We are 183

L21: Monday – Week 13

Reminders!

- Wednesday – Maxim creates an iOS app during lecture
- Friday – Final Project Core Due

Last Time… on EECS 183

For Loops

Lists

Ranges

List Member Functions

Slicing

i>Clicker #1

```
print 'Hello',
print 'World'
```

What prints?

```
A) 'Hello'
   'World'
B) Hello
   World
C) HelloWorld
D) Hello World
```

i>Clicker #2

```
if score >= 60:
    print 'D',
elif score >= 70:
    print 'C',
elif score >= 80:
    print 'B',
elif score >= 90:
    print 'A',
else:
    print 'F',
```

What prints if `score` is 85?

A) B
B) D
C) B A
D) D C B A
E) None of the above

i>Clicker #3

```
s = str()
for i in range(2, 5):
    s += 'Z'

print s
```

What prints?

A) Z
B) ZZ
C) ZZZ
D) Nothing
E) This is not valid Python

i>Clicker #4

Given the list

```
numbers = [1, 2, 3, 4]
```

which of the following sets each element in `numbers` to 0?

```
A) numbers = 0
B) for n in numbers:
       n = 0
C) for i in len(numbers):
       numbers[i] = 0
D) for i in range(len(numbers)):
       numbers[i] = 0
```

E) More than one of the above

i>Clicker #5

```
numbers = [1, 2, 3, 4]
print numbers[-1], numbers[-3:-2]
```

What is printed?

A) `2 [4]`
B) `3 [1]`
C) `4 [2]`
D) `4 [2, 3]`
E) None of the above

Today

Tuples
String Split
While Loops
2D Lists
Dictionaries
User-Defined Functions
Classes

Python

tuples

tuple - An unmodifiable list

- A tuple is very similar to a list, but there is no way to modify it
  - Avoids some types of errors
  - More efficient
  - Many of the same functions as list
    - `.count()`, `.index()`, and concatenation

- Can access elements through bracket access

```
        tupleName[2]
```

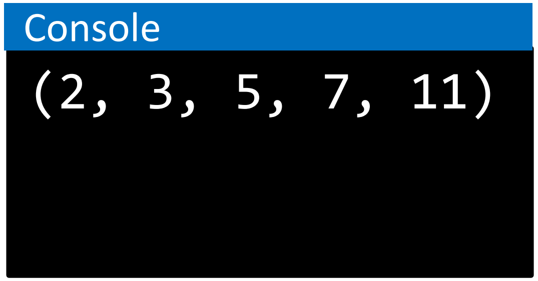
A simple tuple

- You can create a tuple by putting the values inside of parentheses, separated by commas

```
primes = (2, 3, 5, 7, 11)
```

- The print statement knows how to display a tuple for nice output

```
print primes
```

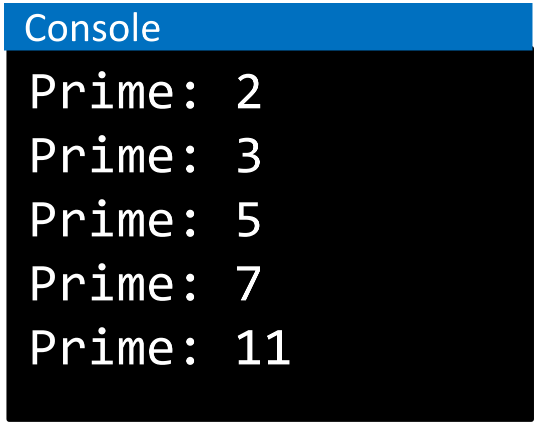


```
(2, 3, 5, 7, 11)
```

Looping over a tuple

- You could output each value on a separate line using a for loop:

```
primes = (2, 3, 5, 7, 11)

for value in primes:
    print 'Prime:', value
```



```
Prime: 2
Prime: 3
Prime: 5
Prime: 7
Prime: 11
```

Trying to change value

```
primes = (2, 3, 5, 7, 11)


primes[2] = 7
```

Console

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Tuples

- You can't add elements to a tuple.
- Tuples have no append or extend method
- You can't remove elements from a tuple.
- Tuples have no remove or pop method.

- You can find elements in a tuple, since this doesn't change the tuple.
- You can use the `in` operator to check if an element exists in the tuple.

Tuples

- Tuples are faster than lists.

