UDACITY

University by Industry

Python Quick Review

- Variables and Assignment
- Arithmetic Operators
- Strings and String Methods
- Boolean Operators
- Loops
- Functions and Modules
- Data Structures
- File Operations

Variables and assignments

```
>>> riyadh_pop = 6506700

# reassignment
>>> riyadh_pop = riyadh_pop + 120

# reassignment operators
>>> riyadh_pop += 120
>>> riyadh_pop -= 10
>>> riyadh_pop *= 1.1
>>> riyadh_pop /= 2
```

Variables **names** can use *letters, digits*, and the *underscore*

Arithmetic operators

Operator	Functionality
+	Addition (String, tuple, list concatenation)
-	Subtraction (Set difference)
*	Multiplication (String, tuple, list multiplication)
/	Division
%	Modulus
**	Exponentiation
//	Integer division rounded towards minus infinity

Strings

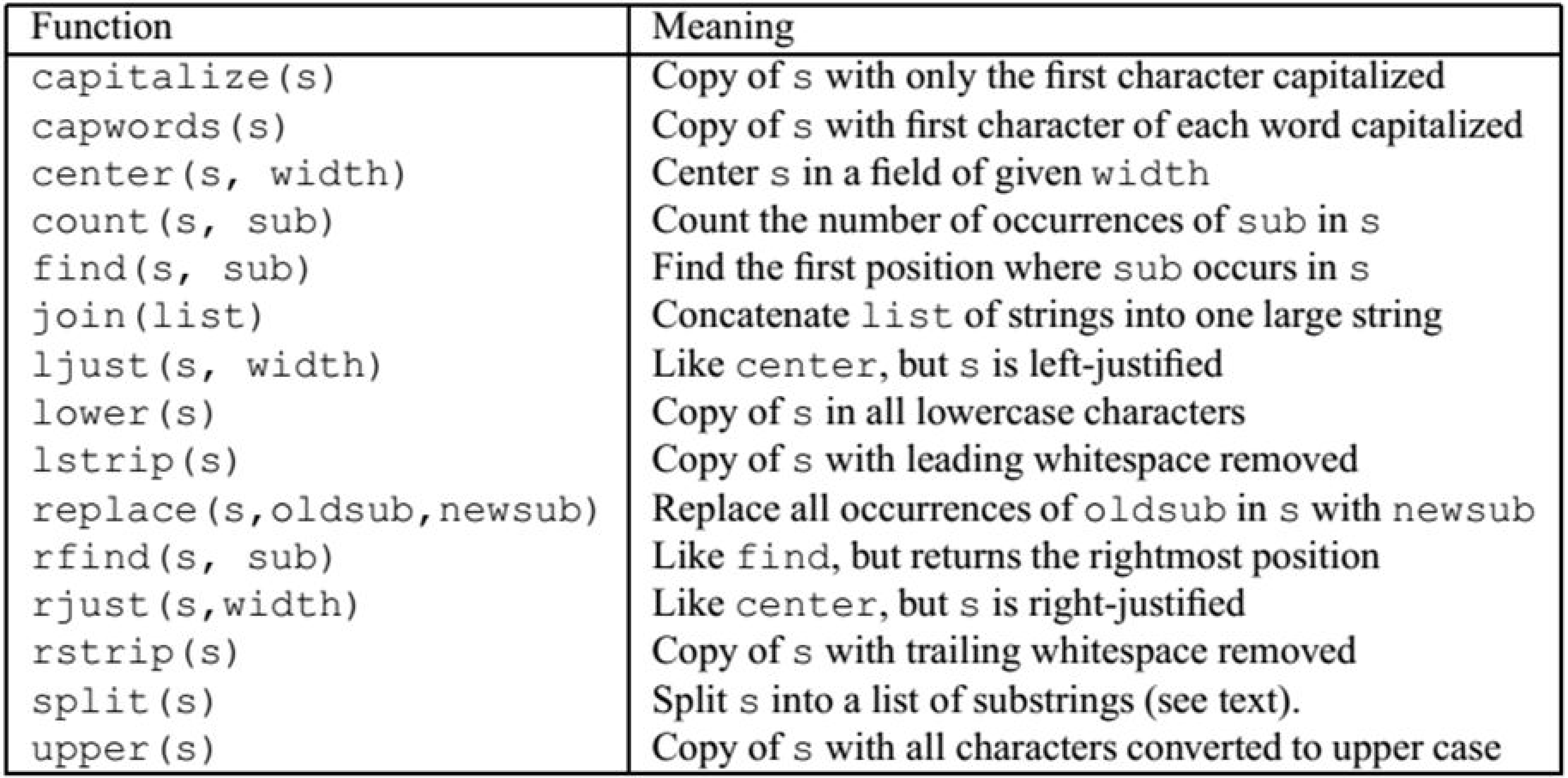
Examples:

```
'I am a string'
"me too"

>>> name = "Muhammed"
>>> 'Have a ' + "good day, " + name
Have a good day, Muhammed

>>> print len(name)
8
```

