PL0a: Tutorial de Unix / Python / Autograder

Tabla de contenido

Introducción

Los proyectos de esta clase se realizarán usando Python 3.6+.
El proyecto 0 cubrirá lo siguiente:

- Un tutorial de mini-UNIX (particularmente importante si se trabaja desde el terminal.
- Instrucciones sobre cómo configurar la versión correcta de Python.
- Un tutorial de mini-Python.
- Calificación del proyecto: la versión de cada proyecto incluye su autocalificador para que lo ejecute usted mismo.

Archivos para editar: completará partes de los ficheros **addition.py**, **buyLotsOfFruit.py** y **shopSmart.py** durante la tarea.

Autoevaluación: su código se calificará automáticamente para verificar su correcto funcionamiento. Por favor, ***no*** cambie los nombres de las funciones o clase proporcionada dentro del código, o causará problemas en el autograder.

Conceptos básicos de Unix

A continuación, se muestran los comandos básicos de UNIX para navegar y editar archivos.

Manipulación de archivos / directorios
Cuando abre una ventana de terminal, se le coloca en un símbolo del sistema:

[IA@lab ~]$

El mensaje muestra su nombre de usuario, el host en el que está conectado y su ubicación actual en la estructura de directorio (su ruta). El carácter **~** es una abreviatura de su directorio de inicio. Para crear un directorio, use el comando **mkdir**. Use **cd** para cambiar a ese directorio:

[IA@lab ~]$ mkdir foo
[IA@lab ~]$ cd foo
[IA@lab ~/foo]$

Utilice **ls** para ver una lista del contenido de un directorio y **touch** para crear un archivo vacío:

[IA@lab ~/foo]$ ls
[IA@lab ~/foo]$ touch hello_world
[IA@lab ~/foo]$ ls
Hola Mundo
[IA@lab ~/foo]$ cd ..
[IA@lab ~]$

Descargue *python_basics.zip* (nota: el nombre del archivo zip puede ser ligeramente diferente). Use descomprimir para extraer el contenido del archivo zip:

[IA@lab ~]$ ls * .zip
python_basics.zip
[IA@lab ~]$ unzip python_basics.zip
[IA@lab ~]$ cd python_basics
[IA@lab ~/python_basics]$ ls
foreach.py
helloWorld.py
listcomp.py
listcomp2.py
quickSort.py
shop.py
shopTest.py

Algunos otros comandos útiles de Unix:

- **cp** copia un archivo o archivos
- **rm** elimina un archivo
- **mv** mueve un archivo (es decir, cortar / pegar en lugar de copiar / pegar)
- **man** muestra la documentación de un comando
- **pwd** muestra toda la ruta actual
- **xterm** abre una nueva ventana de terminal
- **firefox** abre un navegador web
- Presione "**Ctrl-z**" para mandar un proceso a segundo plano ejecución
- Agregue **&** a un comando para ejecutarlo en segundo plano
- **fg** trae un programa que se está ejecuta en segundo plano a primer plano
- Presione "**Ctrl-c**" para matar un proceso en ejecución

El editor de texto de Emacs

Emacs es un editor de texto personalizable que tiene algunas características diseñadas específicamente para programadores. Sin embargo, puede usar cualquier otro editor de texto que prefiera (vi, pico, gedit, notepad++, Visual Studio Code … en Unix; o Bloc de notas, notepad++, Visual Studio Code en Windows; o TextWrangler en OS X; y muchos mas).

Para ejecutar Emacs, escriba emacs en la terminal:

```
[IA@lab ~/python_basics]$ emacs helloWorld.py &
[1] 3262
```

De esa forma, se abrirá el fichero helloWorld.py para ser editado, si no existe, se crea el fichero. Emacs se da cuenta que este es un archivo fuente de Python (debido a la terminación .py) y entra en el modo Python que te ayudará a escribir código. Al editar este archivo, puede notar parte de ese texto se colorea automáticamente: este es el resaltado de sintaxis para ayudarlo a distinguir elementos como palabras clave, variables, cadenas y comentarios. Si presiona Intro, Tabulador o Retroceso puede causar el cursor para saltar a ubicaciones extrañas: esto se debe a que Python es muy exigente con la sangría, y Emacs está prediciendo la tabulación adecuada que debe utilizar. Algunos comandos básicos de edición de Emacs (C- significa "mientras se mantiene presionada la tecla Ctrl"):

