# PyLearn-PyTutorial-3.InformalIntro

April 20, 2020

#

Learning Python

## 1 The Python Tutorial -> An Informal Introduction to Python

Link: https://docs.python.org/3/tutorial/introduction.html

## 2 References

- Python documentation
  https://docs.python.org/
- The Python Tutorial
  https://docs.python.org/tutorial/
- The Python Standard Library
  https://docs.python.org/library/index.html#library-index
- The Python Language Reference
  https://docs.python.org/reference/index.html#reference-index
- Extending and Embedding the Python Interpreter
  https://docs.python.org/extending/index.html#extending-index
- Python/C API Reference Manual
  https://docs.python.org/c-api/index.html#c-api-index

### 2.1 Numbers

[1]:
```
3 + 4   # returns int
```

[1]:
```
7
```

[4]:
```
11 / 3   # returns float
```

[4]:
```
3.6666666666666665
```

[5]:
```
9 / 3    # returns float
```

[5]: 3.0

[6]:
```
11 // 3   # returns int which is floor of the result
```

[6]: 3

[17]:
```
11 % 3   # returns int which is remainder of the interger division
```

[17]: 2

[30]:
```
14 ** 3   # power, NOTE: NOT ^
```

[30]: 2744

[30]:
```
14 ^ 3   # this is NOT power, it is XOR, i.e. binary 1110 ^ 0011 equals 1101␣
↪(decimal 13)
```

[30]: 13

[176]:
```
a = -3**2      # BE CAREFUL !!! '**' HAS HIGHER PRECEDENCE THAN '-'
b = (-3)**2
print(a)
print(b)
```

```
-9
9
```

[31]:
```
a = 2
b = 3 + 4
c = _ + a * b   # _ refers to the last PRINTED expression
c
```

[31]: 27

### 2.2 Strings

[43]:
```
"Isn't this nice?"
```

[43]: "Isn't this nice?"

[44]:
```
'He said "hello"!'
```

[44]: 'He said "hello"!'

[45]:
```
'Isn\'t this nice?'   # single quote escaped
```

[45]:
```
"Isn't this nice?"
```

[46]:
```
"He said \"hello\"!"   # double quote escaped
```

[46]:
```
'He said "hello"!'
```

[48]:
```
print("He said \"hello\"!")   # output have no outer quotes
```

```
He said "hello"!
```

[49]:
```
"this is first line\nThis is second line."
```

[49]:
```
'this is first line\nThis is second line.'
```

[50]:
```
print("this is first line\nThis is second line.")
```

```
this is first line
This is second line.
```

[52]:
```
print(r"this is first line\nThis is second line.")   # r to specify that we want a RAW string.
```

```
this is first line\nThis is second line.
```

[53]:
```
print(r"He said \"hello\"!")   # r to specify that we want a RAW string.
```

```
He said \"hello\"!
```

[76]:
```
# String literals can span multiple lines
# \ is to prevent new line
print("""\
    *
   * *
  *   *
   * *
    *\
""")
```

```
   *
  * *
 *   *
  * *
   *
```

[77]:
```
# \ is to prevent new line
# triple single quotes or triple double quotes
print("""A B C
D E F
```

```
G H I
J K L
""")
print('''A B C \
D E F
G H I \
J K L
''')
```

```
A B C
D E F
G H I
J K L

A B C D E F
G H I J K L
```

[80]:
```
print(3 * "ha" + 2 * "he" + 4 * 'ho')
3 * "ha" + 2 * "he" + 4 * 'ho'
```

```
hahahahehehohohoho
```

[80]:
```
'hahahahehehohohoho'
```

[90]:
```
# This only works with two literals though, not with variables or expressions:
# If you want to concatenate variables or a variable and a literal, use +:
print("ha" "he" 'ho')
print("ha""he"'ho')
"ha" "he" 'ho'
```

```
haheho
haheho
```

[90]:
```
'haheho'
```

[89]:
```
# This only works with two literals though, not with variables or expressions:
# If you want to concatenate variables or a variable and a literal, use +:
"ha""he"'ho'
```

[89]:
```
'haheho'
```

[92]:
```
test = ('this is a sentence that is very long but '
        "we do not want to write everything "
        'in one line inside our code!'
        )
print (test)
```

this is a sentence that is very long but we do not want to write everything in one line inside our code!

