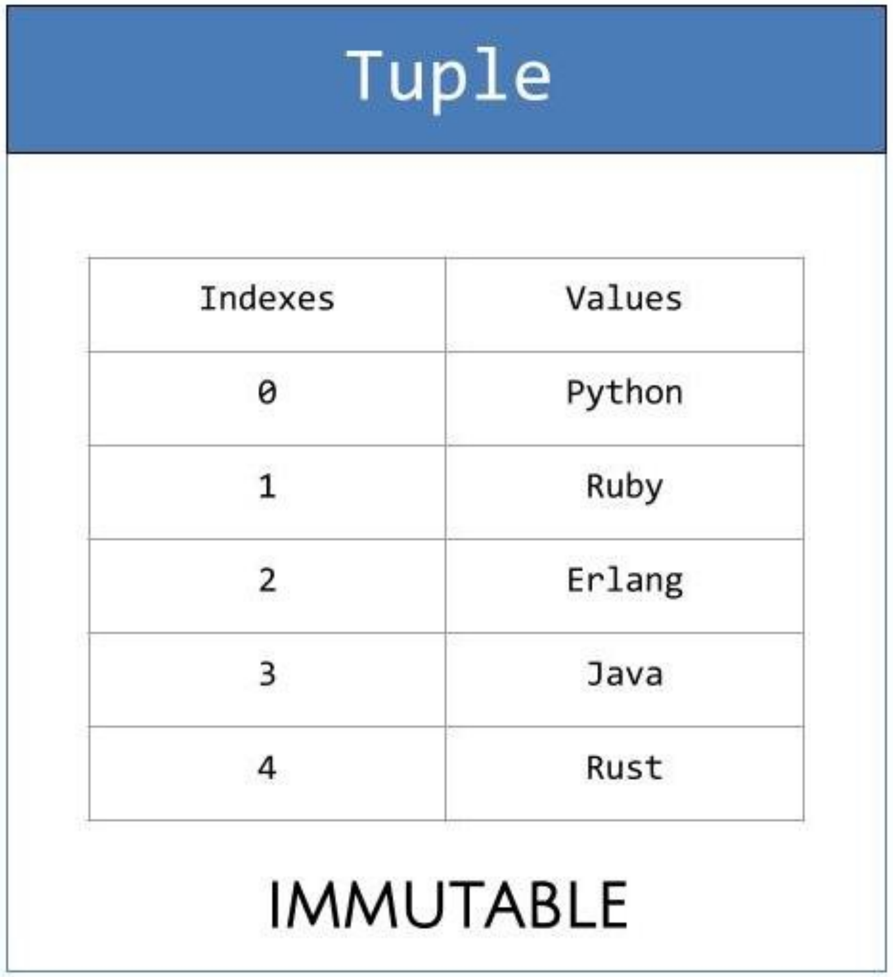
Compound Types



List

Indexes	Values
0	Six Eggs
1	Milk
2	Flour
3	Baking Powder
4	Bananas

MUTABLE



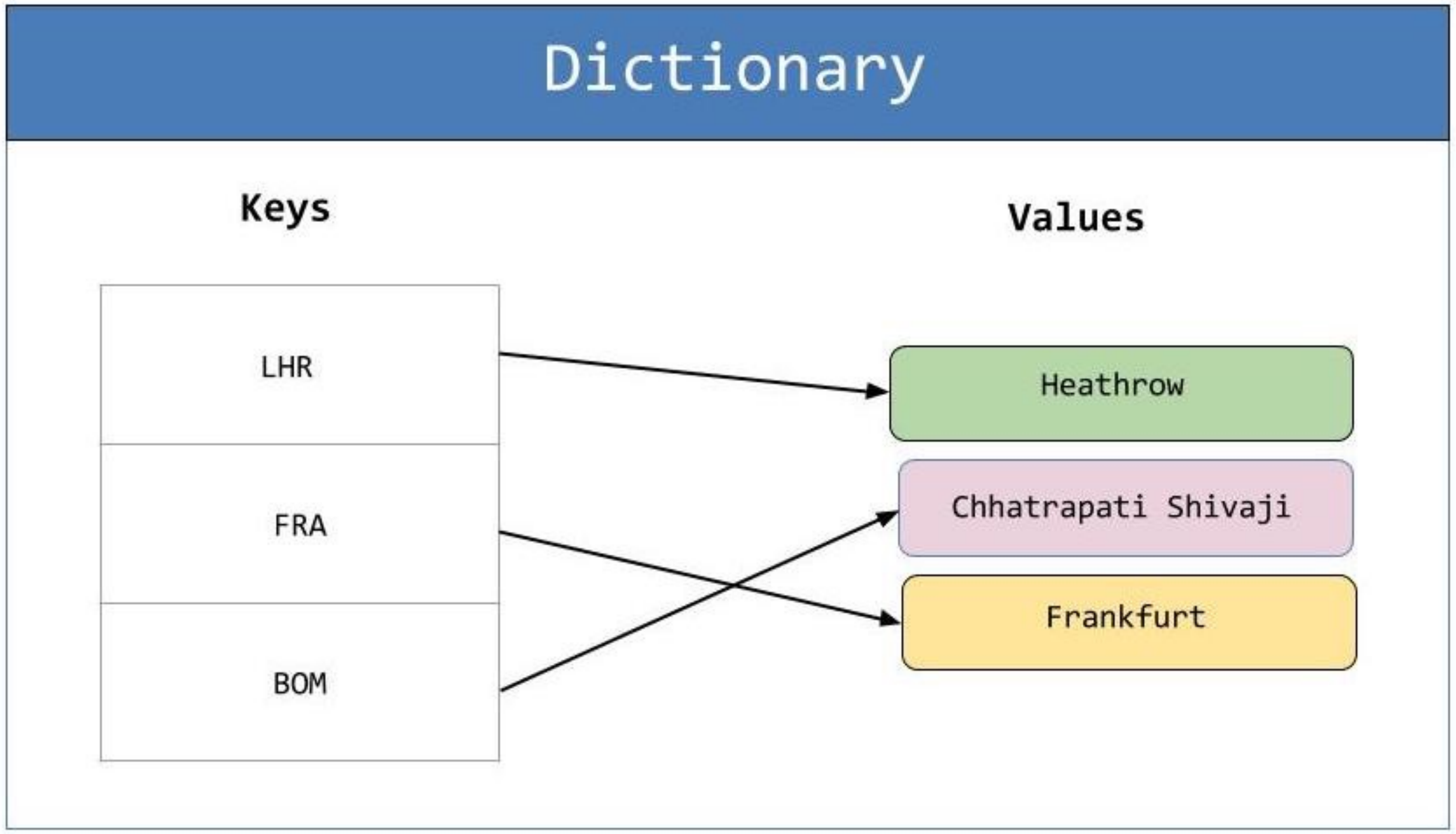
Tuple

