MANIPULATING STRINGS AND LAMBDA FUNCTIONS

CS 3080: Python Programming

University of Colorado Colorado Springs

String literals

  - *’This is a string literal’*
  - *'That is Alice's cat.’*   >>>   *s cat.’ is invalid Python code.*
- Double quotes
  - *“That is Alice's cat.”*
- Escape characters
  - *'Say hi to Bob**\'**s mother.’*
- Raw string
  - *print(**r**'That is Carol\'s cat.’)*

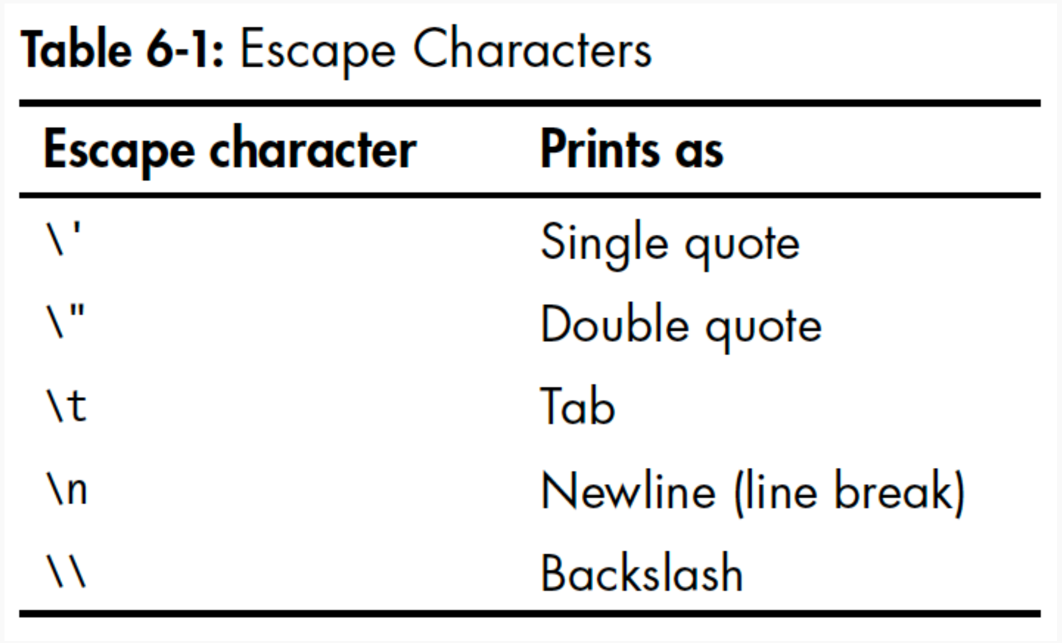


Table 6-1: Escape Characters

Escape character	Prints as
`\'`	Single quote
`\"`	Double quote
`\t`	Tab
`\n`	Newline (line break)
`\\`	Backslash

Full list of escape characters: https://www.quackit.com/python/reference/python_3_escape_sequences.cfm

String literals

Be careful of copy/paste from Powerpoint… quotes are special characters in Microsoft products…

- Multiline string
  - *print('''Dear Alice,*

    Eve's cat has been arrested for catnapping, cat burglary, and extortion.

    Sincerely,
    Bob'’’)

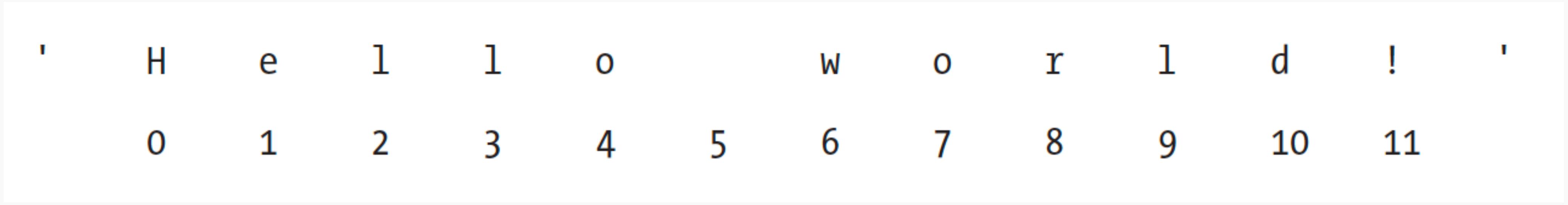
Same as:

  - *print('Dear Alice,\n\nEve\'s cat has been arrested for catnapping, cat burglary, and extortion.\n\nSincerely,\nBob')*

Indexing and slicing strings

- Strings use indexes and slices the same way lists do
  - *Spam = “Hello world”*

    spam[4] >>> *‘o’*
- You can think of the string 'Hello world!' as a list and each character in the string as an item with a corresponding index.
  - *spam[0:5]* >>> *’Hello’*



'	H	e	l	l	o		w	o	r	l	d	!	'
	0	1	2	3	4	5	6	7	8	9	10	11	

The in and not in Operators with Strings

- The **in** and **not in** operators can be used with strings just like with list values.
  - *'Hello' in 'Hello World'* >>> *True*
  - *'Hello' in 'Hello'* >>> *True*
  - *'HELLO' in 'Hello World'* >>> *False*
  - *'' in 'spam'* >>> *True*

Useful string methods

- *spam = 'Hello world!'*
- *spam = spam.**upper()*** *# these methods do not change the string itself*
- *spam = spam.**lower()*** *# but return new string values*
- *spam = spam.**capitalize()*** *# Capitalize first letter, ‘hello world’ to ‘Hello world’*
- *spam.**islower()*** *# False*
- *spam.**isupper()*** *# False, ‘HELLO WORLD!’.isupper() -> True*
- *'HELLO'.**lower().islower()*** *# True*
- ***isalpha()*** *# True if only letters and is not blank*
- ***isalnum()*** *# True if only letters and numbers and is not blank.*
- ***isdecimal()*** *# True if only numeric characters and is not blank.*
- ***isspace()*** *# True if only spaces, tabs, and new lines and is not blank.*
- ***istitle()*** *# True if only words that begin with an uppercase letter followed by*
  # only lowercase letters or space.