Python

split()

String member functions

- A string object has almost 40 different member functions!
- See

  docs.python.org/2/library/stdtypes.html#string-methods

  - or Google "python library reference string functions"

- We can use `.split()` to turn a string into a list

Splitting a string into a list

- The `.split()` member function breaks up a string, based on a “separator” character
  - By default, the separator is a space
    - or any amount of whitespace
  - Can specify which character to use as a parameter

- Returns a list of strings when done

Examples of .split()

```
'a b c'.split()
['a', 'b', 'c']
```

Examples of .split()

```
'a b c'.split()
['a', 'b', 'c']

'a b        c'.split()
['a', 'b', 'c']
```

Examples of .split()

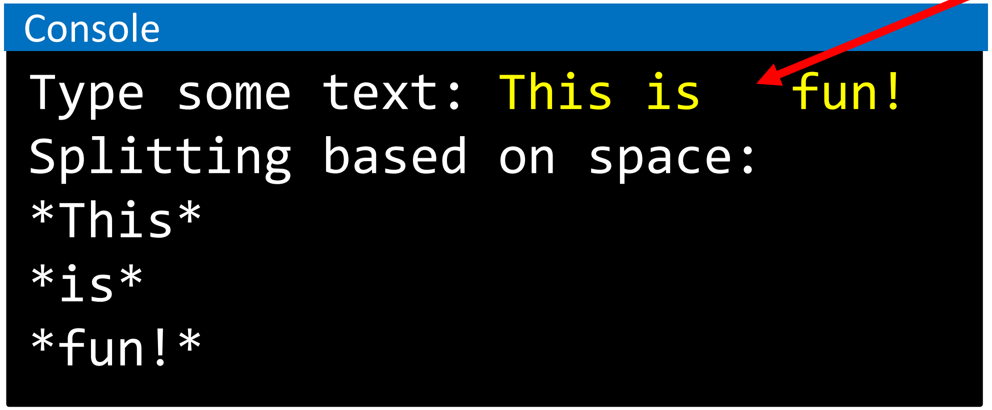
```
'a b c'.split()
['a', 'b', 'c']

'a b        c'.split()
['a', 'b', 'c']

'ab cd'.split()
['ab', 'cd']
```

Examples of .split()

```
'a b c'.split()
['a', 'b', 'c']

'a b        c'.split()
['a', 'b', 'c']

'ab cd'.split()
['ab', 'cd']

'a b, cd'.split(',')
['a b', ' cd']
```

Looping over the .split() result

```
text = raw_input('Type some text: ')
print 'Splitting based on space:'

for word in text.split():
    print '*' + word + '*'
```



```
Type some text: This is   fun!
Splitting based on space:
*This*
*is*
*fun!*
```

Repeated spaces only split once

Python

While Loops

Loop review

- Python has two looping structures:
  - `while` – Loop until a condition is met
  - `for` – Loop a certain number of times

The while Loop

- The while loop is very similar to C++ in both form and syntax

```
while condition:
    # Loop this content while condition == True
    print 'Condition is still True'

# This is outside the scope of the while
print 'Condition is not True'
```

Reading/summing numbers one number per line

```
sum = 0.0 # start with float
count = 0
print 'Enter a number (negative to quit):',
x = float(raw_input())

while x >= 0:
    sum += x
    count += 1 # sorry no ++ in Python
    print 'Enter a number (negative to quit):',
    x = float(raw_input())

print '\nRead', count, 'numbers, sum is:', sum
```

Reading/summing numbers With commas in line

- what if user wants to enter multiple values on a single line, separated by a comma

Reading/summing numbers
Multiple numbers per line

```
sum = 0.0
count = 0
prompt = 'Enter a number or numbers separated by commas\n'
prompt = prompt + '(just hit <Enter> to quit): '

line = raw_input(prompt)            # Input a line
while line != '':                   # <Enter> will exit
    line_splitup = line.split(',')  # Split the line
    for num in line_splitup:
        f = float(num)              # Convert to float
        sum += f
        count += 1

    line = raw_input(prompt)        # Input another line

# Output results
print '\nRead', count, 'numbers, sum is:', sum
```

Python

2D Lists

What about a “2D list”?