- **C-x C-s** Guarda el archivo actual
- **C-x C-f** Abre un archivo o crea uno nuevo si no existe
- **C-k** Corta una línea, agrégala al portapapeles
- **C-y** Pega el contenido del portapapeles
- **C-_** Deshacer
- **C-g** Abortar un comando medio ingresado

También puede copiar y pegar usando el ratón. Con el botón izquierdo, seleccione la región de texto para copiar. Haga clic en el botón central para pegar.

Hay dos formas de usar Emacs para desarrollar código Python. La forma más sencilla es utilizar simplemente como un editor de texto: cree y edite archivos Python en Emacs; luego ejecute Python para probar el código en una ventana de terminal. Alternativamente, puede ejecutar Python dentro de Emacs: vea las opciones en "Python" en la barra de menú, o escriba **C-c !** para iniciar un intérprete de Python en una pantalla dividida. (Utilice **C-x o** para cambiar entre las pantallas divididas, o simplemente hacer clic si **C-x** no funciona).

Si desea dedicar un tiempo de configuración adicional a convertirse en un usuario avanzado, puede probar un IDE como *Eclipse* (descargue el paquete Eclipse y consulte *PyDev* para el soporte de Python en Eclipse).

Conceptos básicos de Python

Archivos requeridos
Puede descargar todos los archivos asociados con el mini-tutorial de Python como un archivo zip: *python_basics.zip*. Si hizo el tutorial de Unix, ya habrá descargado y descomprimido este archivo.

Tabla de contenido

Invocar al intérprete
Python se puede ejecutar en dos modos. Se puede utilizar de forma *interactiva*, a través de un intérprete, o se puede llamar desde la línea de comandos para ejecutar un *script*. Primero usaremos el intérprete de Python interactivamente.
Invoca al intérprete ingresando python en el símbolo del sistema de Unix.

```
(IA)[IA@lab ~]$ python
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May 3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Operadores

El intérprete de Python se puede utilizar para evaluar expresiones, por ejemplo:

```
>>> 1 + 1
2
>>> 2 * 3
6
```

Los operadores booleanos también existen en Python para manipular los valores primitivos True y False.

```
>>> 1 == 0
False
>>> no (1 == 0)
True
>>> (2 == 2) and (2 == 3)
False
>>> (2 == 2) or (2 == 3)
True
```

String

Como Java, Python tiene integrado un tipo de cadena. El operador + está sobrecargado para hacer concatenación de cadenas.

```
>>> "inteligencia" + 'artificial'
"inteligenciaartificial”
```

Hay muchos métodos que le permiten manipular cadenas.

```
>>> 'artificial'.upper()
'ARTIFICIAL'

>>> 'AYUDA'.lower ()
'ayuda'

>>> len('Ayuda')
5
```

Observe que podemos usar comillas simples ' ' o comillas dobles " " para las cadenas. Esto permite anidar fácilmente las cadenas.

También podemos almacenar expresiones en variables.

```
>>> s = 'hola mundo'
>>> print(s)
hola mundo
>>> s.upper()
'HOLA MUNDO'
>>> len(s.upper())
10
>>> num = 8.0
>>> num + = 2.5
```

```
>>> print (num)
10.5
```

En Python, no tienes que declarar variables antes de asignarlas.

Ejercicio: Dir y help

Obtenga más información sobre los métodos que Python proporciona para las cadenas. Para ver qué métodos proporciona Python para cada tipo de datos, use los comandos dir y help:

```
>>> s = 'abc'
>>> dir(s)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__','__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__','__repr__', '__rmod__', '__rmul__', '__setattr__', '__str__', 'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'replace', 'rfind', 'rindex', 'rjust', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

>>> help (s.find)
```

```
Help on built-in function find:

find(...) method of builtins.str instance
    S.find(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
```