String methods



Function	Meaning
`capitalize(s)`	Copy of `s` with only the first character capitalized
`capwords(s)`	Copy of `s` with first character of each word capitalized
`center(s, width)`	Center `s` in a field of given `width`
`count(s, sub)`	Count the number of occurrences of `sub` in `s`
`find(s, sub)`	Find the first position where `sub` occurs in `s`
`join(list)`	Concatenate `list` of strings into one large string
`ljust(s, width)`	Like `center`, but `s` is left-justified
`lower(s)`	Copy of `s` in all lowercase characters
`lstrip(s)`	Copy of `s` with leading whitespace removed
`replace(s,oldsub,newsub)`	Replace all occurrences of `oldsub` in `s` with `newsub`
`rfind(s, sub)`	Like `find`, but returns the rightmost position
`rjust(s,width)`	Like `center`, but `s` is right-justified
`rstrip(s)`	Copy of `s` with trailing whitespace removed
`split(s)`	Split `s` into a list of substrings (see text).
`upper(s)`	Copy of `s` with all characters converted to upper case

String formatting

```
>>> name = "Muhammed"

>>> track = "DAND"

>>> welcome_message = "Welcome {}, to {}
connect session".format(name, track)

>>> welcome_message = "Welcome %s, to %s
connect session" %(name, track)

>>> print welcome_message

"Welcome Muhammed, to DAND connect session"
```

Booleans and comparisons

```
>>> the_sun_is_up = True
>>> the_sky_is_blue = False
```

Operator	**Functionality**
less than	<
greater than	>
less than or equal to	<=
greater than or equal to	>=
equal to	==
not equal to	!=

If, in python

```
if points <= 50:
    print "Congratulations, you won a yellow rubber duck!"
elif points <= 150:
    print "Congratulations, you won a wooden rabbit!"
else:
    print "Sorry, 150 is highest allowed number"
```

Notice the indentation highlighted in orange, this eliminates the need for curly brackets {}, it's python way of defining code blocks.

Boolean expressions

```
if is_raining and is_sunny:
    print "Is there a rainbow?"
```

