Python 3

Suriyadeepan Ramamoorthy

Introduction

- widely used **high-level**, **interpreted**, **dynamic** programming language
- emphasizes code **readability**
- syntax allows programmers to **express** concepts in fewer lines of code

Guido von Rossum

Pythonic Thinking

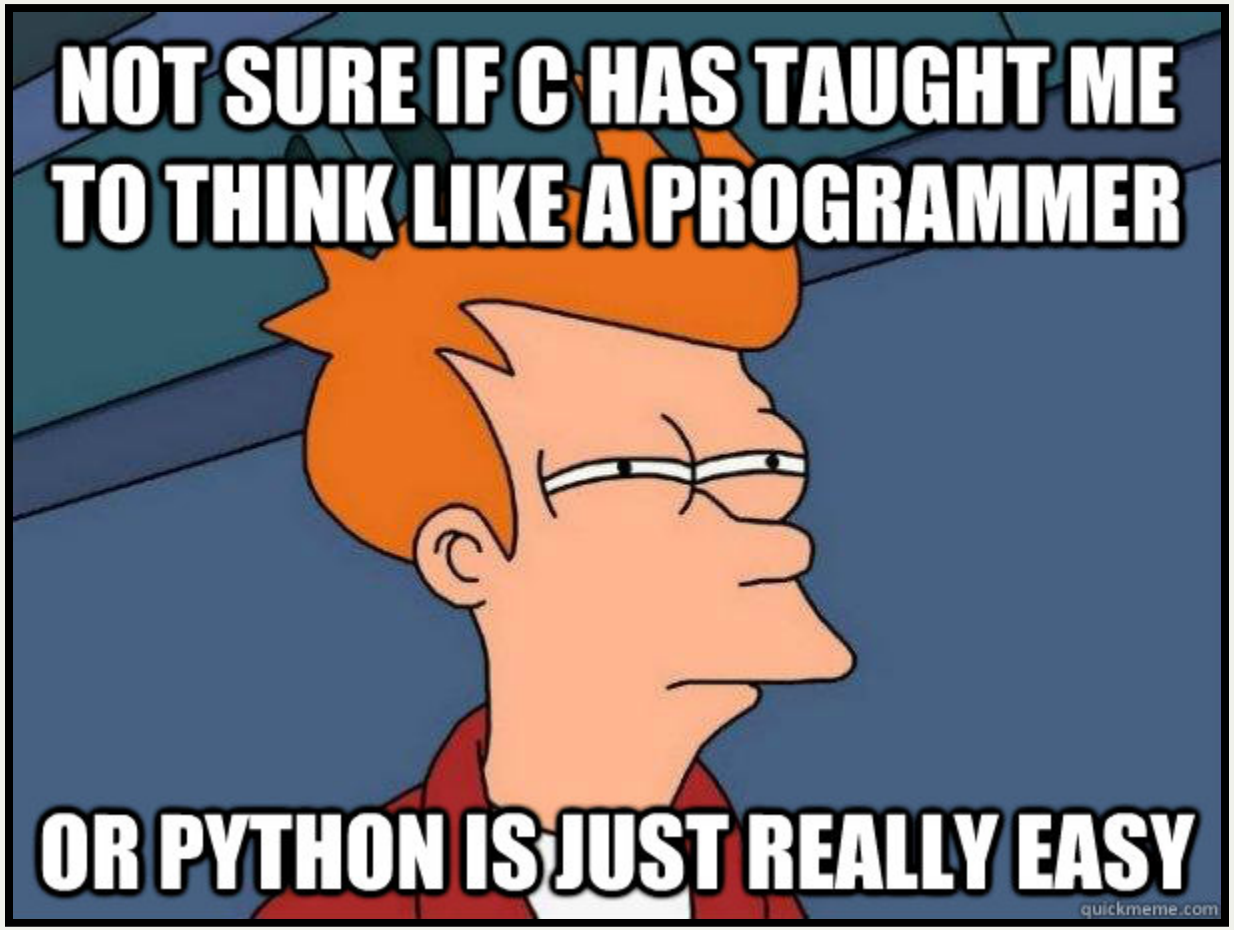
- be explicit
- simple over complex
- maximize readability

PEP 8 style guide

How do we proceed?

1. Introduce a python concept
2. Describe it
3. Demonstration
4. Practice the code snippets in exercises/
5. Solve problems
6. My solution

Python is EASY



Hello World

```
print('hello world')
print('hello {}'.format('world'))
print('hello {0}, {1}, {2} and {3}'.format('Bruce',
                    'Clark','Martha','Martha'))
```

Print the name of your batch mates; one name per line.

```
print('hello world')
print('hello {}'.format('world'))
print('hello {0}, {1}, {2} and {3}'.format('Bruce','Clark','Martha','Mar
```

Values

- **int** : 7
- **float** : 12.46
- **string** : "try this" = 'try this'
- **bool** : True/False
- **comment** : # comments are for people who read your code to understand
- **multiline comment**

''' Be more expressive.
Descibe in multiple
lines.
'''

Arithmetics I

```
x = 2
y = 3
print('Sum :  {}'.format(x+y))
z = x - y
print('Diff : {}'.format(z))
print('Prod : {}'.format(x*y))
print('Quo  : {}'.format(x/y))
```

```
print('{0}\n{1}\n{2}'.format('John Doe','John Smith', 'Jane Doe'))
```

