
Programming in Python



Sarath Babu

SESSION-I



INDIAN INSTITUTE OF SPACE SCIENCE AND TECHNOLOGY
THIRUVANANTHAPURAM, KERALA, INDIA 695547

7th August, 2019

IEEE STUDENT BRANCH IIST



Workshop plan

■ Session-I

- Introduction to Python programming language
- Basic data structures

■ Session-II

- Control structures
- Functions
- Exception handling
- File handling
- Object Oriented Programming in Python

■ Session-III

- Introduction to **NumPy**
- Plotting in Python using **matplotlib**
- Discussion



Outline for today

1 Introduction

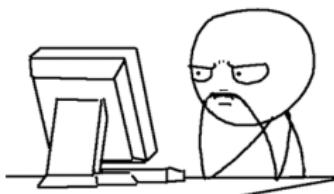
2 Data Types

3 Reference Materials



As programmers, while coding¹

It doesn't work...Why?



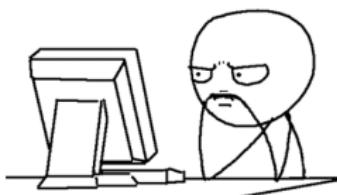
¹Optimates stories - William Yeung's journey to becoming a Graduate Developer Optimization Group.

<https://www.optimization.co.nz/news-and-thoughts/optimates-stories-william-yeungs-journey-to-become-our-newest-graduate-developer/>. (Accessed on 08/06/2019).

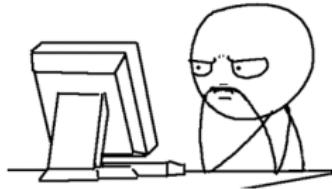


As programmers, while coding¹

It doesn't work...Why?



It works...Why?



¹Optimates stories - William Yeung's journey to becoming a Graduate Developer Optimization Group.

<https://www.optimates.co.nz/news-and-thoughts/optimates-stories-william-yeungs-journey-to-become-our-newest-graduate-developer/>. (Accessed on 08/06/2019).



Are we Python programmers by default?



Are we Python programmers by default?

How many of you wrote at least one Python program?



Are we Python programmers by default?

How many of you wrote at least one Python program?

2 + 3



Are we Python programmers by default?

How many of you wrote at least one Python program?

2 + 3

Python code to add numbers 2 and 3



Are we Python programmers by default?

How many of you wrote at least one Python program?

2 + 3

Python code to add numbers 2 and 3

C Equivalent

```
#include<stdio.h>

int main()
{
    printf("%d", 2+3);

    return 0;
}
```



Are we Python programmers by default?

How many of you wrote at least one Python program?

2 + 3

Python code to add numbers 2 and 3

C Equivalent

```
#include<stdio.h>
int main()
{
    printf("%d", 2+3);
    return 0;
}
```

Java Equivalent

```
class Add
{
    public static void main(String args[])
    {
        System.out.println(2+3);
    }
}
```



Are we Python programmers by default?

How many of you wrote at least one Python program?

2 + 3

Python code to add numbers 2 and 3

C Equivalent

```
#include<stdio.h>
int main()
{
    printf("%d", 2+3);
    return 0;
}
```

Java Equivalent

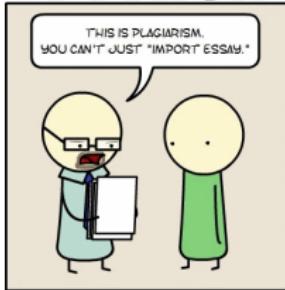
```
class Add
{
    public static void main(String args[])
    {
        System.out.println(2+3);
    }
}
```

Which is better?

