Introduction to Computing

Basics of Python Programming – I

Malay Bhattacharyya

Associate Professor

MIU, CAIML, TIH
Indian Statistical Institute, Kolkata

August, 2024

Basic characteristics

- Python is a free and open source language, first written in C.

Basic characteristics

- Python is a free and open source language, first written in C.
- Python is an interpreted language.

Basic characteristics

- Python is a free and open source language, first written in C.
- Python is an interpreted language.
- Python is not a free-form language.

Basic characteristics

- Python is a free and open source language, first written in C.
- Python is an interpreted language.
- Python is not a free-form language.
- Python is a strongly typed language.

Basic characteristics

- Python is a free and open source language, first written in C.
- Python is an interpreted language.
- Python is not a free-form language.
- Python is a strongly typed language.
- Python is an object-oriented language but it also supports procedural oriented programming.

Basic characteristics

- Python is a free and open source language, first written in C.
- Python is an interpreted language.
- Python is not a free-form language.
- Python is a strongly typed language.
- Python is an object-oriented language but it also supports procedural oriented programming.
- Python is a high level language.

Basic characteristics

- Python is a free and open source language, first written in C.
- Python is an interpreted language.
- Python is not a free-form language.
- Python is a strongly typed language.
- Python is an object-oriented language but it also supports procedural oriented programming.
- Python is a high level language.
- Python is a portable and cross-platform language.

Basic characteristics

- Python is a free and open source language, first written in C.
- Python is an interpreted language.
- Python is not a free-form language.
- Python is a strongly typed language.
- Python is an object-oriented language but it also supports procedural oriented programming.
- Python is a high level language.
- Python is a portable and cross-platform language.
- Python is an extensible language.

Basic characteristics

- Python is a free and open source language, first written in C.
- Python is an interpreted language.
- Python is not a free-form language.
- Python is a strongly typed language.
- Python is an object-oriented language but it also supports procedural oriented programming.
- Python is a high level language.
- Python is a portable and cross-platform language.
- Python is an extensible language.

Basic characteristics

- Python is a free and open source language, first written in C.
- Python is an interpreted language.
- Python is not a free-form language.
- Python is a strongly typed language.
- Python is an object-oriented language but it also supports procedural oriented programming.
- Python is a high level language.
- Python is a portable and cross-platform language.
- Python is an extensible language.

Note: The reference implementation CPython is now available on GitHub (see: `https://github.com/python/cpython`). This is no more updated.

Knowing Python in a better way

Looking into the help manual:

```
help(<built-in object>)
help(<built-in function>)
```

Knowing Python in a better way

Looking into the help manual:

```
help(<built-in object>)
help(<built-in function>)
```

Looking into the source code:

```
import inspect
inspect.getfullargspec(<built-in function>)
inspect.getsource(<live object>)
```

Knowing Python in a better way

Looking into the help manual:

```
help(<built-in object>)
help(<built-in function>)
```

Looking into the source code:

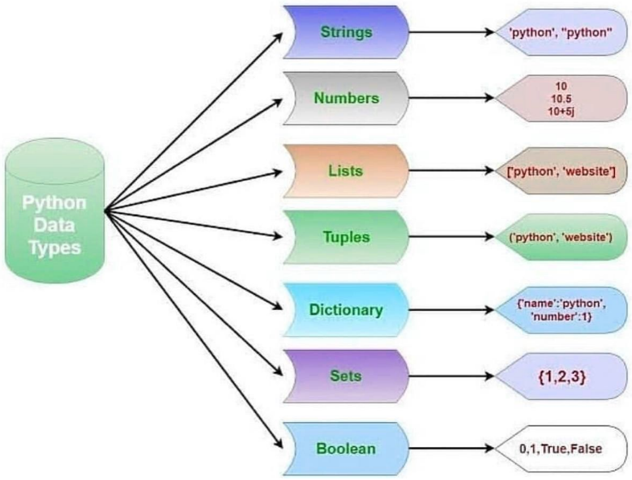
```
import inspect
inspect.getfullargspec(<built-in function>)
inspect.getsource(<live object>)
```

Note: The official Python source releases are available at: `https://www.python.org/downloads/source`.