[100]:
```
# There is no separate character type; a character is simply a string of size one
text = "python"
s1 = text[0]    # first character
s2 = text[-1]   # last character
s3 = text[0] + text[1] + text[2] + text[3] + text[4] + text[5]
s4 = text[-1] + text[-2] + text[-3] + text[-4] + text[-5] + text[-6]
s5 = text[2] + text[1] + text[0] + text[4]
print(s1)
print(s2)
print(s3)
print(s4)
print(s5)
```

```
p
n
python
nohtyp
typo
```

[101]:
```
# slicing allows you to obtain substring
text = "python"
s1 = text[0:2]  # characters from position 0 (included) to 2 (excluded)
s2 = text[2:5]  # characters from position 2 (included) to 5 (excluded)
print(s1)
print(s2)
```

```
py
tho
```

[114]:
```
# an omitted first index defaults to zero, an omitted second index defaults to the size of the string being sliced
text = "python"
s1 = text[:2]
s2 = text[2:]
s3 = text[:4]
s4 = text[4:]
s5 = text[:-3]
s6 = text[-3:]
print(s1)
print(s2)
print(s3)
print(s4)
print(s5)
```

```
print(s6)
```

```
py
thon
pyth
on
pyt
hon
```

[111]:
```
text = "python"
s1 = text[:3] + text[3:]   # always return text
i = 0
s2 = text[:i] + text[i:]   # always return text
i = 2
s3 = text[:i] + text[i:]   # always return text
i = -3
s4 = text[:i] + text[i:]   # always return text
print(s1)
print(s2)
print(s3)
print(s4)
```

```
python
python
python
```

[116]:
```
# out of range slice indexes are handled gracefully when used for slicing
text = "python"
s1 = text[2:20]
print(s1)
```

```
thon
```

[118]:
```
# WILL RESULT ERROR: Attempting to use an index that is too large
text = "python"
s1 = text[6]
print(s1)
```

```
␣
↪---------------------------------------------------------------------------

IndexError                                Traceback (most recent call␣
↪last)

<ipython-input-118-1fed6ab10c30> in <module>
  1 # Attempting to use an index that is too large will result in an␣
↪error
```

```
      2 text = "python"
----> 3 s1 = text[6]
      4 print(s1)


IndexError: string index out of range
```

[119]:
```
# WILL RESULT ERROR: Python strings cannot be changed - they are immutable.
#   Therefore, assigning to an indexed position in the string results in an␣
 ↪error:
text = "This is Python"
text[8] = "C"
print(text)
```

```
      ␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-119-0321b3e3ea1b> in <module>
      2 #   Therefore, assigning to an indexed position in the string␣
↪results in an error:
      3 text = "This is Python"
----> 4 text[8] = "C"
      5 print(text)


TypeError: 'str' object does not support item assignment
```

[121]:
```
# This is okay
text = "This is Python"
text = text[:8] + "C" + text[9:]
print(text)
```

```
This is Cython
```

[122]:
```
text = "This is Python"
print(len(text))
```

```
14
```

### 2.2.1 See also

Text Sequence Type — str
Strings are examples of sequence types, and support the common operations supported by such types.

String Methods
Strings support a large number of methods for basic transformations and searching.

Formatted string literals
String literals that have embedded expressions.

Format String Syntax
Information about string formatting with str.format().

printf-style String Formatting
The old formatting operations invoked when strings are the left operand of the % operator are described in more detail here.

## 2.3 Lists

[130]:
```
squares = [1, 4, 9, 16, 25, 36, 49]
print(squares)
squares
```

```
[1, 4, 9, 16, 25, 36, 49]
```

[130]:
```
[1, 4, 9, 16, 25, 36, 49]
```

[133]:
```
# Like strings (and all other built-in sequence types), lists can be indexed␣
↪and sliced:
squares = [1, 4, 9, 16, 25, 36, 49]
a = squares[0]        # indexing returns the item
b = squares[-1]       # indexing returns the item

list1 = squares[2:]   # slicing returns a new list
list2 = squares[-2:]  # slicing returns a new list
list3 = squares[:2]   # slicing returns a new list
list4 = squares[:-2]  # slicing returns a new list
list5 = squares[2:-2] # slicing returns a new list

print(a)
print(b)
print(list1)
print(list2)
print(list3)
print(list4)
print(list5)
```

```
1
49
[9, 16, 25, 36, 49]
[36, 49]
[1, 4]
[1, 4, 9, 16, 25]
[9, 16, 25]
```