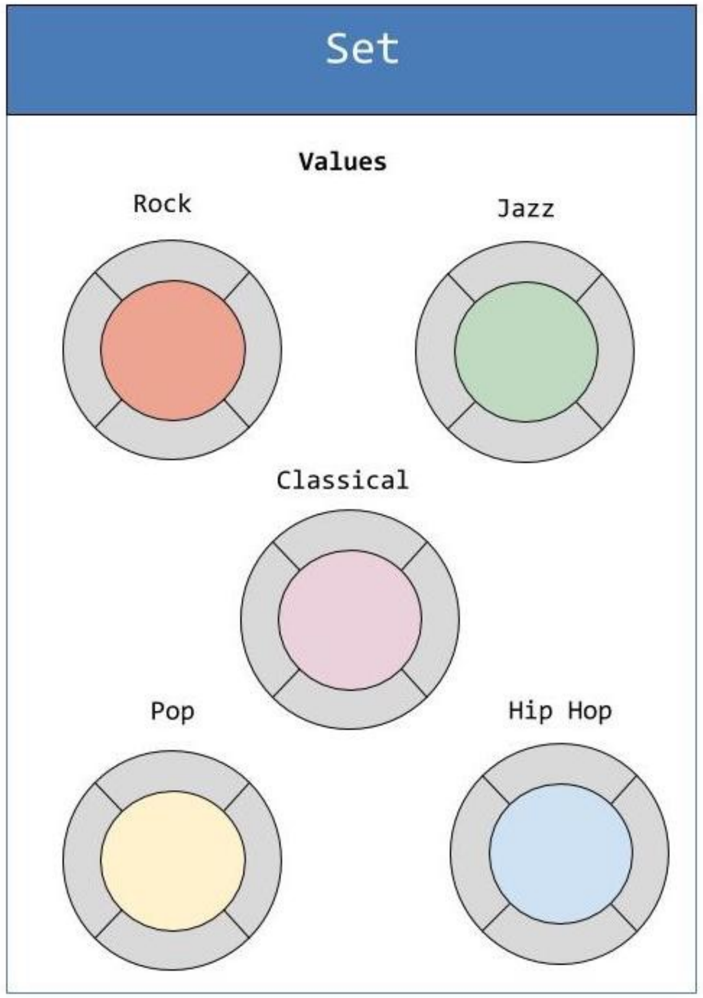
Indexes	Values
0	Python
1	Ruby
2	Erlang
3	Java
4	Rust

IMMUTABLE

Vihar Kurama, 2018

Compound Types





Vihar Kurama, 2018

Lists

- Enclosed with `[]`

- **Mutable**.