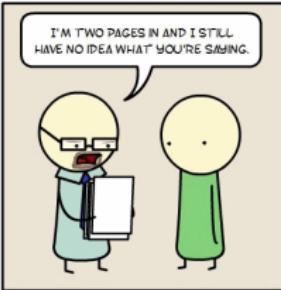


If programming languages were essays²

PYTHON



JAVA



ASSEMBLY



C

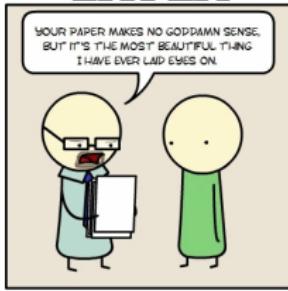
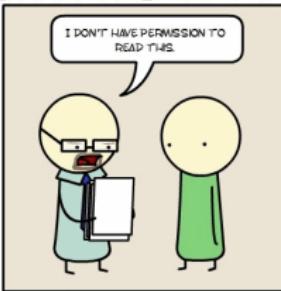
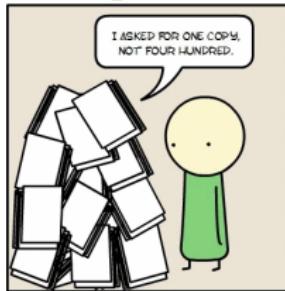


C++

UNIX SHELL

LATEX

HTML

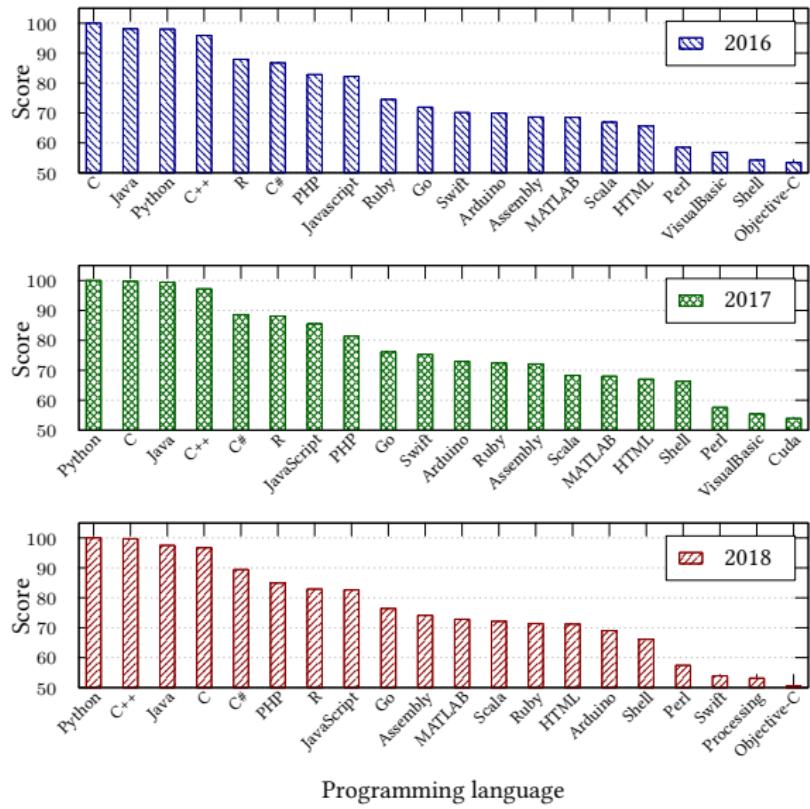


2010-2011. SOMETHINGHAPPENSTHATISLX.COM

² If programming languages were essays... : programming. https://www.reddit.com/r/programming/comments/fqtqk/if_programming_languages_were_essays/. (Accessed on 08/06/2019).



IEEE Spectrum ranking on languages





The Python language



“The joy of coding Python should be in seeing short, concise, readable classes that express a lot of action in a small amount of clear code – not in reams of trivial code that bores the reader to death.”

– Guido van Rossum



The Python language



"The joy of coding Python should be in seeing short, concise, readable classes that express a lot of action in a small amount of clear code – not in reams of trivial code that bores the reader to death."

