Python Introduction

Amirhossein Abaskohi

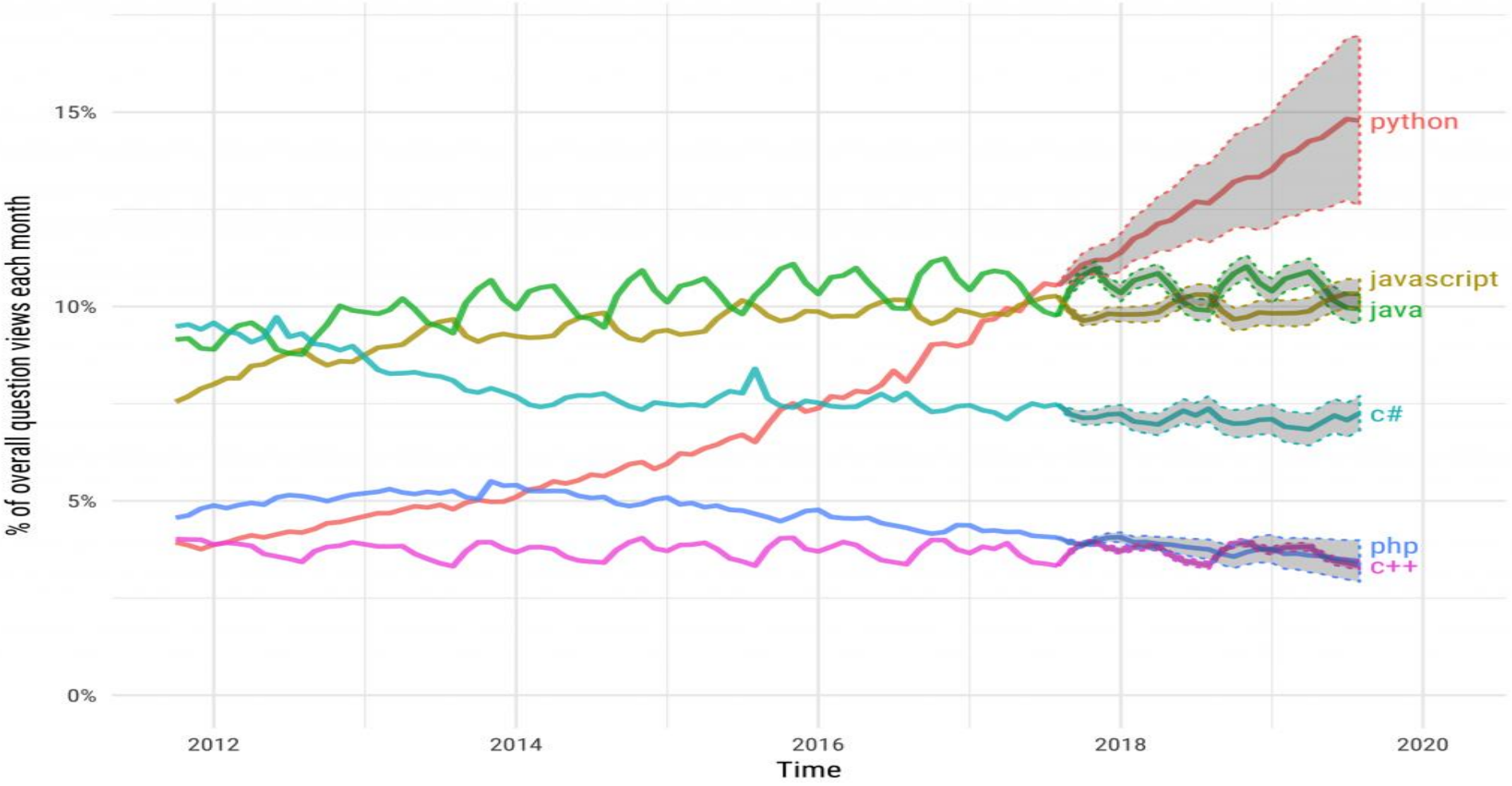
University of Tehran ACM Summer School 2021

Why python?

- Most recent popular (scripting/extension) language
  - Although origin ~ 1991
- Heritage
  - TCL: shell
  - Perl: string (regex) processing
- Object-oriented
  - Rather than add-on(OOTCL)
- Good features
  - Coherence : Easy to read, write and maintain
  - Power
  - Scope: Rapid development + Large systems
  - Powerful Libraries

