APS106

UNIVERSITY OF TORONTO

Tutorial 5 – Week 6

We'll be starting at the 10 minute mark

if nothing else, write #cleancode

Agenda

- Lecture Review
  - Objects & Methods
  - String Objects & String Methods
- Practice problems

Learning Objectives

After completing this tutorial, learners should:

- understand the notion of object
- understand the notion of method (including method parameters/arguments, returned values)
- understand the connections between objects, methods, and arguments
- know how to call methods on objects
- know what an escape sequence is
- know how to use escape sequence
- know how to convert a value from one representation (i.e., data type) into another
- know how to access string components using the index and slice operations (using both positive and negative indices)
- understand the notion of “mutability” (and its reverse, aka ‘immutability’)
- know several frequently used string methods
- know how to “chain”/compose methods

APS106

UNIVERSITY OF TORONTO

Review of Lecture

Objects and Methods

if nothing else, write #cleancode

Lecture Review: Objects

In Python, (almost) EVERYTHING is an OBJECT!!

- In Python every value, variable, function, etc. is embodied by an object
- How to check if something is an (instance) object:

```
>>> isinstance(4, int)
    True
```

- Each object has specific functions that can be applied to that object. These functions are called **methods !!**
- General syntax: `object_name.method_name(arguments)`

notice the dot !

Remember Lab 1?

Step 1. gain access to the "`Turtle`" class definition that is packaged in module "`turtle`"

Step 2. create an object of type **`Turtle`**, named **`tina`**

```
import turtle

tina = turtle.Turtle()


tina.forward(50)
tina.left(90)
tina.forward(50)
tina.right(90)
tina.forward(90)
tina.right(90)
tina.forward(50)
tina.left(90)
tina.forward(50)

turtle.done()
```

Notice the difference between "`turtle`" and "`Turtle()`"

`class name`

`module name`

Remember Lab 1? (cont)

```
import turtle

tina = turtle.Turtle()

tina.forward(50)
tina.left(90)
tina.forward(50)
tina.right(90)
tina.forward(90)
tina.right(90)
tina.forward(50)
tina.left(90)
tina.forward(50)

turtle.done()
```

- What is the object ?
- What are the methods ?
- What are the method arguments ?

APS106

UNIVERSITY OF TORONTO

Review of Lecture

String Objects & String Methods

if nothing else, write #cleancode

String Objects:

Displaying / Printing Strings

Formatting Strings – Escape Sequences

- The \ is a special character, called an *escape character*
- When used in strings in sequence with other characters, `print()` treats \ as follows:

Escape Sequence	Name	Example	Output
`\n`	newline (ASCII linefeed - LF)	`"How\nare\nyou?"`	How are you?
`\t`	tab (ASCII horizontal tab - TAB)	`'3\t4\t5'`	3 4 5
`\\`	blackslash	`'\\'`	\
`\'`	single quote	`'don\'t'`	don't
`\"`	double quote	`"He says, \"hi\"."`	He says, "hi".

Converting between Data Types

`str()` takes in an object and returns the string representation of that object.

```
>>> str(12.3)
'12.3'
>>> str(12+5)
'17'
>>> str(1+abs(-2))
'3'
```

`int()` and `float()` take in an object and attempt to return its number representation

```
>>> int('42')
42
>>> float('-42')
-42.0
>>> int('1.1')
Traceback (most recent call last):
  Python Shell, prompt 25, line 1
builtins.ValueError: invalid literal for int() with base 10: '1.1'
```

Accessing the Components of a String via indexing

- An **index** is a position in the string
- The syntax of the **index operator** (the bracket notation):

  `String[ind]`, where `ind` is an integer, e.g., `s[5]`, `"Hello"[-3]`

- the index operator **returns** a **string**
- **Indexing starts at position 0 !!** (In Python and other languages, but not all)
- **The index must be an integer !!**
- **The index operator, [ ], does not modify the string it is applied to** !
  ⇒ It only gives access to a string's element..... More on this soon ☺️

Hello

Positive Indices:	0	1	2	3	4
Negative Indices:	-5	-4	-3	-2	-1

```
str="Hello"
```

Indexing Examples

str[0] = 'H'	str[-1] = 'O'
str[1] = 'E'	str[-2] = 'L'
str[2] = 'L'	str[-3] = 'L'
str[3] = 'L'	str[-4] = 'E'
str[4] = 'O'	str[-5] = 'H'