- **Heterogenous**.

```
>>> cart = ["eggs", True, 0, 24, [9]]

>>> type(cart)
<class 'list'>
```

List Methods

```
.sort

.append

.reverse

.insert

.index
```

Built-in Methods for Sequences

len(sequence)

min(sequence)

max(sequence)

sum(sequence)

Built-in Methods for Sequences

```
dir(variable)

help(variable)

type(variable)
```

Methods

- **Everything** in Python is a **object**.
- Methods are **special kind of functions** that work on an object.

Vihar Kurama, 2018

Methods

```
object.method(params)
```

Tuples

- Enclosed with `()`
- Immutable
- Heterogenous

```
>>> langs = ("py", "java", "cpp", "c")
>>> type(langs)
<class 'tuple'>
```

Vihar Kurama, 2018

Tuples Methods

```
>>> # Tuples have no methods.
```

Dictionaries

- Enclosed with **`{}`**
- Items exists as **Key-value** pairs.
- Access value by **key** of the item.

```
>>> airports = {"", True, 0, 24, [9]}

>>> type(airports)

<class 'list'>
```

Dictionaries Methods

```
.keys
.values
.items
```

Sets

- Enclosed with `{}`

- All items are unique.

- Use `set()` for typecasting.

```
>>> primes = {2, 3, 3 5, 7, 7, 11, 13, 17}

>>> type(primes)

<class 'set'>
```

Sets Methods

```
.union
.intersection
.difference
.symmetric_difference
```

White Space

- **White space is extremely important in Python.**
- **Even a single space matters to the Python Interpreter.**
- **Don't mix tabs and space.**

Indentation

```
x, y = 0, 2
if x==0:
....print("Say Hello")
....print("World")
if y == 2:
....print("Ok I Need to do something")
else:
....print("Say Bye")
```

Indentation

```
def hello(x, y):
....if x==0:
....print("Say Hello")
........print("World")
....if y == 2:
........print("Ok I Need to do something")
....else:
........print("Say Bye")

hello(0, 7)
```

Vihar Kurama, 2018

White Space

```
if (x==0)
{
printf("Hello");
printf("World");
}
else
{
printf("Say Bye")
}
```

```
if x==0:
....print("Say Hello")
....print("World")
else:
....print("Say Bye")
```

Vihar Kurama, 2018

Control flow

- Key words `if-elif-else`.
- Colon: after every condition.
- Indent statements by four(4) spaces.

Conditional Flow: Example

```
temperature = 43

if temperature <= 30:
    print("It's very cold. Consider wearing a scarf.")
```

Vihar Kurama, 2018

Conditional Flow: Example

```
temperature = 43

if temperature <= 30:
    print("It's very cold. Consider wearing a scarf.")
else:
    print("It's not that cold. Wear a t-shirt")
```

Conditional Flow: Example

```
temperature = 43

if temperature <= 30:
    print("It's very cold. Consider wearing a scarf.")
elif temperature >=40:
    print("It's really warm. Don't forget to wear a sunscreen")
else:
    print("It's not that cold. Wear a t-shirt")
```

Vihar Kurama, 2018

Repetitive Flow

Two kinds of Looping techniques

- `for`
- `while`

for: Example

```
for num in [2, 17, 19, 24]:
    print(num)


for con in ("ind", "aus", "eng", "srl"):
    print(con)


for key in {"python": "py", "ruby":"rb", "erlang": "erl"}:
    print(key)


for key, value in {"python": "py", "ruby":"rb", "erlang": "erl"}.items():
    print(key, value)


for element in {2,3,5,7,11, 13, 17}:
    print(element)
```

for range: Example

```
for i in range(0,100):
    print("Python!!")


for i in range(0, 24):
    print("ISB000{}".format(i))
```

while: Example

```
pool = 0
while pool < 100:
    pool += 10
    print("{} litres".format(pool))
```

Loop forever

```
while True:
    print("Hello!")
    print("IIDT")
```

```
while True:
    print("Receiving…")
```

Vihar Kurama, 2018

Making it clear!

for loop, iterates over sequences

while loop runs until the condition is False

Loop Jumps

`break`

The break statement exits a **for** or **while** loop completely.

