MANIPULATING STRINGS AND LAMBDA FUNCTIONS

CS 3030: Python

Instructor: Damià Fuentes Escoté

University of Colorado Colorado Springs

Previous lesson - Dictionaries

- Dictionary vs list
- Add and remove key-value pairs
- Dictionary methods >>> `.keys(), .values(), .items(), .get(), setdefault(), .clear(), .pop(), .popitem()`
- in and not in dictionary
- Nested dictionaries
- Dicts comprehensions >>> `{x:chr(65+x) for x in range(1, 11)}`

TEXT IS ONE OF THE MOST COMMON FORMS OF DATA YOUR PROGRAMS WILL HANDLE

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.’)*

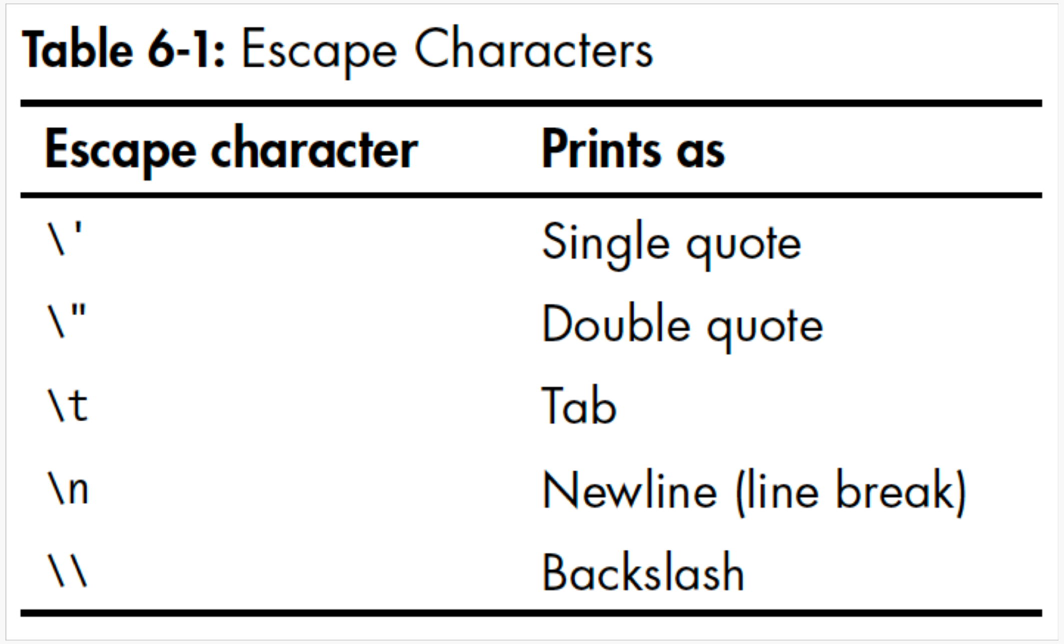


Table 6-1: Escape Characters

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

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

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

    Sincerely,
    Bob'”)

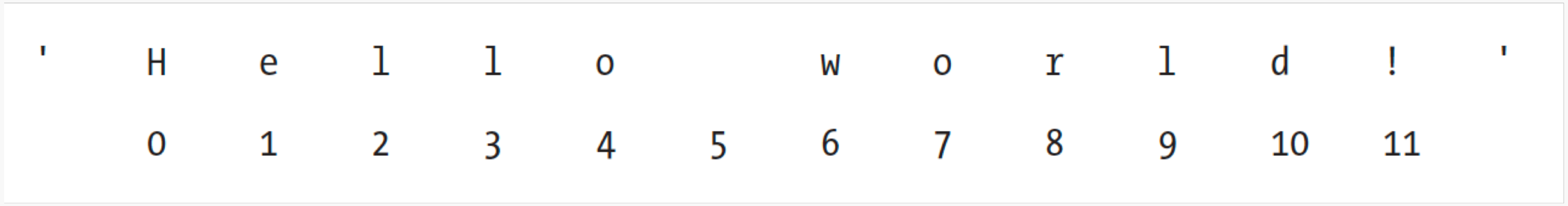
Same as:

- *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*

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

Running programs

- You don’t want to open PyCharm (or whatever editor) to run your programs!
- In order to tell your computer you want python to run your program your first line should be the **shebang**:

– *Windows*	>>>	*#! python3*
– *OS X*	>>>	*#! /usr/bin/env python3*
– *Linux*	>>>	*#! /usr/bin/python3*

Running programs – OS X and Linux

- Terminal
  - `python3 pythonScript.py    # Without the need of the shebang`

  - `chmod +x pythonScript.py  # To make it executable, need shebang`
  - `./pythonScript.py`

- For Windows go to page 444 of the book

Handle command line arguments

- `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']
  ```

Time to code – Password locker - HW3 Ex3

- You probably have accounts on many different websites.
- It’s a bad habit to use the same password for each of them because if any of those sites has a security breach, the hackers will learn the password to all of your other accounts.
- Develop a simple password manager software on your computer where:
- Write the account name (blog, facebook, instagram, etc) as an argument and the password is copied to the clipboard
  - *Then you can paste it into the website’s Password field.*

```
python3 ./pw.py instagram
# Password (example: htS:D`t*hQH3]9"C) copied to the clipboard
```

LAMBDA FUNCTIONS

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

```
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

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

- *The difference is we didn’t bind it to a name like* `add` *before I used it.*
- *We simply stated the expression I wanted to compute and then immediately evaluated it by calling it like a regular function*

Lambda example

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

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

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