Knowing Python in a better way

The full standard library documentation of Python 3 can be found at: `https://docs.python.org/3/contents.html`

The data types in Python



Strings

Substring of a string:

```
str = "malfunctioning" # Indexing starts with 0
print(str[3:6]) # Elements indexed 3 through 5
```

Strings

Substring of a string:

```
str = "malfunctioning" # Indexing starts with 0
print(str[3:6]) # Elements indexed 3 through 5
```

Output:

```
fun
```

Strings

Substring of a string:

```
str = "malfunctioning" # Indexing starts with 0
print(str[3:6]) # Elements indexed 3 through 5
```

Output:

```
fun
```

Splitting a string:

```
str = "I don't love Python, the snake."
print(str.split(" "), str.split(", "))
```

Strings

Substring of a string:

```
str = "malfunctioning" # Indexing starts with 0
print(str[3:6]) # Elements indexed 3 through 5
```

Output:

```
fun
```

Splitting a string:

```
str = "I don’t love Python, the snake."
print(str.split(" "), str.split(", "))
```

Output:

```
['I', "don’t", 'love', 'Python,', 'the', 'snake.']
["I don’t love Python", 'the snake.']
```

Strings

What is the connection between the outputs of the following two cases?

```
str = "BStat_I"
print(hex(id(str)))
```

```
print(hex(id("BStat" + "_" + "I")))
```

Strings

What is the connection between the outputs of the following two cases?

```
str = "BStat_I"
print(hex(id(str)))
```

```
print(hex(id("BStat" + "_" + "I")))
```

They are the same!!!

Note: `id()` is used to return the identity of an object in Python.

Strings

Negative indexing:

```
str = 'please step on no pets'
strrev = str[::-1]
print(strrev)
```

Strings

Negative indexing:

```
str = 'please step on no pets'
strrev = str[::-1]
print(strrev)
```

Output:

```
step on no pets esaelp
```

Strings

Python 3 standard library documentation on the common string operations can be found at:
`https://docs.python.org/3/library/string.html`

The source code for the string module is available here: `https://://github.com/python/cpython/tree/3.11/Lib/string.py`

Numbers

Numbers are of three types – int, float, complex.

Examples:

```
int(1)          1
1.              +3.45e67
float(1)        +3.45e-67
.1              -3.45e-67
1.2345          3.45e67
-0.6789         .00345e-32
+.4560          1e-15
-.1234          1e+15
complex(1,-1)   (1-1j)
```

Numbers

Numbers are of three types – int, float, complex.

Examples:

```
int(1)          1
1.              +3.45e67
float(1)        +3.45e-67
.1              -3.45e-67
1.2345          3.45e67
-0.6789         .00345e-32
+.4560          1e-15
-.1234          1e+15
complex(1,-1)   (1-1j)
```

- Do not use commas as thousand-separators.
- At times behavior may be counter-intuitive.

Numbers

What will be the output of the following?

```
a = 10
b = 10
c = 3+7
d = 11
e = 12
print(hex(id(a)), hex(id(b)), hex(id(c)))
print(hex(id(d)), hex(id(e)))
```

Output:

```
0x7fe59418c210 0x7fe59418c210 0x7fe59418c210
0x7fe59418c230 0x7fe59418c250
```

Numbers

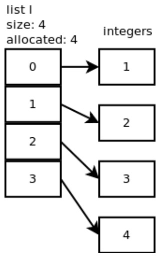
What will be the output of the following?

```
a = complex(1,-1)
b = 1-1j
print(a,b)
print(hex(id(a)), hex(id(b)))
c = b * complex(1,1)
d = 2
print(c,d)
print(hex(id(c)), hex(id(d)))
```

Output:

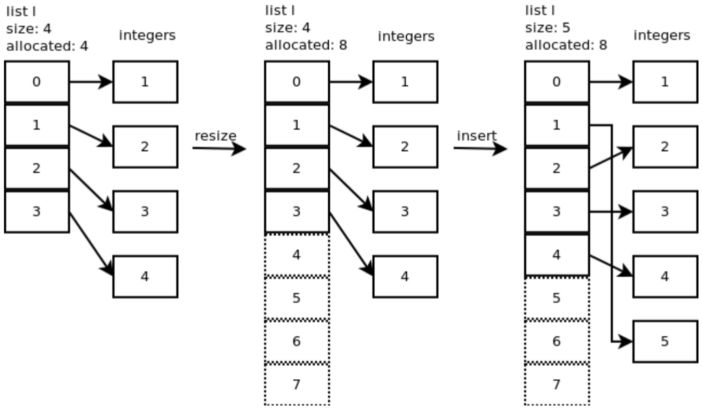
```
(1-1j) (1-1j)
0x7f7f6445a770 0x7f7f6445a6b0
(2+0j) 2
0x7f7f6445a6f0 0x7f7f645bc110
```

Lists



- *Length* of the list $= n$.
- n *linked* memory locations that gets dynamically reallocated.
- The elements are heterogeneous.
- *Elements* can be mapped to each of the n memory locations.
- Elements are indexed 0 through $n - 1$ ($\equiv -n$ through -1).