`continue`

A continue statement is used to **end the current loop** iteration and **return control to the loop** statement.

break: Example

```
puzzle_input = "great minds think alike"
puzzle_output = ""
vowels = ['a', 'e', 'i', 'o', 'u']
for character in puzzle_input:
    if character in vowels:
        continue
    else:
        puzzle_output.append(character)
print(puzzle_output)
```

continue: Example

```
puzzle_input = "great minds think alike"
puzzle_output = ""
vowels = ['a', 'e', 'i', 'o', 'u']
for character in puzzle_input:
    if character in vowels:
        continue
    else:
        puzzle_output.append(character)
print(puzzle_output)
```

Vihar Kurama, 2018

Placeholder - Do nothing;

```
if condition:
    pass

while condition:
    pass


def create_alarm:
    pass

class Bank:
    pass
```

Looping: In Summary

Making it clear!

- **for** - iterates over sequence.
- **while** - until the condition is false.

List Comprehensions

```
c = [39.2, 36.5, 37.3, 37.8]

f = [((float(9)/5)*t + 32) for t in c]


# [102.56, 97.7, 99.14,100.03999999999999]
```

Vihar Kurama, 2018

List Comprehensions

```
colors = ["red", "green", "yellow","blue"]

things = [ "house", "car", "tree" ]

ct = [(x,y) for x in colors for y in things]

print(ct)
```

Vihar Kurama, 2018

Functions

Two things

1. Define a Function.

2. Call a Function.

Functions

```
def function_name(params):

    # statement_1

    # statement_2

    # statement_3


function_name(params)
```

Functions in Python

```python
# Defining Function greet.
def greet():
    print("Hello World!")


# Calling the Function greet.
greet()
```

Functions in Python

```
def greet(name):
    print("Hello {}!".format(name))


greet("stark")
```

Functions in Python

```python
def greet(name, gender=''):
    if gender == 'm':
        print("Hello Mr. {}".format(name))
    elif gender == 'f':
        print("Hello Ms. {}".format(name))
    else:
        print("Hello {}".format(name))

greet('stark', 'm')
greet('potts', 'f')
greet('parry')
```

Lambda

```
f1 = lambda x: x*x

f2 = lambda a, b: a**2 + b**2 + 2*a*b

f3 = lambda a, b: a if (a > b) else b
```

Primer

- Storing Information ✓
- Making Decisions ✓
- Repeating Techniques ✓
- Making Lists / Organising Data ✓
- Building Instructions ✓
- Avoiding Pit holes ✓

Vihar Kurama, 2018

Modules: num.py

```
def square(x):
    return x * x


def cube(x):
    return x * x * x
```

Vihar Kurama, 2018

Module

- `import num`
- `from num import square`
- `from num import *`
- `from num import cube as c`

Module

- `import module`
- `from module import something`
- `from module import *`
- `from module import something as name`

Object Oriented Programming

Object Orientation offers Abstraction.

Three Principles of Object Oriented Programming

- **Encapsulation**
- **Inheritance**
- **Polymorphism**

Vihar Kurama, 2018

OOP

- **Object**

A real world entity which has state and behaviour.

- **Class**

A blue print of an object.

OOP with Python

Everything is an object in Python.

```
class Person:
    pass

jack = Person()
```

Vihar Kurama, 2018

Classes

```
class Box:

    def method_1(param):

        pass



b1 = Box()

b1.method()
```

Vihar Kurama, 2018

Methods

```
class Box:
    def method_1(param):
        pass
    pass

b1 = Box()
b1.method()
```

__init__ Method

```
class Box():
    def __init__(a, b):
        pass
    pass



b1 = Box(a, b)
```

Modules continued.

```
from warehouse import Box

b1 = Box()

from warehouse import Box as B

b1 = B()

import warehouse

b1 = warehouse.Box()
```

Errors and Exceptions

```
try:
    # statements
except:
    # statements
finally:
    # statements
else:
    # statements
```

Vihar Kurama, 2018

exec and eval

```
>>> exec("a = 2")
>>> eval("a + 19")
21
>>> loop = """
d = [2, 17, 19, 24]
for nums in d:
    print(d)
"""
>>> exec(loop)
```

Standard Library

- `math, decimal, time, datetime, re`
- `glob, os, shutil, tempfile`
- `random`
- `sqlite, json, pickle`
- `urllib, wsgiref, logging`
- `itertools, functools`

Thanks!