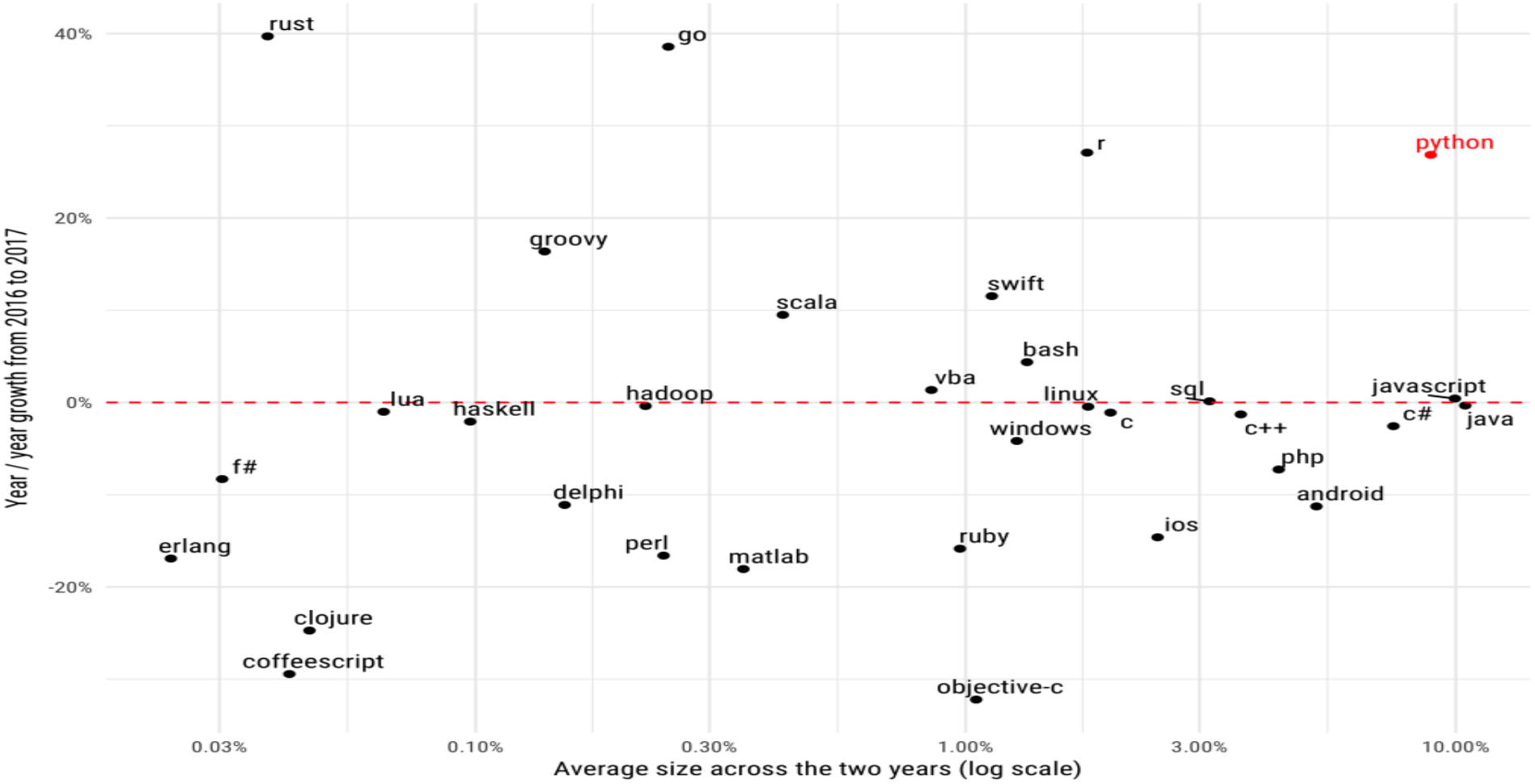
Projections of future traffic for major programming languages

Future traffic is predicted with an STL model, along with an 80% prediction interval.



Year over year growth in traffic to programming languages/platforms

Comparing question views in January-August of 2016 and 2017, in World Bank high-income countries.
TypeScript had a growth rate of 142% and an average size of .36%; and was omitted.