- That would be a **list of lists**
  - A list, each element of which is a list

- Remember that you cannot declare variables ahead of time
  1) Start with an empty list
  2) Append a list to it (put a list INSIDE of it, at the end)
  3) Repeat step (2) as needed

Student scores

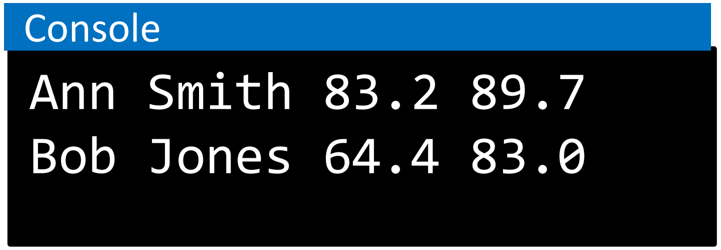
```
students = []
stu = ['Ann Smith', 83.2, 89.7]
students.append(stu)
stu = ['Bob Jones', 64.4, 83.0]
students.append(stu)
print students
```

Console

```
[['Ann Smith', 83.2, 89.7], ['Bob Jones', 64.4, 83.0]]
```

Printing without [ ]

```
# Assume "students" list from
# previous slide

for stu in students:
    for col in stu:
        print col,
    print # move to next line
```



Console

```
Ann Smith 83.2 89.7
Bob Jones 64.4 83.0
```

Printing without [ ]

```python
# Assume "students" list from
# previous slide

for stu in students:
    for col in stu:
        print col,
    print # move to next line
```

Note the importance of indents!

Console

```
Ann Smith 83.2 89.7
Bob Jones 64.4 83.0
```

Python

Dictionaries

Dictionaries: Used in Final Projects

- Like arrays/lists, but indexed by any object
- Pairs of "key" and "value"

```
>>> dct = {}
```

Dictionaries: Used in Final Projects

- Like arrays/lists, but indexed by any object
- Pairs of "key" and "value"

```
>>> dct = {}
```

Define an empty dictionary with {}

Dictionaries: Used in Final Projects

- Like arrays/lists, but indexed by any object
- Pairs of "key" and "value"

```
>>> dct = dict()
```

Define an empty dictionary with constructor

Dictionaries: Used in Final Projects

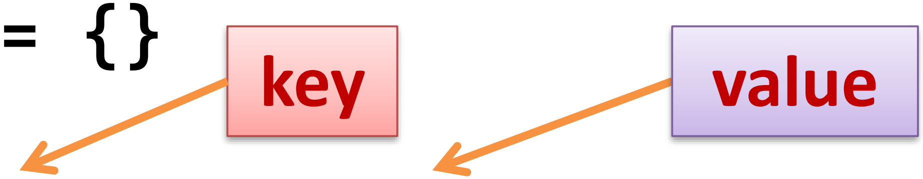
- Like arrays/lists, but indexed by any object
- Pairs of "key" and "value"

```
>>> dct = {}

>>> dct[0] = 'hello'
```

Dictionaries: Used in Final Projects

- Like arrays/lists, but indexed by any object
- Pairs of "key" and "value"

```
>>> dct = {}

>>> dct[0] = 'hello'
```



key

value

Dictionaries: Used in Final Projects

- Like arrays/lists, but indexed by any object
- Pairs of "key" and "value"

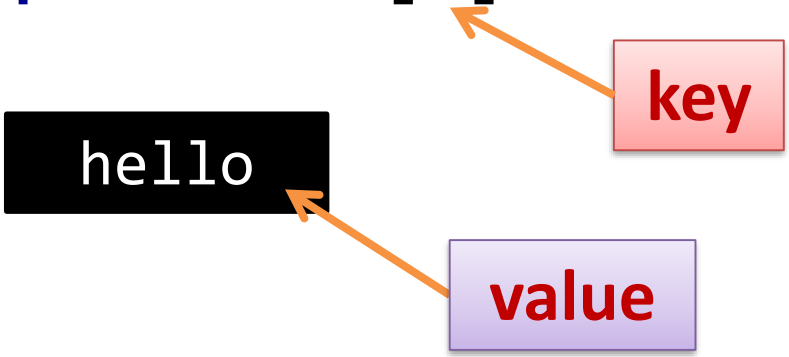
```
>>> dct = {}

>>> dct[0] = 'hello'
>>> print dct[0]
```

```
hello
```

Dictionaries: Used in Final Projects

- Like arrays/lists, but indexed by any object
- Pairs of "key" and "value"

```
>>> dct = {}

>>> dct[0] = 'hello'
>>> print dct[0]
```



```
hello
```

Dictionaries: Used in Final Projects

- Like arrays/lists, but indexed by any object
- Pairs of "key" and "value"

```
>>> dct = {}

>>> dct[0] = 'hello'
>>> print dct[0]
    hello

>>> dct['word'] = 'world'
>>> print dct['word']
    world
```

Strings can be used as keys!!!