Useful string methods

- ***.startswith()***
- ***.endswith()***
- *', '**.join([**'cats', 'rats', 'bats’**])*** # 'cats, rats, bats'
- *' '.join(['My', 'name', 'is', 'Simon’])* # 'My name is Simon'
- *'ABC'.join(['My', 'name', 'is', 'Simon’])* # 'MyABCnameABCisABCSimon’
- *'My name is Simon'**.split()*** # ['My', 'name', 'is', 'Simon’]
- *'MyABCnameABCisABCSimon'.split('ABC’)* # ['My', 'name', 'is', 'Simon']
- *'My name is Simon'.split('m’)* # ['My na', 'e is Si', 'on']

Useful string methods

```
- 'Hello'.rjust(20)             # '               Hello'
- 'Hello World'.rjust(20)       # '         Hello World'
- 'Hello'.ljust(20)             # 'Hello               '
- 'Hello'.rjust(20, '*')        # '***************Hello'
- 'Hello'.center(20, '=‘)       # '=======Hello========'
```

Useful string methods

- *spam = '   Hello World   '*
- *spam.**strip**()   # 'Hello World'*
- *spam.**lstrip**()   # 'Hello World   '*
- *spam.**rstrip**()   # '   Hello World'*

- *spam = 'SpamSpamBaconSpamEggsSpamSpam'*
- *spam.strip('ampS')   # 'BaconSpamEggs'*
- *# strip('ampS') == strip('mapS') == strip('Spam').*
- *# Strip all occurrences of a, m, p, and S from the left*
- *# and right of the string. The order of the characters*
- *# does not matter!*

Useful string methods

- *name = ‘Bob’*
- *age = 20*
- *print("{0} has {1}!"****.format****(name, age)) # Bob has 20!*
- *print("{} has {}!"****.format****(name, age)) # Bob has 20!*
- *print("{1} has {0}!"****.format****(name, age)) # 20 has Bob!*

Read more about the format() method
https://www.w3schools.com/python/ref_string_format.asp

- Formats value(s) and insert them inside the string's placeholder
- Returns the formatted string
- Values and placeholders {}

pyperclip module

- The pyperclip module has copy() and paste() functions that can send text to and receive text from your computer’s clipboard.
- Sending the output of your program to the clipboard will make it easy to paste it to an email, word processor, or some other software.

```
– import pyperclip
– pyperclip.copy('Hello world!')
– pyperclip.paste()       # 'Hello world!'
```

Running programs – OS X and Linux

- Terminal

```
– cd                                  # Change Directory
– cd /Users/df/projects/etc  # Move into subfolders
– cd ~/projects/      # ‘~’ stands for your user account's home folder
– cd ..                               # Move one directory upwards
– pwd                                 # Print the Working Directory
– ls                                  # lists the file contents of a directory
```

- For Windows go to page 444 of the book

Handle command line arguments

Or in PyCharm, Run (on top) → Edit Configurations, then run the program

- *python3 pythonScript.py arg1 arg2 arg3*

pythonScript.py

```
#!/usr/bin/python

import sys

print('Number of arguments:', len(sys.argv), 'arguments.’ )
print('Argument List:', str(sys.argv))
```

-
  ```
  Number of arguments: 4 arguments.
  Argument List: ['test.py', 'arg1', 'arg2', 'arg3']
  ```

LAMBDA FUNCTIONS (NOT IN TEXTBOOK)

Lambda functions

- The **lambda keyword** in Python provides a shortcut for declaring small anonymous functions.
- Lambda functions behave just like regular functions declared with the def keyword.
- They can be used whenever function objects are required.

Lambda functions

```
def add(x, y):
    return x + y

print(add(5, 3))                    # 8
```

Lambda functions

```
def add(x, y):
    return x + y

print(add(5, 3))                    # 8

add = lambda x, y: x + y
print(add(5, 3))                    # 8
```

Lambda functions

Syntax:
lambda arguments : expression
- It can only contain expressions, not statements
- It is written as a single line of execution
- It can be immediately invoked

```
def add(x, y):
     return x + y

print(add(5, 3))                       # 8


add = lambda x, y: x + y
print(add(5, 3))                       # 8


print((lambda x, y: x + y)(5, 3))     # 8
```

Lambda functions

```python
(lambda x, y: x + y)(5, 3)    # 8
```

- *The difference is we didn't bind it to a name like* add *before we used it.*
- *We simply stated the expression we wanted to compute and then immediately evaluated it by calling it like a regular function*
- Unlike lambda forms in other languages, where they add functionality, Python lambdas are only a shorthand notation if you're too lazy to define a function (says the official Python documentation https://docs.python.org/3/faq/design.html#why-can-t-lambda-expressions-contain-statements)

Lambda example

Anonymous function inside another function

```
def makeAdder(n):
     return lambda x: x + n
```

a function that takes one arg (n), and that arg will be added with an unknown number (x)

```
plus3 = makeAdder(3)
plus5 = makeAdder(5)
```

make a function that always adds the number you send in with 3 or 5

```
print(plus3(4))          # 7
print(plus5(4))          # 9
```