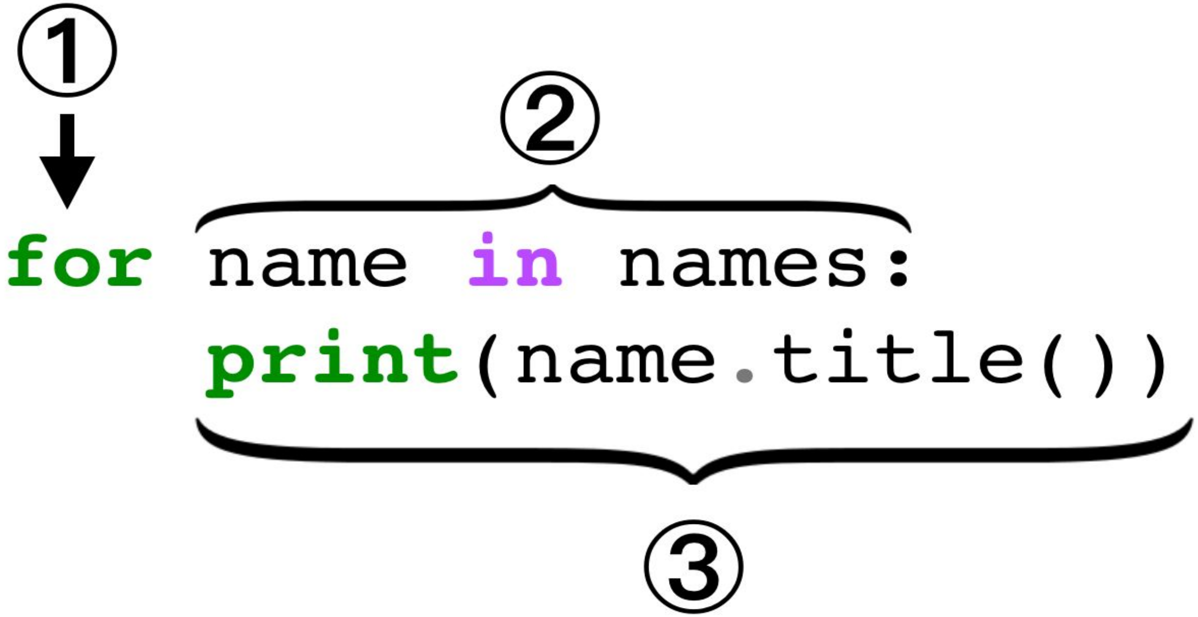
```
if location=="Riyadh" or location=="Khobar":
    send_email()
```

```
if name is not None:

    print("Welcome {}".format(name))
```

For loops

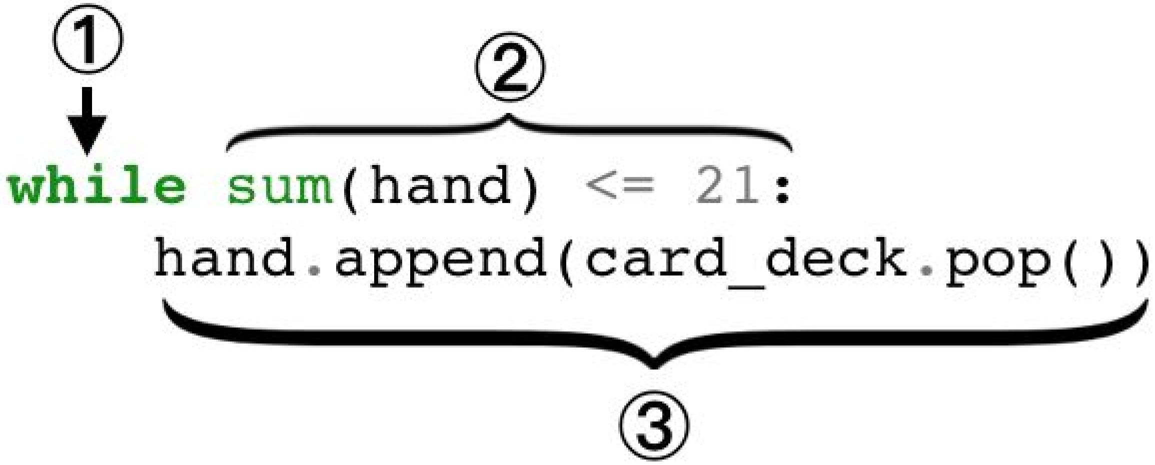
```
names = ['mohsen samir','salah hosny',
       'medhat morad', 'emad ehab']
```



```
for name in names:
    print(name.title())
```