[134]:
```
# All slice operations return a new list containing the requested elements.
# This means that the following slice returns a SHALLOW COPY of the list:
squares = [1, 4, 9, 16, 25, 36, 49]
list = squares[:]    # slicing returns a new list
print(list)
```

```
[1, 4, 9, 16, 25, 36, 49]
```

[142]:
```
# list can be concatenated
squares = [1, 4, 9, 16, 25, 36, 49]
squares = squares + [64, 81, 100, 121]
print(squares)

squares.append(144)
squares.append(13**2)
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169]
```

[137]:
```
# lists are a mutable type, i.e. it is possible to change their content
cubes = [1, 8, 27, 65, 125]
cubes[3] = 64
print(cubes)
```

```
[1, 8, 27, 64, 125]
```

[145]:
```
# lists are a mutable type, i.e. it is possible to change their content
# Assignment to slices is also possible,
# and this can even change the size of the list or clear it entirely:
cubes = [1, 8, 20, 65, 125]
cubes[2:4] = [27, 64]
print(cubes)

cubes = [1, 8, 20, 65, 125]
cubes[2:4] = [27, 64, 99]   # 2 items replaced with 3 items
print(cubes)
```

```
[1, 8, 27, 64, 125]
[1, 8, 27, 64, 99, 125]
```

[148]:
```
# Assignment to slices is also possible,
# and this can even change the size of the list or clear it entirely:
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
print(letters)

letters[2:5] = ['C', 'D', 'E']
print(letters)

letters[2:5] = ['X', 'Y', 'Z', 'W']
print(letters)

letters[2:5] = ['H', 'K']
print(letters)

letters[:] = []   # clear the list by replacing all the elements with an empty␣
 ↪list
print(letters)
```

```
['a', 'b', 'c', 'd', 'e', 'f', 'g']
['a', 'b', 'C', 'D', 'E', 'f', 'g']
['a', 'b', 'X', 'Y', 'Z', 'W', 'f', 'g']
['a', 'b', 'H', 'K', 'W', 'f', 'g']
[]
```

[149]:
```
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
print( len(letters) )
```

```
7
```

[151]:
```
# It is possible to nest lists (create lists containing other lists), for␣
 ↪example:
letters = ['a', 'b', 'c']
num = [1, 2, 3, 4, 5]
names = ['Ali', 'Chong', 'Gobi', 'Alan', 'Ahmad']
x = [letters, num, names]
print(x)

print(x[0])
print(x[1])
print(x[2])

print(x[0])
print(x[1][2])
print(x[2][1:3])
```

```
[['a', 'b', 'c'], [1, 2, 3, 4, 5], ['Ali', 'Chong', 'Gobi', 'Alan', 'Ahmad']]
['a', 'b', 'c']
[1, 2, 3, 4, 5]
```

```
['Ali', 'Chong', 'Gobi', 'Alan', 'Ahmad']
['a', 'b', 'c']
3
['Chong', 'Gobi']
```

### 2.4 First Steps Towards Programming

[170]:
```
# Fibonacci series:
# the sum of two elements defines the next
term, a, b = 1, 0, 1
while term <= 7:
  print('Term ', term, ' is ', b)
  term, a, b = term+1, b, a+b
```

```
Term  1  is  1
Term  2  is  1
Term  3  is  2
Term  4  is  3
Term  5  is  5
Term  6  is  8
Term  7  is  13
```

[173]:
```
# Fibonacci series:
# the sum of two elements defines the next
term, a, b = 1, 0, 1
while term <= 6:
  print(b, end=' => ')
  term, a, b = term+1, b, a+b
print(b)
```

```
1 => 1 => 2 => 3 => 5 => 8 => 13
```

## 3 END OF The Python Tutorial -> An Informal Introduction to Python

[ ]:
```

```