– Guido van Rossum

- Designed by **Guido van Rossum** in early 1990s
- Name from *Monty Python's Flying Circus*
- Maintained by Python Software Foundation ([PSF](#))
- Multi-paradigm language
- Licensed under Python Software Foundation Licence
- Latest stable releases: Python 3.7.4 and Python 2.7.16
- Web: www.python.org





Design philosophy

The zen of Python

- | | |
|---|---|
| Beautiful is better than ugly. | Errors should never pass silently. |
| Explicit is better than implicit. | Unless explicitly silenced. |
| Simple is better than complex. | In the face of ambiguity, refuse the temptation to guess. |
| Complex is better than complicated. | There should be one—and preferably only one—obvious way to do it. |
| Flat is better than nested. | Now is better than never. |
| Sparse is better than dense. | Although never is often better than “right” now. |
| Readability counts. | If the implementation is hard to explain, it’s a bad idea. |
| Special cases aren’t special enough to break the rules. | If the implementation is easy to explain, it may be a good idea. |
| Although practicality beats purity. | |



Features

- Bytecode interpreted language
- Focus on readability
- Support to multiple programming paradigms
 - Structured programming
 - Object oriented programming
 - Aspect oriented programming
- Dynamism in
 - Typing
 - Name resolution
 - Memory management



Python IDEs





'A' for 'Apple'

IDLE

- Interactive DeveLopment Environment
- Interactive shell for Python code
- Includes editor for Python scripts
- Developed using Python and *tkinter* package



'A' for 'Apple'

IDLE

- Interactive DeveLopment Environment
- Interactive shell for Python code
- Includes editor for Python scripts
- Developed using Python and *tkinter* package

>>>



'A' for 'Apple'

IDLE

- Interactive DeveLopment Environment
- Interactive shell for Python code
- Includes editor for Python scripts
- Developed using Python and *tkinter* package

```
>>>  
>>> print("hello, world")
```



'A' for 'Apple'

IDLE

- Interactive DeveLopment Environment
- Interactive shell for Python code
- Includes editor for Python scripts
- Developed using Python and *tkinter* package

```
>>>  
>>> print("hello, world")  
hello, world
```



Data model

- Python considers data as **objects**
- Object has
 - 1 Identity
 - 2 Type
 - 3 Value

```
>>> a = 2
```

- Objects' identity and type cannot be changed

```
>>> id(a)
```

```
18653568
```

```
>>> type(a)
```

```
<type 'int'>
```

- Two types of objects
 - 1 **Mutable:** Value can be changed (Ex. list, dictionary)
 - 2 **Immutable:** Value cannot be changed (Ex. string, tuple, int, float, long)