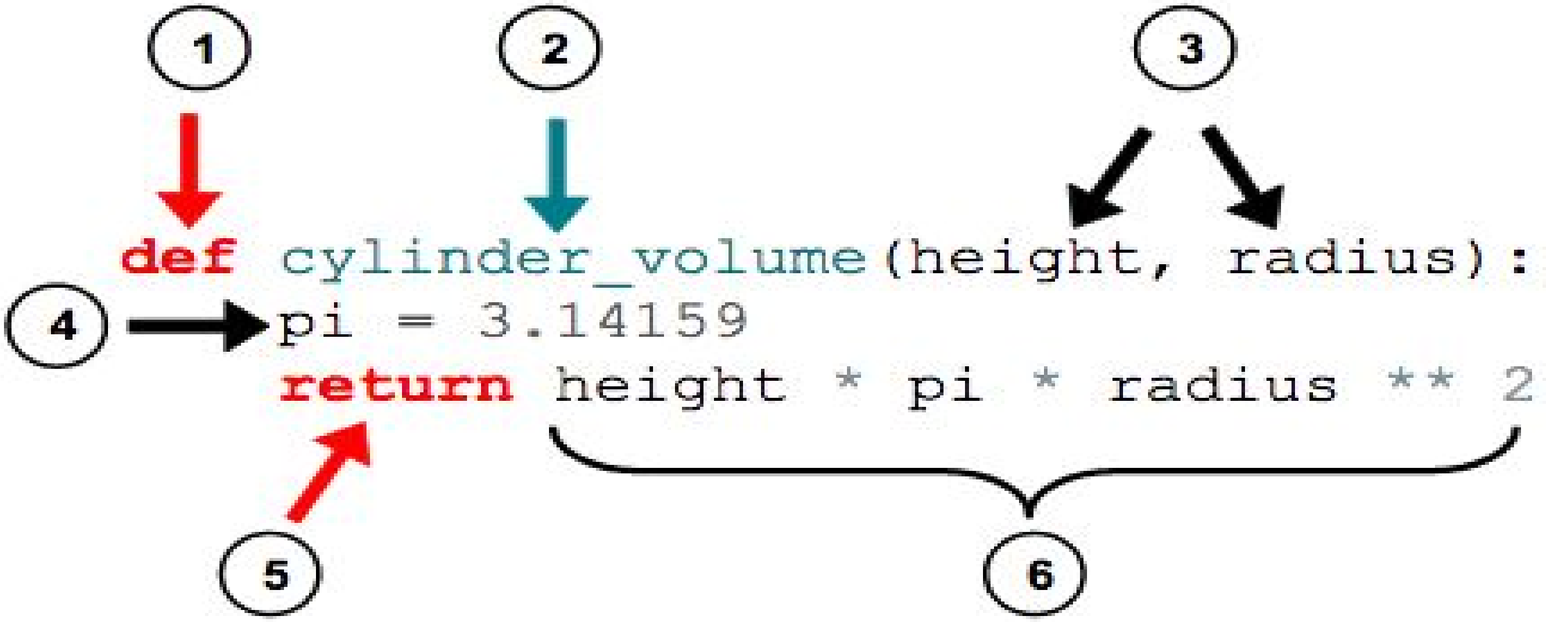
While loops

```
card_deck = [4, 11, 8, 5, 13, 2, 8, 10]
hand = []
```



```
while sum(hand) <= 21:
    hand.append(card_deck.pop())
```

Functions



```
def cylinder_volume(height, radius):
    pi = 3.14159
    return height * pi * radius ** 2
```

Default arguments

```
def todo_list(new_task, base_list=['wake up']):
    base_list.append(new_task)
    return base_list
```

We can call the function like this:

```
>>> todo_list("check the mail")
['wake up', 'check the mail']
```

Importing modules

```
>>> import math
>>> print math.factorial(3)
6
>>> from collections import defaultdict
>>> from collections import defaultdict, namedtuple
>>> import multiprocessing as mp
>>> from csv import reader as csvreader
```

One technique that should NOT be used

```
from random import *
```

Commonly used modules

csv: very convenient for reading and writing csv files

collections: useful extensions of the usual data types including OrderedDict, defaultdict and namedtuple

random: generates pseudo-random numbers, shuffles sequences randomly and chooses random items

string: more functions on strings. This module also contains useful collections of letters like string.digits (a string containing all characters with are valid digits).

re: pattern-matching in strings via regular expressions

math: some standard mathematical functions

os: interacting with operating systems

os.path: submodule of os for manipulating path names

sys: work directly with the Python interpreter

json: good for reading and writing json files (good for web work)

Lists

Lists are **mutable**, **indexed**, **ordered** container

Indexed from zero to length-1

```
a = []                        # the empty list
a = ['dog','cat','bird']      # simple list
a = [[1, 2],['a','b']]        # nested lists
a = [1,2,3] + [4,5,6]         # concatenation
a = [1,2,3] * 4               # replication
a = list(x)                   # conversion
```

```
if 1 in a:
    print a
```