```
>> s.find ('b')
1
```

Pruebe algunas de las funciones para las cadenas que se enumeran con `dir` (ignore aquellas con guiones bajos `'_'` alrededor del nombre del método, son métodos privados).

Estructuras de datos integradas

Python viene equipado con algunas estructuras de datos, muy similares a las de Java.

Lists [ ]

Las listas almacenan una secuencia de elementos modificables:

```
>>> frutas = ['manzana', 'naranja', 'pera', 'platano']
>>> frutas [0]
'manzana'
```

Podemos usar el operador + para hacer la concatenación de listas:

```
>>> otrasFrutas = ['kiwi', 'fresa']
>>> frutas + otrasFrutas
>>> ['manzana', 'naranja', 'pera', 'platano', 'kiwi', 'fresa']
```

Python también permite la indexación negativa desde la parte posterior de la lista. Por ejemplo, frutas [-1] accederíamos al último elemento, 'platano':

```
>>> frutas [-2]
'pera'
>>> frutas.pop() ‘por defecto devuelve el ultimo elemento y lo elimina de la lista’
'platano'
>>> frutas.pop(2) ‘devuelve el elemento que se encuentra en la posición 2 y lo elimina de la lista’
'pera'

>>> frutas
['manzana', 'naranja']
>>> frutas.append(‘pera’)
>>> frutas
['manzana', 'naranja', 'pera']
>>> frutas.append(‘pomelo’)
>>> frutas
['manzana', 'naranja', 'pera', 'pomelo']
>>> frutas [-1] = 'pina'
>>> frutas
['manzana', 'naranja', 'pera', 'pina']
```

También podemos indexar múltiples elementos adyacentes usando el operador de corte. Por ejemplo, frutas[1:3], devuelve una lista que contiene los elementos en la posición 1 y 2. En general frutas[inicio:fin] obtendrá los elementos en *inicio*, *inicio + 1*, ..., *fin-1*. Podemos también hacer frutas[inicio:] que devuelve todos los elementos a partir del índice de *inicio*, frutas[:fin] devolverá todos los elementos antes del elemento de la posición *final*:

```
>>> frutas [0:2]
['manzana’, ‘naranja']
>>> frutas [:3]
['manzana', 'naranja', 'pera']
>>> frutas [2:]
['pera', 'pina']
>>> len(frutas)
4
```

Los elementos almacenados en listas pueden ser de cualquier tipo de datos de Python. Por ejemplo, podemos tener listas de listas:

```
>>> lstOfLsts = [['a', 'b', 'c'], [1, 2, 3], ['uno', 'dos', 'tres']]
>>> lstOfLsts[1][2]
```

```
3
>>> lstOfLsts[0].pop()
'c'
>>> lstOfLsts
[['a', 'b'], [1, 2, 3], ['uno', 'dos', 'tres']]
```

Ejercicio: listas

Juega con algunas funciones de listas.

```
>>> dir (list)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
'__delslice__', '__doc__', '__eq__', '__ge__', '__getattribute__',
'__getitem__', '__getslice__', '__gt__', '__hash__', '__iadd__', '__imul__',
'__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__',
'__rmul__', '__setattr__', '__setitem__', '__setslice__', '__str__',
'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse',
'sort']
```

```
>>> help(list.reverse)
Help on built-in function reverse:

reverse(...)
    L.reverse() -- reverse *IN PLACE*
```

```
>>> lst = ['a', 'b', 'c']
>>> lst.reverse ()
>>> ['c', 'b', 'a']
```

Nota: Ignore las funciones con guiones bajos "_" alrededor de los nombres; estos son métodos privados.

Presione 'q' para salir de una pantalla de *help*.

Tuplas ( )

Una estructura de datos similar a la lista es la *tupla*, que es como una lista pero no se puede modificar los valores una vez creadas. Tenga en cuenta que las tuplas están rodeadas de paréntesis, mientras que las listas tienen corchetes.

```
>>> par = (3, 5)
>>> par[0]
3
>>> x,y = par
>>> x
3
>>> y
5
>>> par[1] = 6
```

```
TypeError: 'tuple' object does not support item assignment.
```

El intento de modificar una tupla genera una excepción. Las excepciones indican errores:
indexar fuera de límites, errores de tipo, etc., todos reportarán excepciones de esta manera.

Conjuntos { }
Un *conjunto* es otra estructura de datos que sirve como una lista desordenada sin elementos duplicados.