Dictionaries: Used in Final Projects

- Like arrays/lists, but indexed by any object
- Pairs of "key" and "value"

```
>>> dct = {}

>>> dc
>>> pr

>>> dc
>>> pr
```

```
# for debugging:
dct = dict()
print dct    #{}
dct[0] = 'hello'
print dct  # {0: 'hello'}
dct['word'] = 'world'
print dct  # {0: 'hello', 'word': 'world'}
```

Dictionaries: Used in Final Projects

- Can also define dictionaries at creation
- Use **key: value** in definition

```
>>> dct = { 1: 'hello', 3.14: 'pi',
            'num_students': 20 }

>>> print dct[1]

       hello

>>> print dct['num_students']

        20

>>> print dct[3.14]

        pi
```

Dictionaries

- Often used to store heterogeneous data in a meaningful way:

```
student = {}
student['firstName'] = 'Meghana'
student['lastName'] = 'Shankar'
student['grades'] = [ 90, 95, 92 ]

print 'Student:', student['firstName'],
print student['lastName']
```

```
Student: Meghana Shankar
```

i>Clicker #6

```
student = {}
student['firstName'] = 'Meghana'
student['lastName'] = 'Shankar'
student['grades'] = [ 90, 95, 92 ]

print 'Grades:',
for grade in student['grades']:
    print grade,
```

A)

```
Grades: [90, 95, 92]
```

B)

```
Grades: 90 95 92
```

C)

```
Grades:
[90, 95, 92]
```

D)

```
Grades:
90 95
```

E)

```
Grades:
90
95
92
```

Iterating Over Dictionary `.items()`

```
grades = {}
grades['proj1'] = 60
grades['proj2'] = 70
grades['proj3'] = 80
grades['exam1'] = 90
grades['exam2'] = 100

for k, v in grades.items():
    print k + ':', v
```

```
exam2: 100
proj3: 80
exam1: 90
proj2: 70
proj1: 60
```

Note that dictionaries are NOT reliably ordered.

k will be assigned each key
v will be assigned each value

`.items()` returns a list of (key, value) tuple pairs

Iterating Over Dictionary `.items()`

```
grades = {}
grades['proj1'] = 60
grades['proj2'] = 70
grades['proj3'] = 80
grades['exam1'] = 90
grades['exam2'] = 100

for k, v in grades.items():
    print k + ':', v
```

Note that dictionaries are NOT reliably ordered.

```
exam2: 100
proj3: 80
exam1: 90
proj2: 70
proj1: 60
```

```
proj3: 80
exam2: 100
exam1: 90
proj2: 70
proj1: 60
```

Nested Dictionaries

```
student = {}
student['firstName'] = 'Meghana'
student['lastName'] = 'Shankar'
student['grades'] = {}
student['grades']['proj1'] = 90
student['grades']['proj2'] = 95
student['grades']['exam1'] = 92
```

Nested Dictionaries

```
student = {}
student['firstName'] = 'Meghana'
student['lastName'] = 'Shankar'
student['grades'] = {}
student['grades']['proj1'] = 90
student['grades']['proj2'] = 95
student['grades']['exam1'] = 92

for k, v in student['grades'].items():
    print k + ':', v
```

```
exam1: 92
proj2: 95
proj1: 90
```

Note that dictionaries are STILL NOT reliably ordered.

Python

User-Defined Functions

Functions in Python

- Example: Square function

```
def square(a):
    sq = a * a
    return sq
```

Functions in Python

- Example: Square function

def tells Python that we
are defining a function
no return type specified

```
def square(a):
    sq = a * a
    return sq
```

Functions in Python

- Example: Square function

Function name

```python
def square(a):
    sq = a * a
    return sq
```

Functions in Python

- Example: Square function

Input parameter

```
def square(a):
    sq = a * a
    return sq
```

Functions in Python

- Example: Square function

Colon marks the end of the definition

```python
def square(a):
    sq = a * a
    return sq
```

Functions in Python

- Example: Square function

```
def square(a):
    sq = a * a
    return sq
```

Return value

Functions in Python

- Example: Square function

```
def square(a):
    sq = a * a
    return sq
```

Indenting is
CRITICAL
4 spaces is standard

User Defined Function

```
def square(a):
    sq = a * a
    return sq
```

```
# main program
x = 3
y = square(x)
```

function call

Function Call

Execution

```
def square(a):
    sq = a * a
    return sq



# main program
x = 3
y = square(x)
```