List slicing

```
x = [0,1,2,3,4,5,6,7,8]
x[2]            # 3rd element - reference not slice
x[1:3]          # 2nd to 3rd element (1, 2)
x[:3]           # The first three elements (0,1,2)
x[-3:]          # last three elements
x[:-3]          # all but the last three elements
x[:]            # every element of x – copies x
x[1:-1]         # all but first and last element
x[::3]          # (0, 3, 6, 9, …) 1st then every 3rd
x[1:5:2]        # (1,3) start 1, stop >= 5, every 2nd
```

List methods

Method	**What it does**
l.append(x)	Add x to end of list
l.extend(other)	Append items from other
l.insert(pos, x)	Insert x at position
del l[pos]	Delete item at pos
l.remove(x)	Remove first occurrence of x; An error if no x
l.pop([pos])	Remove last item from list (or item from pos); An error if empty list
l.index(x)	Get index of first occurrence of x; An error if x not found
l.count(x)	Count the number of times x is found in the list
l.sort()	In place list sort
l.reverse(x)	In place list reversal

Tuples

Tuples useful when you have two or more values that are so closely related that they will always be used together, like latitude and longitude coordinates.

```
>>> dimensions = 52,40,100
>>> length,width,height = dimensions
>>> print "The dimensions are
{}x{}x{}".format(length, width, height)
The dimensions are 52x40x100
```

Sets

A Python set is an unordered, mutable collection of unique hashable objects.

a = set()	empty set
a = {'red', 'white', 'blue'}	simple set
a = set(x)	convert list

Dictionaries

Rather than storing single objects like lists and sets do, dictionaries store pairs of elements: **keys** and **values**

```
 elements = {'hydrogen': 1, 'helium': 2,
'carbon': 6}
```

```
 >>> print element['carbon']

 6
```

Iterating a dictionary

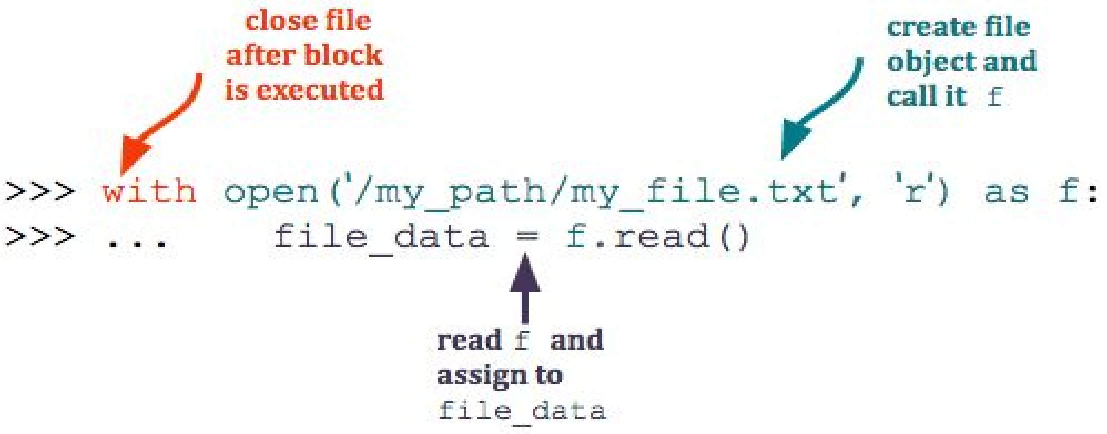
```
for key in dictionary:
    print key
for key, value in dictionary.items():
    print key, value
```

Dealing with files

```
f = open('/my_path/my_file.txt','w')
f.write("Hello World!")
f.close()
```

Cleaner more convenient way:

```
>>> with open('/my_path/my_file.txt','r') as f:
>>>    file_data = f.read()
```



close file after block is executed
create file object and call it f
read f and assign to file_data

```
>>> with open('/my_path/my_file.txt', 'r') as f:
>>> ...     file_data = f.read()
```

Additional Resources

Python Tutorial: TutorialsPoint

Learn Python (Interactive Guide)

Python Tutor (A tool for visualizing execution)