Lists – Adding elements



```
ls = []
ls.append(<data>) # Inserts at the end
ls.insert(<index>, <data>) # Inserts at the <index>
```

Lists – Adding elements

What will be the output of the following program?

```
ls = [1, 2, 3, 4, 5]
ls.insert(3, 20) # Inserts 20 at index 3
print(ls)
ls.insert(20, 3) # Inserts 3 at index 6 (not at index 20)
print(ls)
```

Lists – Adding elements

What will be the output of the following program?

```
ls = [1, 2, 3, 4, 5]
ls.insert(3, 20) # Inserts 20 at index 3
print(ls)
ls.insert(20, 3) # Inserts 3 at index 6 (not at index 20)
print(ls)
```

Output:

```
[1, 2, 3, 20, 4, 5]
[1, 2, 3, 20, 4, 5, 3]
```

Note: If the list insertion index is out of range then the maximum possible range is taken.

Lists – Adding elements

What will be the output of the following program?

```
ls = [1, 2, 3, 4, 5]
ls[3] = 10 # Assigns 10 at index 3
print(ls)
ls[10] = 3 # Assigns 3 at index 10
print(ls)
```

Lists – Adding elements

What will be the output of the following program?

```
ls = [1, 2, 3, 4, 5]
ls[3] = 10 # Assigns 10 at index 3
print(ls)
ls[10] = 3 # Assigns 3 at index 10
print(ls)
```

Output:

```
[1, 2, 3, 10, 5]
```

Error

Note: List assignment index cannot be out of range.

Lists – Adding multiple elements

```
ls1 = [1, 2, 3, 4]
ls2 = [8, 9, 10, 11]
ls1.extend(ls2) # Extends ls1 by appending ls2
print(ls1)
ls1[4:4] = [5, 6, 7] # Inserts elements from a list
print(ls1)
ls1[4] = [4.5, 5, 5.5] # Inserts a list
print(ls1)
```

Lists – Adding multiple elements

```
ls1 = [1, 2, 3, 4]
ls2 = [8, 9, 10, 11]
ls1.extend(ls2) # Extends ls1 by appending ls2
print(ls1)
ls1[4:4] = [5, 6, 7] # Inserts elements from a list
print(ls1)
ls1[4] = [4.5, 5, 5.5] # Inserts a list
print(ls1)
```

Output:

```
[1, 2, 3, 4, 8, 9, 10, 11]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[1, 2, 3, 4, [4.5, 5, 5.5], 6, 7, 8, 9, 10, 11]
```

Lists – Adding multiple elements

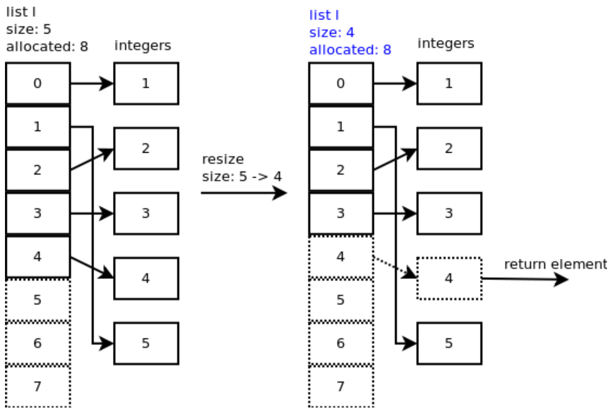
```
ls1 = [1, 2, 3, 4]
ls2 = [8, 9, 10, 11]
ls1.extend(ls2) # Extends ls1 by appending ls2
print(ls1)
ls1[4:4] = [5, 6, 7] # Inserts elements from a list
print(ls1)
ls1[4] = [4.5, 5, 5.5] # Inserts a list
print(ls1)
```

Output:

```
[1, 2, 3, 4, 8, 9, 10, 11]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[1, 2, 3, 4, [4.5, 5, 5.5], 6, 7, 8, 9, 10, 11]
```

Note: You may use an iterative control flow.

Lists – Removing elements



```
ls = []
<data> = ls.pop() # Deletes from the end
<data> = ls.pop(<index>) # Deletes from the <index>
<data> = ls.remove(<index>) # Deletes from the <index>
```

Lists – Removing multiple elements

You may use an iterative control flow.

Lists

Creating list of lists:

```
ls = [[]] * 5
print(ls)
```

Lists

Creating list of lists:

```
ls = [[]] * 5
print(ls)
```

Output:

```
[[], [], [], [], []]
```

Lists of sublists

What will be the output of the following program?

```
ls1 = [[] for i in range(3)]
ls1[0].append(10)
ls1[1].append(20)
ls1.append(30)
print(ls1)
ls2 = [[]]*3
ls2[0].append(10)
ls2[1].append(20)
ls2.append(30)
print(ls2)
```

Lists of sublists

What will be the output of the following program?

```
ls1 = [[] for i in range(3)]
ls1[0].append(10)
ls1[1].append(20)
ls1.append(30)
print(ls1)
ls2 = [[]]*3
ls2[0].append(10)
ls2[1].append(20)
ls2.append(30)
print(ls2)
```

```
[[10], [20], [], 30]
[[10, 20], [10, 20], [10, 20], 30]
```