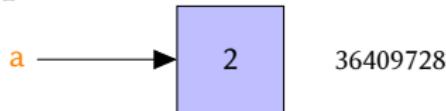
Memory management

```
>>> a=2
```



Memory management

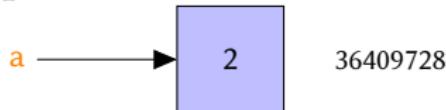
```
>>> a=2
```





Memory management

```
>>> a=2
```

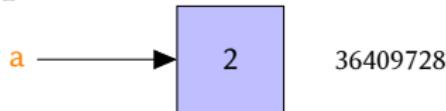


```
>>> b=2
```

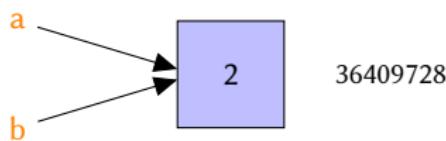


Memory management

```
>>> a=2
```



```
>>> b=2
```



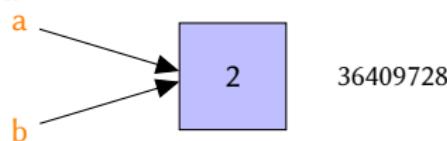


Memory management

```
>>> a=2
```



```
>>> b=2
```



```
>>> a = a + 1
```

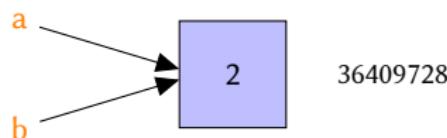


Memory management

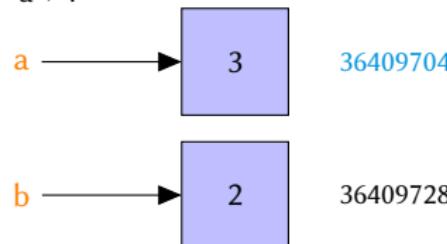
```
>>> a=2
```



```
>>> b=2
```



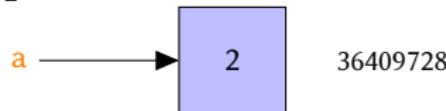
```
>>> a = a + 1
```



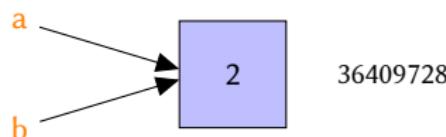


Memory management

```
>>> a=2
```

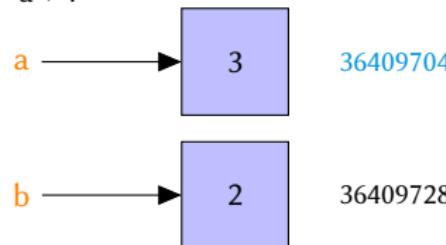


```
>>> b=2
```



```
>>> b = 5
```

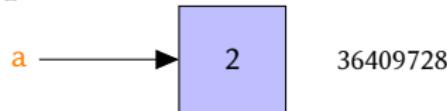
```
>>> a = a + 1
```



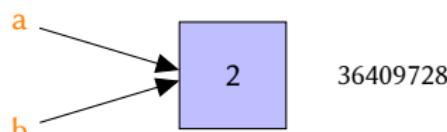


Memory management

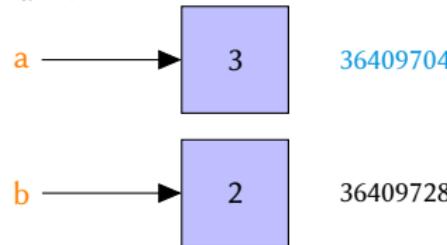
```
>>> a=2
```



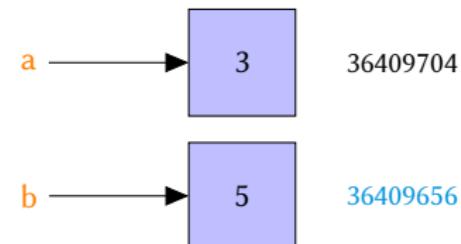
```
>>> b=2
```



```
>>> a = a + 1
```



```
>>> b = 5
```





Data types

- *None*: Absence of a value



Data types

- *None*: Absence of a value
- *Numbers*
 - 1 *Integers*
 - *Plain integers*: Numbers with limited range
 - *Long integers*: Numbers with unlimited range
 - *Boolean*: Truth values (True or False)
 - 2 *Floating point numbers*: Double precision floating point numbers
 - 3 *Complex numbers*: Pair of double precision floating point numbers



Data types

- **None:** Absence of a value
- **Numbers**
 - 1 *Integers*
 - *Plain integers:* Numbers with limited range
 - *Long integers:* Numbers with unlimited range
 - *Boolean:* Truth values (True or False)
 - 2 *Floating point numbers:* Double precision floating point numbers
 - 3 *Complex numbers:* Pair of double precision floating point numbers
- **Sequences:** Finite ordered set of items **indexed by non-negative numbers**
 - 1 *String:* Items are characters
 - 2 *Unicode:* Items are Unicode units
 - 3 *Tuple:* Contains arbitrary python objects
 - 4 *List:* Contains arbitrary python objects