Execution starts at the top of the file

Function Call

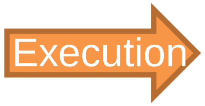
Execution

```
def square(a):
    sq = a * a
    return sq




# main program
x = 3
y = square(x)
```

Function definitions are executable statements

User Defined Function

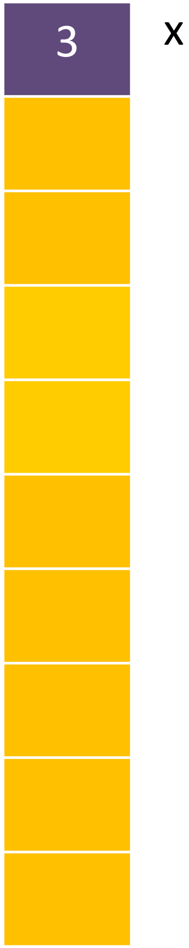
```
def square(a):
    sq = a * a
    return sq



# main program
x = 3
y = square(x)
```



Function Call

```
def square(a):
    sq = a * a
    return sq



# main program
x = 3
y = square(x)
```

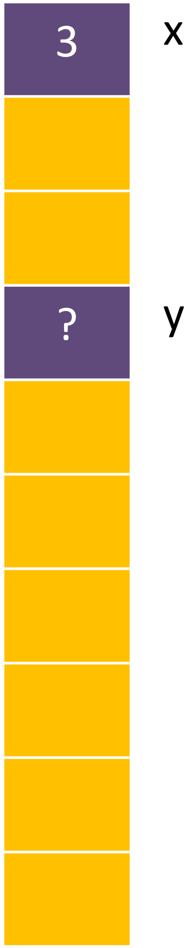
Execution



Function Call

```
def square(a):
    sq = a * a
    return sq




# main program
x = 3
y = square(x)
```

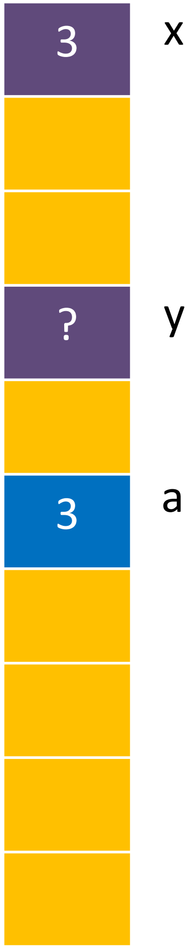
Execution



Function Call

Execution

```
def square(a):
    sq = a * a
    return sq


# main program
x = 3
y = square(x)
```

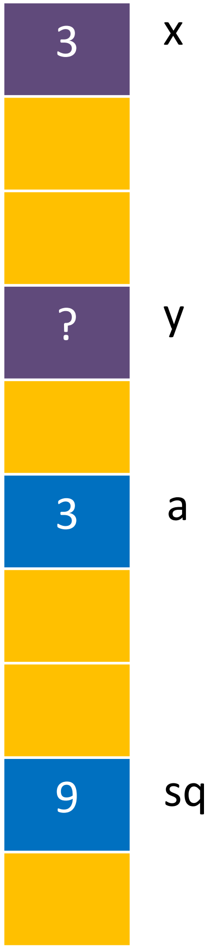


Function Call

Execution

```
def square(a):
    sq = a * a
    return sq
```

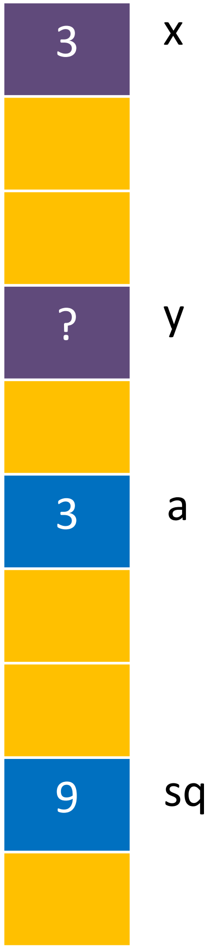
```
# main program
x = 3
y = square(x)
```



Function Call

Execution

```
def square(a):
    sq = a * a
    return sq


# main program
x = 3
y = square(x)
```