String Slicing

- Syntax of the slicing operator:
  `string [ start : finish ]`
  `string [ start : finish : step ]`
- The slice operator **returns a string !**
- **Indexing starts at position 0 !!** (In Python and other languages, but not all)
- `start, finish,` and `step` must be integers
- The slice operator, **[:], does not modify the string it is applied to !**

	H	e	l	l	o
Positive Indices:	0	1	2	3	4
Negative Indices:	-5	-4	-3	-2	-1

```
str="Hello"
```

str[:] returns 'HELLO'

str[0:] returns 'HELLO'

str[:5] returns 'HELLO'

str[:3] returns 'HEL'

str[0:2] returns 'HE'

str[-3:-1] returns 'LL'

str[-4:-1] returns 'ELL'

str[-5:-3] returns 'HE'

str[-4:] returns 'ELLO'

str[::-1] returns 'OLLEH'

String Indexing vs Slicing - Examples

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

str[0]	returns	'H'	str[:]	returns	'HELLO'
str[1]	returns	'E'	str[0:]	returns	'HELLO'
str[2]	returns	'L'	str[:5]	returns	'HELLO'
str[3]	returns	'L'	str[:3]	returns	'HEL'
str[4]	returns	'O'	str[0:2]	returns	'HE'

str = "HELLO"

H	E	L	L	O
-5	-4	-3	-2	-1

str[-1]	returns	'O'	str[-3:-1]	returns	'LL'
str[-2]	returns	'L'	str[-4:-1]	returns	'ELL'
str[-3]	returns	'L'	str[-5:-3]	returns	'HE'
str[-4]	returns	'E'	str[-4:]	returns	'ELLO'
str[-5]	returns	'H'	str[::-1]	returns	'OLLEH'

indexing vs slicing

Note that the values returned by the index and slice operators are strings

String Indexing vs Slicing Examples (cont')

If we want to extract substrings of length 1, we can use using either the slice or the index operators.

```
>>> "Hello"[1]
'e'
>>> "Hello"[-4]
'e'
```

```
>>> "Hello"[-4:-3]
'e'
>>> "Hello"[1:2]
'e'
```

String Indexing

- when you know the exact index you want to access

String Slicing

- when you know from where to where you want to slice
- For example: In the first scenario, we do not need to know the size of the string, just that it's the third and fourth last characters

If we want to extract substrings of length greater than 1, we can use the slice operator.

"Modifying" Strings

Strings are **IMMUTABLE:**

- Slicing and indexing DO NOT modify the string they are acting on

```
>>> s = 'YOLO'
s[1] = 'P'
Traceback (most recent call last):
  Python Shell, prompt 5, line 1
builtins.TypeError:  'str'  object does not
support item assignment
```

How to "modify" a string ? ⇒ We CANNOT modify strings!

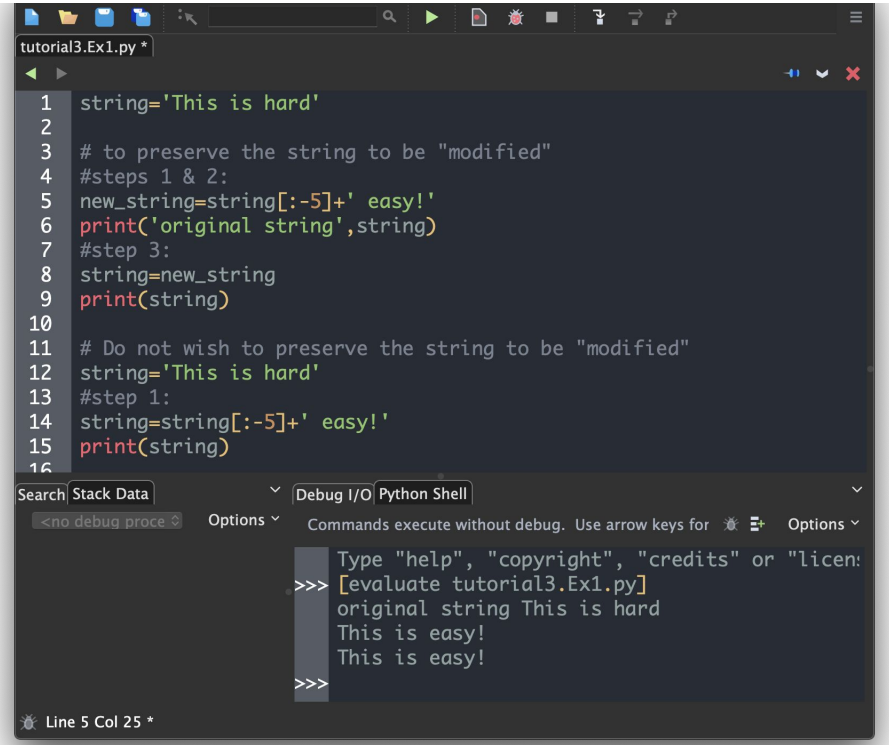
If you need to change the value of a variable and:

A. wish to preserve the existing string value (aka the value to be "modified")

1. Create a new variable
2. Assign the "old" string to the new variable
3. Assign the desired new string to the "original" variable

B. do not wish to preserve the existing string value

1. Assign to the variable the desired new string



String Methods

`find()`, `rfind()`, `replace()`, `lower()`, `upper()`, …

Note: `find()` and `rfind()` are different

- `find()` starts looking from index 0
- `rfind()` starts looking from index -1

Tut3_Week4.py *

```
#String Methods
name="Christine"

print('lower method:',name.lower())

print('upper method:',name.upper())

print('find method: ', name.find('i'))

print('rfind method: ',name.rfind('i'))

print('replace method: ',name.replace('i','y'))

print('name string: ',name)
```

Search | Stack Data | Debug I/O | Python Shell

<no debug process s | Options | Commands execute without debug. Use arrow keys for history. | Options

```
Python 3.8.3 (default, Jul  2 2020, 11:26:31)
[Clang 10.0.0 ]
Type "help", "copyright", "credits" or "license" fo
>>> [evaluate Tut3_Week4.py]
lower method: christine
upper method: CHRISTINE
find method:  3
rfind method:  6
replace method:  Chrystyne
name string:  Christine
>>>
```

Line 1 Col 15 *

Method "Chaining"

- Methods can be "chained"

```
name.upper().replace('Ch','k')
```

- What would the following code return?

```
name.lower().replace('ch','k')
```

Practice Problems

slido

Consider the code. Which expression does NOT produce 'asap'

ⓘ Start presenting to display the poll results on this slide.

Review Practice Problem 1

Q1. Consider the code below. Which expression does NOT produce `'asap'`

```
phrase = "as soon as possible"
```

A
```
phrase[0] + phrase[3] + phrase[8] + phrase[11]
```

B
```
phrase[:4:3] + phrase[-11] + phrase[-8]
```

C
```
phrase[1] + phrase[4] + phrase[8] + phrase[-8]
```

D
```
phrase[-19] + phrase[-16] + phrase[-11] + phrase[-8]
```

slido

What is printed after the code below executes?

ⓘ Start presenting to display the poll results on this slide.

Review Practice Problem 2

Q2. What is printed after the code below executes?

```
def stringFunction(string, substring):
    if (string.find(substring) != -1):
        new_string = string.replace(substring, 'Boo')
        print(string)
        return new_string
    else:
        print(string)
        return 'substring not found'


str1 = 'Go Leafs Go'
substring1 = 'Go'
str2 = stringFunction(str1, substring1)
print(str2)
```

A
```
Go Leafs Go
Boo Leafs Boo
```

B
```
Go Leafs Go
substring not found
```

C
```
Boo Leafs Boo
Boo Leafs Boo
```

D
```
Go Leafs Go
Go Leafs Go
```

E None of the above

Coding Question 1

- **Problem statement**
  - write a function `string_sum()` that takes in a string and returns the **sum** of the digits that appear in the input string. Characters other than digits are ignored.

```
>>> string_sum("PYnative29@#8496")
 38
```

- **Additional requirement:**
  - use a **`while`** loop to solve this question.

Coding Question 2

- **Problem statement**
  - write a function `string_avg()` that takes in a string and returns the **average** of the digits that appear in the input string, rounded to 2 decimals. Characters other than digits are ignored.

```
>>> string_avg("PYnative29@#8496")
    6.33
```

- **Additional requirement:**
  - use a **`while`** loop to solve this question.

slido

Any quesitons?

ⓘ Start presenting to display the poll results on this slide.