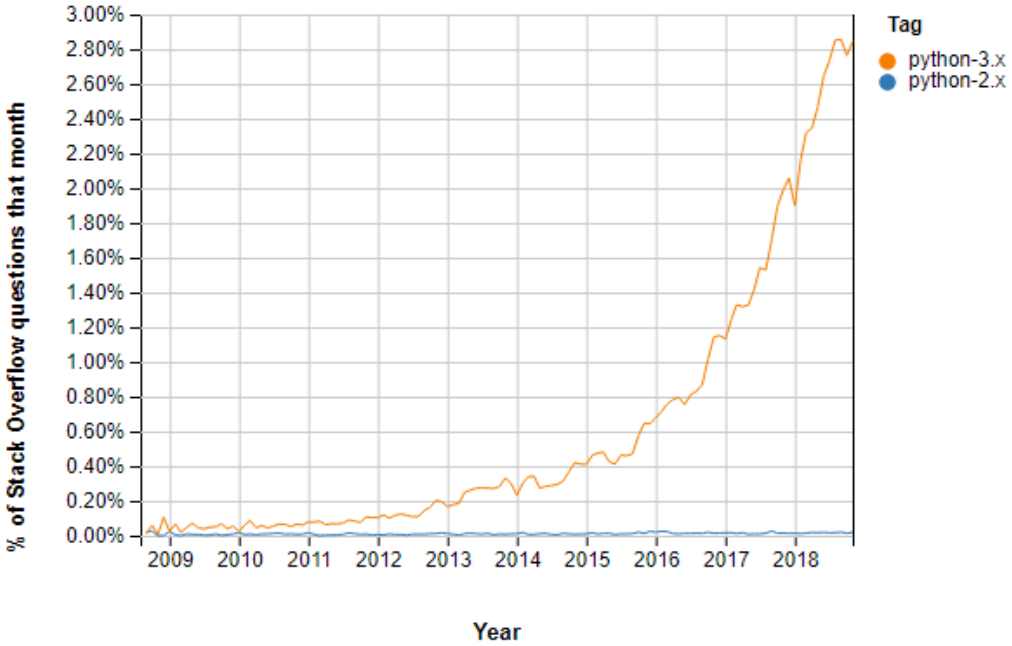


Python Usage

- Data analytics and machine learning
- Shell tools
  - System admin tools, command line programs
- Extension-language work
- Rapid prototyping and development
- Language-based modules
  - Instead of special-purpose parsers
- Graphical user interfaces
- Database access
- Distributed Programming
- Internet scripting

Python 2 OR Python 3?

- Python 3 is supported by a large Python developer's community. Getting support is easy.
- Python 3 supports modern techniques like AI, machine learning, and data science



Importing and Modules

- Use classes and functions defined in another file
- A Python module is a file with the same name
- Use Import to use modules
- Examples:

```
import somefile

from somefile import *

from somefile import className

from somefile import className as CN
```

List, Dictionary, Tuple and Set

List	Tuple	Set	Dictionary
Stores data in single row and multiple columns	Stores in single row and multiple rows	Stores data in a single row	Stores data in key-value format
Can be represented by []	Can be represented by ()	Can be represented by {}	Can be represented by {}
Allows duplicate elements	Allows duplicate elements	Doesn’t allow duplicate elements	Values can be duplicated but not keys
Mutable	Immutable	Mutable	Mutable

Time complexity is important in where to use : See this

Functions

```
def my_function(fname):
  print(fname + “Refers”)
  return 2,True

a,b = my_function(“ACM”)
```

*args and **kwargs

*args (Non keyword Arguments)

- we are not sure about the number of arguments that can be passed to a function

```
def adder(*num):
    sum = 0

    for n in num:
        sum = sum + n

    print("Sum:",sum)

adder(3,5)
adder(4,5,6,7)
adder(1,2,3,5,6)
```

**kwargs(Keyword Arguments)

- To pass keyword arguments

```
def intro(**data):
    print("\nData type of argument:",type(data))

    for key, value in data.items():
        print("{} is {}".format(key,value))

intro(Firstname="Sita", Lastname="Sharma", Age=22, Phone=1234567890)
```

Defining a Class

- A class is a special data type which defines how to build a certain kind of object
- The class also stores some data items that are shared by all the instances of this class
- Instances are objects that are created which follow the definition given inside of the class
- Python doesn't use separate class interface definitions as in some language
- You just define the class and then use it
- BUT ENCAPSULATION IN PYTHON IS AWFUL

Methods in Classes

- Define a method in a class by including function definitions within the scope of the class block
- There must be a special first argument `self` in all of the methods definitions which gets bound to the calling instance
- There is usually a special method called __init__ which is the class constructor

A simple class example

```
class student:
"""A class representing a student """
def __init__(self,n,a):
      self.full_name = n
      self.age = a
def get_age(self):
      return self.age
```

Creating and deleting instances

- There is no “new” keyword here
- Just use the class name with () notation and assign the result to a variable
- The arguments passed to the class name are given to its __init__ method
- When you are done with an object, you don’t have to delete or free it explicitly
- Python has automatic garbage collector
- Generally works well, few memory leaks
- There is no “destructor” method

Data vs. Class Attributes

```
class counter:
  overall_total = 0
       # class attribute
  def __init__(self):
      self.my_total = 0
        # data attribute
  def increment(self):
     counter.overall_total = \
     counter.overall_total + 1
     self.my_total = \
     self.my_total + 1
```

```
>>> a = counter()
>>> b = counter()
>>> a.increment()
>>> b.increment()
>>> b.increment()
>>> a.my_total
1
>>> a.__class__.overall_total
3
>>> b.my_total
2
>>> b.__class__.overall_total
3
```

Inheritance

- A class can extend the definition of another class
- To define a subclass, put the name of the superclass in parentheses after the subclass's name on the first line of definition:

```
Class Cs_student(student):
```

- Python has not extends keyword like Jave
- Multiple inheritance is supported

Definition of a class extending students

```
Class Student:
  "A class representing a student."

  def __init__(self,n,a):
      self.full_name = n
      self.age = a

  def get_age(self):
      return self.age

Class Cs_student (student):
  "A class extending student."

  def __init__(self,n,a,s):
      student.__init__(self,n,a) #Call __init__ for student
      self.section_num = s

  def get_age():     #Redefines get_age method entirely
      print "Age: " + str(self.age)
```

Special Methods - __repr__

```
class student:
   ...
    def __repr__(self):
      return "I'm named " + self.full_name
   ...

>>> f = student("Bob Smith", 23)
>>> print f
I'm named Bob Smith
>>> f
"I'm named Bob Smith"
```

Other special methods

You can define as well:

- __init__
- __cmp__ : Define how == works
- __len__ : Define how len(obj) works
- __copy__ : Define how to copy a class