CS 130R: Programming in Python

# 3: Simple I/O & IF statement

Reading: Chapter 2, Chapter 3 pp.49-59
Outline

• Simple I/O
• Line continuation character ‘\’
  and Indentation
• Conversion Functions
• IF statement
Simple I/O

- Using quotes inside strings
- Specifying a Final String to Print
- Creating Triple-quoted strings
- Using escaping sequences with strings
- Concatenating and repeating strings
- Getting user input
Use quotes inside strings

• If you use **double quotes to bookend** your string, you can use as many **single quotes inside** the string as you want.

• If you use **single quotes to bookend** your string, you can use as many **double quotes inside** the string as you want.

```python
print "Hello, 'Python'!"
print 'Hello, "Python"!'```


Printing multiple values

• You can print multiple values with a single call to the “print” function

```python
print("Same", "message", "as before")
print("Same", "message", "as before")
```
Specifying a Final String to Print

• By default, print() function prints a newline character as a final value.

• But you can specify your own final string to be printed.
  
  \[
  \text{print("(Your text)", end = "(Your final string)")}
  \]
  
  # This final string can be a space, a tab, ......

# Prints the text "Here " (including a space after "e"),
# but no newline character
print("Here", end = " ")
print("it is ...")

print("Here", end = "\n")
print("it is ...")

print("Here")
print("it is ...")
Creating Triple-quoted strings

- Triple-quoted strings can span multiple lines
- They print on the screen exactly the way you type them

```python
print(
    """"
    Here
    it is ...
    """
)
```
Using escaping sequences with strings

- Escape sequences allow you to put special characters into your strings.
- Made up of two characters:
  - a backslash + another character
  e.g. `\n`, `'`, `"`, `\t`, `\`

Moving forward a Tab | print(“\t\t\t Hello, Python!”)
Printing a backslash  | print(“\t\t\t Hello, Python! \ \ \ \ \”)  
Inserting a newline   | print(“I say, \n Hello, Python!”)
Inserting a quote     | print(“I say, \n ”Hello, Python!””)
                                print(“I say, \n ’Hello, Python!’”)
## Using escaping sequences with strings

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\</code></td>
<td>Backslash. Prints one backslash.</td>
</tr>
<tr>
<td><code>'</code></td>
<td>Single quote. Prints a single quote.</td>
</tr>
<tr>
<td><code>&quot;</code></td>
<td>Double quote. Prints a double quote.</td>
</tr>
<tr>
<td><code>\a</code></td>
<td>Bell. Sounds the system bell.</td>
</tr>
<tr>
<td><code>\b</code></td>
<td>Backspace. Moves cursor back one space.</td>
</tr>
<tr>
<td><code>\n</code></td>
<td>Newline. Moves cursor to beginning of next line.</td>
</tr>
<tr>
<td><code>\t</code></td>
<td>Horizontal tab. Moves cursor forward one tab stop.</td>
</tr>
</tbody>
</table>
Concatenating and repeating strings

- Concatenating strings means joining several strings together to create a whole new string
- Operator + joins two strings

```
print("You can concatenate two " + "strings with the '+' operator.")
```

- No space or separator is inserted between two joined strings

```
print("Gold" + "Medal")
print("Gold" + "Medal")
print("Gold" + "Medal")
```

Running results:
```
GoldMedal
Gold Medal
Gold Medal
```

Repeating strings: print("Hello" * 10)
Getting user input

• Let the user enter his or her input

```python
name = input("Hi. What's your name? ")
print(name)
print("Hi,", name)
input("\n\nPress the enter key to exit."")

Running result:

>>> Hi. What's your name? Michael
Michael
Hi, Michael

Press the enter key to exit.
Getting user input

input("\n\n Press the Enter key to exit program.")

This line makes the program wait until user presses the “Enter” key.

- **Input always returns a string!**
  
  e.g.  
  
  ```python
  age = input("How old are you? ")
  age = int(age)  # to convert string into integer
  ```

  Or in one line
  
  ```python
  age = int(input("How old are you? "))
  ```
Line continuation character `\`

`\` or the backslash character can be used to continue a statement on the next line.

```python
print("This string is a long string that continues on the next \\ "+ "line to demonstrate the usage of the continuation character.")
```

This string is a long string that continues on the next line to demonstrate the usage of the continuation character.

e.g. ```
mylongstring = "string 1, " + " string 2, " + \\
    "string3, " + "string 4"
``` 

```
print(mylongstring)
```

Running result:

string 1, string 2, string3, string 4
Indentation

• Python uses indentation (i.e., the number of spaces left at the beginning of a statement – start counting spaces from the left border) to create blocks and interpret code.

A block is one or more consecutive lines indented by the same amount.
(Hint: use 4 spaces for each block)
Conversion Functions

- **float(x)** – returns a floating point value by converting x
  
  ```python
  floatFromString = float("12.3")
  12.3 as float number
  ```

- **int(x)** – returns an integer value by converting x
  
  ```python
  intFromString = int("10")
  10 is an integer
  ```

- **str(x)** – returns a string value by converting x
  
  ```python
  stringFromInteger = str(15)
  '15'
  ```
IF statement

IF statement allows for executing a block if a condition is true.

```python
if condition:
    execute block
```

if str == "python":
    print("IF statement is executed...")
IF statement

str = “python”  
if str == “python”:  
    print(“IF statement is executed...”)
(Running result)  IF statement is executed...

Careful:  
if str = “python”:  
    print(“IF statement is executed...”)

This str = “python” is not a condition, it assigns value “python” to string str.
IF statement

• **Comparison Operators:** There are 6 comparison operators. The result of applying the comparison operators is a Boolean - *True or False.*
  – Equal to: ==
  – Not equal to: !=
  – Greater than: >
  – Greater than or equal to: >=
  – Less than: <
  – Less than or equal to: <=

  if operand 1 != operand 2:
  execute block
IF statement

Boolean Operators: There are 3 Boolean operators:

– *not*: unary operator that returns True if the operand is False and vice versa;

– *x and y*: return True when both x and y are true, otherwise return False;

– *x or y*: return True if any of x or y is True, otherwise return False if both x and y are false.
IF ELSE statement

IF ELSE statement allows for executing a block if a condition is true and another block if the condition is false.

    if condition:  
        execute block1  
    else:  
        execute block2

if str == "python":  
    print("IF condition is true...")
else:  
    print("IF condition is false...")
IF ELIF ELSE statement

IF ELIF ELSE statement allows for executing a block based on a decision of which condition is true from several conditions.

```python
if condition1:
    execute block1
elif condition2:
    execute block2
elif condition3:
    execute block3
else:  # execute only when no condition is true
    execute block
```
IF ELIF ELSE statement

```python
if str == "python":
    print("PYTHON condition is true...")
elif str == "java":
    print("JAVA condition is true...")
elif str == "C++":
    print("C++ condition is true...")
else:
    print("NO condition is true...")
print("Done choosing language!")
```

If my string is `str = "java"` program outputs:
JAVA condition is true...
Done choosing language!

If my string is `str = "abc"` program outputs:
NO condition is true...
Done choosing language!
Next lecture ...

- While loops
- For loops