Data types

- **None:** Absence of a value
- **Numbers**
 - 1 *Integers*
 - *Plain integers:* Numbers with limited range
 - *Long integers:* Numbers with unlimited range
 - *Boolean:* Truth values (True or False)
 - 2 *Floating point numbers:* Double precision floating point numbers
 - 3 *Complex numbers:* Pair of double precision floating point numbers
- **Sequences:** Finite ordered set of items **indexed by non-negative numbers**
 - 1 *String:* Items are characters
 - 2 *Unicode:* Items are Unicode units
 - 3 *Tuple:* Contains arbitrary python objects
 - 4 *List:* Contains arbitrary python objects
- **Mappings**
 - 1 *Dictionary:* Finite objects **indexed by arbitrary index**



Numbers + Operators

```
>>> 2+3
```

```
5
```

```
>>> 7+3.5
```

```
10.5
```

```
>>> 5/2
```

```
2.5
```

```
>>> 2**64
```

```
18446744073709551616
```

```
>>> 10 % 4
```

```
2
```

```
>>> 5 // 2.0
```

```
2.0
```

Arithmetic operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Floor division
%	Modulus
**	Exponent



Operators cont'd

```
>>> a = 3
```

```
>>> b = 5.3
```

```
>>> c = 3
```

```
>>> a > b
```

False

```
>>> a != b
```

True

```
>>> a == c
```

True

```
>>> a == c and b > c
```

True

```
>>> a != c or b == c
```

False

```
>>> a ^ c
```

0

Comparison operators

<, >	less than, greater than
<=, >=	less than or equal, greater than or equal
==	is equal
!=	Not equal

Relational operators

and	Logical AND
or	Logical OR
not	Negation

Bitwise operators

&	Bitwise AND
	Bitwise OR
^	Bitwise XOR
~	1's complement
<<, >>	Binary left-shift and right-shift



Operators cont'd

```
>>> a = 3
```

```
>>> b = 5
```

```
>>> c = a + 2
```

```
>>> a is b
```

False

```
>>> b is c
```

True

```
>>> 2 in [1, 2, 3]
```

True

```
>>> 'p' in 'world'
```

False

Identity operators

is	Returns True if identities of two objects are equal
is not	Returns True if identities of two objects are not equal

Membership operators

in	Returns True if an item is present in a sequence
not in	Returns True if an item is absent in a sequence



“Python is a truly wonderful language. When somebody comes up with a good idea it takes about 1 minute and five lines to program something that almost does what you want. Then it takes only an hour to extend the script to 300 lines, after which it still does almost what you want.” – Jack Jansen



String

- Sequence of characters represented in ASCII
- Immutable

0 1 2 3 4

“hello” ⇒ [‘h’ ‘e’ ‘l’ ‘l’ ‘o’]

Basic Operations

```
>>> a = "hello"
>>> b = "world"
>>> print(a[0])
'h'
>>> print(b[-1])
'd'
>>> c = a + b
>>> print(c)
'helloworld'
>>> a[2] = 'Z'
>>> d = c[2:5]
```

String Functions

```
>>> print(d)
'illo'
>>> print(2 * a)
'hellohello'
>>> p = "ab cb ef"
>>> len(p)
8
>>> q = p.split()
>>> print(q)
['ab', 'cb', 'ef']
```

```
>>> p.split('b')
['a ', ' c', ' ef']
>>> n = 'hello\n'
>>> n.strip()
'hello'
>>> n.strip('h')
ello
>>> n.find('o')
4
>>> str(2.345)
'2.345'
```



List

- Arbitrary objects separated by comma within []
- Mutable

	0	1	2	3	4
[1, 3.4, 'a', 2, 'cd']	1	3.4	'a'	2	'cd'

Basic Operations

```
>>> a = [1, 3.4, 'a', 2, 'cd']
>>> a[2]
>>> a[2:4]
>>> b = a
>>> c = a[:]
>>> print(b)
>>> a[2] = 10
>>> print(a)
>>> print(b)
>>> print(c)
>>> c = c + [9, 10]
>>> print(c)
```

List Functions

```
>>> a.append('pqr')
>>> print(a)
>>> a.reverse()
>>> print(a)
>>> b = [11, 10, 13, 12, 15]
>>> b.sort()
>>> b.remove(10)
>>> b.pop(3)
>>> b.insert(4, 20)
>>> len(a)
>>> max(b)
>>> min(b)
```