Types

1. **int** : Integer
2. **float** : Decimal
3. **bool** : Boolean [True/False]
4. **str** : String
5. **list** : List of any other datatypes [HETEROGENOUS]
6. **tuple** : Tuple [Non-mutable List]
7. **dict** : Dictionary [Key-Value pairs]
8. **function** : Function

Strings

- Sequence of characters (not really)
- A character is also a string
- Appreciate the difference between '152' and 152

String Operations I

- len(string)
- string.find(substring)
- lower/upper(string)
- string.split(' ')

Logical Operations

1. less than : **<**
2. greater than : **>**
3. greater than or equal to : **>=**
4. less than or equal to : **<=**
5. **and** : logical and
6. **or** : logical or
7. **not** : logical not

Bitwise Operations

Conditional Statements

- IF...ELIF...ELSE
- Never forget the INDENTATION
- Nested IF

WHILE

1. Initialize the variable
2. Check condition
3. Increment (change) the variable in a meaningful way
4. Avoid infinite loops

FOR

1. Iterate/Go through items in a List
2. For each item in a list, do an operation once
3. Stop loop when the list ends
4. Understand **range( )** function
5. **item** is a variable that holds the value of current item in list
6. Name of **item** is arbitrary

LIST

1. **Heterogenous** collection of items
2. type : list
3. Each item has a type. Each item can be of any type.
4. A list can be an item in another list
5. Get item by index : list[**index**] = item
6. **index** ranges from 0 to length of the list
7. **len**(list) : lenght of the list
8. NEVER USE 'list' AS YOUR VARIABLE'S NAME

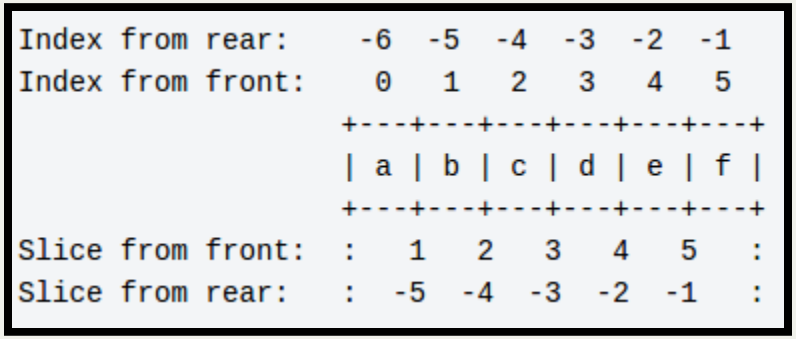
List Operations

1. len(list)
2. list.append(item)
3. list.extend(list2)
4. list1 + list2 : operator overloading
5. list.index(item)
6. max(list) / min(list)

Slicing

1. **a[start:end]** : items start through end-1
2. **a[start:]** : items start through the rest of the array
3. **a[:end]** : items from the beginning through end-1
4. **a[:]** : a copy of the whole array
5. **a[-1]** : last item in the array
6. **a[-2:]** : last two items in the array
7. **a[:-2]** : everything except the last two items
8. **a[low:high:stride]** : **stride** is the amount by which the index increases

Slicing


Index from rear: -6 -5 -4 -3 -2 -1
Index from front: 0 1 2 3 4 5
| a | b | c | d | e | f |
Slice from front: : 1 2 3 4 5 :
Slice from rear: : -5 -4 -3 -2 -1 :

Functions

1. Notice the INDENTATION
2. Take multiple arguments
3. Return multiple values
4. Default arguments
5. Nested functions
6. Anonymous function : **lambda**

Sorting

1. Appreciate the difference between **list.sort()** and **sorted(list)**
2. **reverse = True**
3. Custom key sorting : **key=len**

List Comprehension

1. **Map** : [**operation_on**(item) for **item** in **a_list**]
2. Iterate through the list
3. Take each item and convert it into something else
4. Map list to another list
5. **Filter** : [**operation_on**(item) for **item** in **a_list** if **condition_on(item)**]

import

1. Import external module : **import math**
2. Use : math.ceil( ), math.floor( )
3. Import a function from module : **from math import floor, sqrt**
4. Use : ceil( ), floor( )
5. Import external module as : **import math as M**
6. Use : M.ceil( ), M.floor( )
7. Import function from module, as : **from math import floor as f**
8. Use : f( )

File Operations

1. Write mode : 'w' (overwrites)
2. Append mode : 'a'
3. Read mode : 'r'
4. Write/Append creates file if necessary
5. Open('filename') returns a **file handle**
6. Use this handle to write to/read from file
7. file_handle.close()
8. Shorthand : **with...as**
9. **f.read()** vs **f.readline()**

Dictionary

1. DO NOT USE **dict** AS A VARIABLE NAME
2. Initialize an empty dictionary : d1 = { }
3. Add values d1[key] = value
4. Key can be anything **except a List**
5. Value can be anything.
6. d1.keys() : a list
7. d1.values() : a list
8. Iterating through items : **for key, value in d1.items():**

Tuples

1. Immutable lists
2. Notice the use of paranthesis **(...)**
3. Fun fact : (1) != (1,)
4. Slicing, Indexing works the same way as list
5. tuple(alist) : return a tuple of the list

OOP

1. Objects are encapsulation of variables (and other objects) and functions
2. Consider the class **Rectangle**
3. Attributes : **length**, **breadth**
4. Methods : **get_area**, **get_perimeter**
5. **self** argument : instance of class itself (**this** in java)
6. **init** : constructor
7. Creating an instance : **my_rect = Rectangle(w,h)**
8. Calling a method : **area = my_rect.get_area()**