

Python 101

Lec00

Python and Primitive Data Types

July 29, 2019

Syllabus

1. Basic Data Types
2. Ifs and Fors
3. Functions
4. Data Structures
5. Classes and OOP*
6. Efficient Programming*
7. Project: Playing QWOP with Genetic Algorithm*
8. Project: Twitter Crawling*

*Subject to Change

WHY Python?

- ▶ Hello, World! in C

```
#include <stdio.h>
int main()
{
    printf("Hello World!\n");
    return 0;
}
```

- ▶ Hello, World! in JAVA

```
public class hello {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

WHY Python?

- ▶ Hello, World! in Brainfuck¹

```
+++++++  
[  
  >++++++  
  >+++++++  
  >+++  
  >+<<<<-  
]  
>++.>+.++++++..++>+.  
<<+++++++.  
>.++.----->+>.
```

¹<https://namu.wiki/w/BrainFuck>

WHY Python?

However in Python..

```
print("Hello, World!")
```

WHY Python?

Python lets you

- ▶ care less about the details.
- ▶ think at the high level.

WHY Python?

Lets Begin!

The Built-ins

[illegible]

Numerals

```
x = 7
big_num = 99999999999999912834092834918234902814
print(type(x)) # int(eger)
print(type(big_num)) # also int...
some_float = 0.1
print(type(0.1)) # float
```

Numerals

- ▶ Python has **UNLIMITED** precision for integers.
(C's intmax is usually 2147483647)
- ▶ There are errors present in floats.

Possible Operations with Integers

```
print(4 == 1) # notice the '=='!  
print(3 + 5) # == 8  
print(3 <= 5) # True  
print(3 > 5) # False  
print(3 * 5) # == 15  
print(3 ** 2) # == 9  
print(0 ** 0) # == 1  
print(10 / 3) # == 3.3333333333333335  
print(10 // 3) # == 3  
print((1000 / 333) * 333) # == 999.9999999999999  
print(3948190283490128349034 * 19038190238120983)  
print(3948190283490128349034 ** 19038190238120983)  
# possible, but takes forever.
```

Strings

```
some_str = "Hello"  
some_otherstr = 'abc'  
some_char = 'c'  
print(type(some_otherstr)) # str(int)  
print(type('c')) # also str
```

Strings

- ▶ Strings are enclosed in matching (') or (")s.(No difference)
- ▶ Note that a single character is also a string.
- ▶ Strings are Immutable

Strings

Possible Operations with Strings

```
print('bacon'.upper())
print('egg bacon'.split())
print('egg,bacon'.split(','))
# print(3 + '777') error!
print(3 + int('777'))
print(3 + float('0.123'))
```

Booleans

True and False.

```
print(type(True)) # bool(ean)
print(type(False)) # bool(ean)
```


Strings

Possible Operations with Booleans

```
print(1==True)
print(0==False)
print(True and False) #equiv to multiplication
print(True or False) #equiv to addition
print(not True)
print(bool(-1)==True)
print(bool(123)==True)
# any non-zero value is evaluated to True.
print(bool(0)==False) # if and only if 0
```

Questions?

Containers

Now we start combining individual data together, using Containers.

Containers

Possible Operations with Containers

```
tup = (1, 'ab', 3)
lst = [1, 2, 'ab']
nums = [55, -1, 934, 123012, -10034]

print('ab' in tup)
# How long would this take?
print(3 in nums)
```

Containers

If we can check membership, it is a container.¹

¹Duck Typing

From Containers to Iterables

The *containers* we have learned are also *iterables*.
Since *iterables* provide MANY useful functions, lets use them.

Iterables

Possible Operations with Iterables

```
tup = (1, 'ab', 3) # tuple
lst = [1, 2, 'ab'] # list
nums = [55, -1, 934, 123012, -10034]
s = 'StringIsalsoIteRaBLe'

print(tup)
print(lst)
print(sum(nums))
print(max(nums))
print(sorted(nums)) # new list
print(sorted(s))
nums.sorted() # changes nums itself
print(nums)
# why not tup.sorted()?
print(len(s))
print(list(s))
```

Iterables

Possible Questions

- ▶ Tuple vs List?
- ▶ `f(lst)` vs `lst.f()`?

Iterables

Possible Questions

- ▶ Tuple vs List?
 - ▶ Tuples are Immutable. Memory efficient, and fast creation.
 - ▶ Lists are Mutable.
- ▶ `f(lst)` vs `lst.f()`? Function vs Method.
 - ▶ Functions are for multiple class of objects
 - ▶ Method is restricted to the object, and change itself.
(e.g. `len('abc')`, `len([1,2,3])` vs `[1,2,3].sort()`)

Iterables

Possible Operations with Iterables (Cont')

```
print(lst + lst)
print(lst + [4])
print(s + 'abc')
print(s * 3)
lst.append(4)
print(lst)
```

Iterables

Subscription

```
tup = (1, 'ab', 3) # tuple
lst = [1, 2, 'ab'] # list
nums = [55, -1, 934, 123012, -10034]
s = 'StringIsalsoIteRaBLe'

print(nums[0])
print(s[-1])
print(s[-2])

nums[1] = 100
print(nums[1])
# s[-1] = 'c', tup[1] = 'abc'??

#unpacking
x, y, z = lst # len(lst) == 3
print(x, y, z)
x, *y = s # len(s) != 2
print(x, y)
```

Iterables

Slicing

```
# getting parts of the lists(or an iterable)
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# lst[from:until:step]
print(lst, len(lst))
print(lst[1:], len(lst[1:]))
print(lst[1:3], len(lst[1:3])) # notice how 3rd
    item is excluded
print(lst[:]) #everything
print(lst[-1:3:-1])
print(lst[3:-1])
print(lst[::-1]) #reverse
print(lst[::2]) # every 2's
print(lst[::3]) # every 3's
```

Creating Iterables

How do we build a list from 1 to 100?

Range

- ▶ `range(stop)` (`== range(0, stop)`)
- ▶ `range(start, stop)`
- ▶ `range(start, stop, step)`
- ▶ `range` is memory efficient, thanks to lazy evaluation.

Getting Inputs

```
x = input()
# x = input("input something")

# do something with x
print(x)
print(type(x))
# input()'s return type is str, so if we want
# numbers:
# x = int(input())
```


Practice

Input: A string

Output: Print True if the string is a palindrome. Print False otherwise.

Practice

Input: A string

Print the number of words in the string, and the word of the string that comes last in alphabetical order.

Practice

Input: A string containing two numbers, separated by a space.

Output: Print the 4th digit of the product of the two numbers,

1. by converting the product to str.
2. without converting the product to str.

Practice

Score	Grade
90...100	A
80...89	B
70...79	C
60...69	D
under 60	F

Input: Score between 0...100

Output: Print the corresponding grade.

For Reviews and Previews

Book