Lists

Python Standard Library documentation on the extended list operations can be found at: `https://docs.python.org/3/tutorial/datastructures.html`

Converting Strings to Lists

Strings are just like Lists. One can convert a String into the List by passing the String as an argument to the List as follows.

```
str = "Python 3"
list(str)
```

Output: ['P', 'y', 't', 'h', 'o', 'n', ' ', '3']

Tuples

Tuples are like lists but immutable in nature, i.e. the elements in the tuple cannot be added or removed once created.

```
tp = ('Language', 'Python')
print(tp, tp[1])
```

Output: ('Language', 'Python') Python

Tuples

Tuples are like lists but immutable in nature, i.e. the elements in the tuple cannot be added or removed once created.

```
tp = ('Language', 'Python')
print(tp, tp[1])
```

Output: ('Language', 'Python') Python

Converting lists to tuples:

```
ls = [1, 2, 4, 5, 6]
Tuple = tuple(list)
```

Tuples

Python Standard Library documentation on the extended tuple operations can be found at: `https://docs.python.org/3/tutorial/datastructures.html`

Dictionary

Creating a Dictionary:

```
dc = {'Course': 'Business Analytics', 1: [28, 29]}
print("The created dictionary: ", dc)
```

Output:

```
The created dictionary: 'Course': 'Business Analytics', 1: [28, 29]
```

Dictionary

Accessing an element from the dictionary:

```
print(dc['Course'], dc[1]) # Accessed by the key
print(dc.get('Course'), dc.get(1))
```

Output:

```
Business Analytics [28, 29]
Business Analytics [28, 29]
```

Dictionary

Accessing an element from the dictionary:

```
print(dc['Course'], dc[1]) # Accessed by the key
print(dc.get('Course'), dc.get(1))
```

Output:

```
Business Analytics [28, 29]
Business Analytics [28, 29]
```

Deleting an element from the dictionary:

```
dc.pop('Course')
print("The current dictionary: ", dc)
```

Output:
The current dictionary: 1: [28, 29]

Dictionary

Python Standard Library documentation on the extended dictionary operations can be found at: `https://docs.python.org/3/tutorial/datastructures.html`

Sets

Sets are (ordered or unordered) collection of data items that is mutable and does not allow any duplicate element.

```
st = {'day', 1, 2, 'for', 'Python', 'Python'}
print(st)
st.add('language')
print(st)
st.remove('for')
print(st)
```

Output:

```
{'Python', 1, 2, 'for', 'day'}
{'Python', 1, 2, 'for', 'day', 'language'}
{'Python', 1, 2, 'day', 'language'}
```

Note: Sets are not subscriptable.

Sets

Python Standard Library documentation on the extended set operations can be found at: `https://docs.python.org/3/tutorial/datastructures.html`

Boolean values

Any non-zero value is treated as True and zero is treated as False.

Examples:

0	False	0e10	False
1	True	'A'	True
6 - 2 * 3	False	"A"	True
(6 - 2) * 3	True	'\0'	True
0.0075	True	(0, 0)	True
''	False	""	False

Boolean values

Any non-zero value is treated as `True` and zero is treated as `False`.

Examples:

0	False	0e10	False
1	True	'A'	True
6 - 2 * 3	False	"A"	True
(6 - 2) * 3	True	'\0'	True
0.0075	True	(0, 0)	True
''	False	""	False

Note: The expressions like "x = 0" or "x = 1" will exhibit error.

Mutable and immutable objects

Everything in Python is an object and it is either mutable or immutable.

Mutable and immutable objects

Everything in Python is an object and it is either mutable or immutable.

A mutable object can be changed to a different type after it is created, but an immutable object cannot be changed.

Mutable and immutable objects

Everything in Python is an object and it is either mutable or immutable.

A mutable object can be changed to a different type after it is created, but an immutable object cannot be changed.

- Objects of built-in types like int, float, bool, str, tuple, unicode are immutable.
- Objects of built-in types like list, set, dict are mutable.

Verifying the data type

```
type(3)
```

Output: int

Verifying the data type

```
type(3)
```

Output: int

```
type(3.14)
```

Output: float

Verifying the data type

```
type(3)
```

Output: int

```
type(3.14)
```

Output: float

```
type('pi')
```

Output: str

Verifying the data type

```
type(3)
```

Output: int

```
type(3.14)
```

Output: float

```
type('pi')
```

Output: str

```
type([3.14, 3.142, 3.1428])
```

Output: list

Verifying the data type

```
type(3)
```

Output: int

```
type(3.14)
```

Output: float

```
type('pi')
```

Output: str

```
type([3.14, 3.142, 3.1428])
```

Output: list

```
type(True)
```

Output: bool