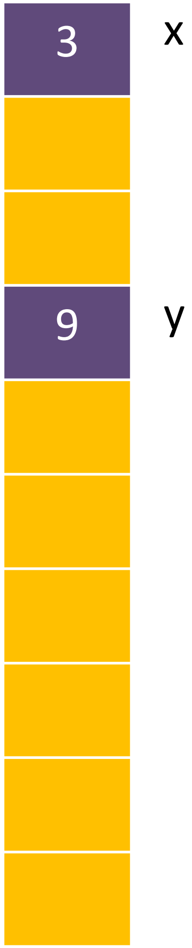


Function Call

```
def square(a):
    sq = a * a
    return sq


# main program
x = 3
y = square(x)
```

Execution

9



The return statement

- Almost exactly the same as in C++
  - Except no explicit return type
- A `return` followed by nothing exits the function but returns no value
- Can return a value if desired
- If return is not present, execution ends when the end of the function is reached
  - **Indent level going back to 0 ends the function definition**

Remember the returned value

- If you don’t use it, store it, or print it, the result of calling the function is wasted time, but nothing else

```
square(4)  # not printed
           # not saved
           # not used
```

Legal in Python just like in C++

i>Clicker #7

```
def increment(x):
    return x + 1

# main program
x = 3
print (increment(x / 2) +
       increment(x + 1))
```

What prints?

A) 5
B) 5.5
C) 6
D) 6.5
E) 7

Documenting Functions

- After the function header, include a block quote (using ' ' '), indented
- The stuff inside the quote block the function’s documentation

- Can retrieve a function’s documentation via print; try this:

```
>>> print abs.__doc__
abs(number) -> number

Return the absolute value of the argument.
```

double underscores

Documented function

- Example: Square function

```
def square(a):
    '''
    Returns the square of a number
    '''
    sq = a * a
    return sq
```

Documented function

- Example: Square function

```
def square(a):
    '''
    Returns the square of a number
    '''
    sq = a * a
    return sq

print square.__doc__
```

Console

```
    Returns the square of a number
```

Setting up a main function

```
import sys

def square(a):
    '''
    Returns the square of a number
    '''
    return a * a

def main():
    x = 3
    y = square(x)
    print 'y is:', y

if __name__ == '__main__':
    main()
```

Python

Classes

Python Classes

- Work similarly to C++ classes
  - Instances can be created
  - Can hold data
  - Can hold functions

Defining a Python Class

- A not-very-useful Student class:

```
class Student:
    firstName = 'Grace'
    lastName = 'Kendall'

grace = Student()

print grace.firstName, grace.lastName
```

Syntax for creating an instance

Same "dot" syntax for members as C++

```
Grace Kendall
```

Python Class Constructor: __init__

```
class Student:
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

grace = Student('Grace', 'Kendall')

print grace.firstName, grace.lastName
```

Constructor

Call constructor with 2 arguments

```
Grace Kendall
```

Python Class Constructor: __init__

```
class Student:
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

grace = Student('Grace', 'Kendall')

print grace.firstName, grace.lastName
```

Instance is parameter rather than implicit

```
Grace Kendall
```

Python Class Constructor: __init__

```
class Student:
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

grace = Student('Grace', 'Kendall')

print grace.firstName, grace.lastName
```

Members not automatically in scope

```
Grace Kendall
```

Python Class Constructor: __init__

```
class Student:
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

grace = Student('Grace', 'Kendall')

print grace.firstName, grace.lastName
```

Members are always public

```
Grace Kendall
```

Everything in Python classes is public

```python
class Student:
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

grace = Student('Grace', 'Kendall')
grace.firstName = 'Pranav'

print grace.firstName, grace.lastName
```

```
Pranav Kendall
```

Python Class Member Functions

```
class Student:
    def __init__(self, fisrt, last):
        self.firstName = fisrt
        self.lastName = last
        self.grades = []

    def addGrade(self, grade):
        self.grades.append(grade);

    def averageGrades(self):
        if (len(self.grades) == 0):
            return 0
        return sum(self.grades) / len(self.grades)
```

Member functions

```python
class Student:
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last
        self.grades = []

    def addGrade(self, grade):
        self.grades.append(grade);

    def averageGrades(self):
        if (len(self.grades) == 0):
            return 0
        return sum(self.grades) / len(self.grades)


grace = Student('Grace', 'Kendall')
grace.addGrade(80)
grace.addGrade(90)
print grace.name + "'s average:", grace.averageGrades()
```

Instance is passed as first argument to member function

```
Grace's average: 85
```