Set

- Unordered collection of unique elements
- Elements should be **immutable objects**
- Equivalent to sets in mathematics
- Mutable object

Basic Operations

```
>>> a = {1, 3.4}  
>>> type(a)  
>>> print(a)  
>>> print(a[1])  
>>> a.add(4)  
>>> print(a)  
>>> b = {4, 5, 6}  
>>> c = a | b # union  
>>> print(c)  
>>> d = a & b # intersection
```

```
>>> print(d)  
>>> a.add((10, 11))  
>>> e = a - b # difference  
>>> print(e)  
>>> e = a ^ b # sym. diff  
>>> print(e)  
>>> p = [7, 8]  
>>> a.add(p)  
>>> len(a)  
>>> print(a <= b) # subset  
>>> print(a >= b) # superset
```



Tuple

- Items are arbitrary objects separated by comma within ()
 - Immutable
-

```
>>> t = (1, 2, 'hello')
>>> print(t[0])
>>> t[1] = 'world'
>>> p = [0, 1, 2]
>>> u = (4, p)
>>> p.append('hello')
>>> print(u)
>>> z = t + u
>>> print(z)
>>> len(z)
```



Dictionary

- Finite set of objects indexed by arbitrary values
 - Items are represented as **key:value** pairs
 - Mutable object
 - Objects which **cannot** be accepted as keys
 - Mutable types compared by values such as lists and dictionaries
 - Uses hashing for efficient memory access

```
>>> d = {'a': 'apple', 'b': 'boy', 1: 'one', 2: 'two', 3:[3, 4, 5]}
```

```
>>> print(d)
```

```
{'a': 'apple', 1: 'one', 2: 'two', 'b': 'boy', 3: [3, 4, 5]}
```

```
>>> print(d['b'])
```

‘boy’

```
>>> d[1] = 'hello'
```

```
>>> d['p'] = 'pen'
```

```
>>> d[(0, 1)] = 'key is a tuple'
```

```
>>> d.keys()
```

```
>>> d.values()
```

```
>>> d[[0, 2]] = "key is a list"
```



Input from keyboard and type casting

input()

- Input numbers from keyboard

```
>>> n = input('Enter the number: ')
```

```
Enter the number: 4
```

```
>>> print(n)
```

```
>>> type(n)
```

Type casting

- Conversion of data of a type to another

```
>>> p = 'hello'
```

```
>>> s = '3.14'
```

```
>>> l = list(p)
```

```
>>> f = float(s)
```

```
>>> print(l)
```

```
>>> int(s)
```

```
>>> t = tuple(p)
```

```
>>> s = str(a) + ' ' + p + ' ' + str(f)
```

```
>>> print(t)
```

```
>>> print(s)
```

```
>>> a = 10
```

```
>>> q = '%d %s %f' % (a, p, f)
```

```
>>> s = str(a)
```

```
>>> print(q)
```

```
>>> type(s)
```

```
>>> b = [1, 2, 'abc']
```

```
>>> c = str(b)
```



Keywords

- Reserved words in the language
 - Not advised to use as variable names
-

and	as	assert	break	exec	is	
del	elif	else	except	in	try	return
from	global	if	import	raise	def	
not	or	pass	print	continue	for	
while	with	yield	class	finally	lambda	



“In many ways, it’s a dull language, borrowing solid old concepts from many other languages & styles: boring syntax, unsurprising semantics, few automatic coercions, etc etc. But that’s one of the things I like about Python.” – Tim Peters



Reference materials

Official Python documentation is available at <https://docs.python.org>



“How to Think Like a Computer Scientist: Learning with Python 3” -
Peter Wentworth, Jeffrey Elkner, Allen Downey, and Chris Meyers

“Dive into Python 3” - Mark Pilgrim

“A Byte of Python” - Swaroop C. H.

Learn Python - [tutorialspoint](#)



Thank you.

sarath.babu.2014@ieee.org