```
>>> formas = ['circulo', 'cuadrado', 'triangulo', 'circulo']
>>> formas
['circulo', 'cuadrado', 'triangulo', 'circulo']
>>> setOfShapes = set(formas)
>>> setOfShapes
{'cuadrado', 'triangulo', 'circulo'}
```

A continuación, se muestra otra forma de crear un conjunto:

```
>>> setOfShapes = {'circulo', 'cuadrado', 'triangulo', 'circulo'}
```

A continuación, se muestra cómo agregar cosas al conjunto, probar si un elemento está en el conjunto y realizar un conjunto de operaciones (diferencia, intersección, unión):

```
>>> setOfShapes
{'cuadrado', 'triangulo', 'circulo'}
>>> setOfShapes.add ('poligono')
>>> setOfShapes
{'cuadrado', 'triangulo', 'circulo', 'poligono'}
>>> 'circulo' in setOfShapes
True
>>> 'rombo' en setOfShapes
False
>>> favoriteShapes = ['circulo', 'triangulo', 'hexagono']
>>> setOfFavoriteShapes = set(favoriteShapes)
>>> setOfShapes - setOfFavoriteShapes
{'cuadrado', 'poligono'}
>>> setOfShapes & setOfFavoriteShapes
{'circulo', 'triangulo'}
>>> setOfShapes | setOfFavoriteShapes
{'circulo', 'cuadrado', 'triangulo', 'poligono', 'hexagono'}
```

Tenga en cuenta que los objetos del conjunto están desordenados; no puede asumir que su recorrido o impresión será el mismo en todas las máquinas.

Diccionarios { }
La última estructura de datos es el *diccionario* que almacena un mapa de un tipo de objeto (la clave o key) a otro (el valor). La clave/key debe ser de un tipo inmutable (cadena, número o tupla). El valor puede ser cualquier tipo de datos de Python.

Nota: En el siguiente ejemplo, el orden impreso de las claves devueltas por Python podría ser diferente a lo que se muestra a continuación. La razón es que, a diferencia de las listas que tienen un orden fijo, un diccionario es simplemente una tabla hash para la que no hay un orden fijo de las claves (como HashMaps en Java). El orden de las claves depende de cómo el algoritmo de hash asigna las claves, siendo normalmente arbitrario. Su código no debe depender del orden de las claves, y no debe sorprenderse si incluso una pequeña modificación en la forma en que su código usa un diccionario da como resultado un nuevo orden de claves.

```
>>> studentIds = {'knuth': 42.0, 'turing': 56.0, 'nash': 92.0}
>>> studentIds ['turing']
56,0
>>> studentIds ['nash'] = 'noventa y dos'
>>>  studentIds
{'knuth': 42.0, 'turing': 56.0, 'nash': 'noventa y dos'}
>>> del studentIds ['knuth']
>>>  studentIds
{'turing': 56.0, 'nash': 'noventa y dos'}
>>> studentIds ['knuth'] = [42.0, 'cuarenta y dos']
>>>  studentIds
{'knuth': [42.0, 'cuarenta y dos'], 'turing': 56.0, 'nash': 'noventa y dos'}
>>> studentIds.keys ()
['knuth', 'turing', 'nash']
>>> studentIds.values()
[[42.0, 'cuarenta y dos'], 56.0, 'noventa y dos']
>>> studentIds.items ()
[('knuth', [42.0, 'cuarenta y dos']), ('turing', 56.0), ('nash', 'noventa y dos')]
>>> len (studentIds)
3
```

Al igual que con las listas anidadas, también puede crear diccionarios de diccionarios.

Ejercicio: diccionarios
Utilice dir y help para conocer las funciones que puede utilizar con los diccionarios.

Escritura de Scripts
Ahora que ya sabe cómo usar Python de forma interactiva, escribamos un script de Python sencillo que demuestra el bucle for de Python. Abra el archivo llamado foreach.py, que se encuentra dentro de la carpeta *python_basics* que debe contener el siguiente código:

```
# This is what a comment looks like
fruits = ['apples', 'oranges', 'pears', 'bananas']
for fruit in fruits:
    print(fruit + ' for sale')

fruitPrices = {'apples': 2.00, 'oranges': 1.50, 'pears': 1.75}
for fruit, price in fruitPrices.items():
    if price < 2.00:
        print('%s cost %f a pound' % (fruit, price))
    else:
        print(fruit + ' are too expensive!')
```

En la línea de comando, use el siguiente comando en el directorio que contiene foreach.py:
[IA@lab ~/python_basics]$ python foreach.py
apples for sale
oranges for sale
pears for sale
bananas for sale
apples are too expensive!
oranges cost 1.500000 a pound
pears cost 1.750000 a pound

Recuerde que pueda que no obtenga el mismo resultado que en el tutorial, ya que se hace un bucle sobre las claves/keys que son desordenadas. Para obtener más información sobre las estructuras de control (por ejemplo, if y else) en Python, consulte la página web de python.
Si te gusta la programación funcional, también te puede interesar map y filter:
Las funciones *map()/filter()* reciben como parámetro una función y una lista/tupla/conjunto y aplican la función a la lista/tupla/conjunto.
Lambda, se refiere a una pequeña función anónima. Al contrario que una función normal, no la definimos con la palabra clave estándar *def* que se utiliza en *Python*. En su lugar, las funciones Lambda se definen como una línea que ejecuta una sola expresión. Este tipo de funciones pueden tomar cualquier número de argumentos, pero solo pueden tener una expresión.

```
>>> list(map(lambda x: x*x, [1,2,3]))
[1, 4, 9]
>>> list(filter(lambda x: x> 3, [1,2,3,4,5,4,3,2,1]))
[4, 5, 4]
```

El siguiente fragmento de código muestra mas construcciones de listas en Python:

```
nums = [1,2,3,4,5,6]
plusOneNums = [x+1 for x in nums]
oddNums = [x for x in nums if x % 2 == 1]
print(oddNums)
oddNumsPlusOne = [x+1 for x in nums if x % 2 ==1]
print(oddNumsPlusOne)
```

Este código está en un archivo llamado listcomp.py, que puede ejecutar:

[IA@lab ~/python_basics]$ python listcomp.py
[1,3,5]
[2,4,6]

Ejercicio: Comprensiones de listas
Escriba el código que, a partir de una lista, genere una nueva lista que contenga las cadenas en minúsculas que tengan una longitud superior a cinco de la lista de entrada. Puede encontrar la solución en listcomp2.py.
¡Cuidado con las tabulaciones!
A diferencia de muchos otros lenguajes, Python usa la tabulación en el código fuente para la interpretación. Entonces, por ejemplo, para el siguiente script:

```
if 0 == 1:
    print ('Estamos en un mundo de dolor aritmético')
print ('Gracias por jugar')
```

El resultado será:
Gracias por jugar
Pero si hubiéramos escrito:

```
if 0 == 1:
    print ('Estamos en un mundo de dolor aritmético')
    print ('Gracias por jugar')
```

No habría salido nada. ¡ten cuidado con la tabulación! Es mejor usar cuatro espacios para la tabulación: eso es lo que usa el código Python.

Tabulación vs espacios
Debido a que Python usa la tabulación para la evaluación de código, necesita realizar un seguimiento del nivel de tabulación en bloques de código. No se puede mezclar tabulaciones con espacios, uno u otro.

Funciones de escritura
Como en Java, en Python puedes definir tus propias funciones:

```
fruitPrices = {'apples':2.00, 'oranges': 1.50, 'pears': 1.75}

def buyFruit(fruit, numPounds):
    if fruit not in fruitPrices:
        print("Sorry we don't have %s" % (fruit))
    else:
        cost = fruitPrices[fruit] * numPounds
        print("That'll be %f please" % (cost))

# Main Function
if __name__ == '__main__':
    buyFruit('apples',2.4)
    buyFruit('coconuts',2)
```

En lugar de tener una función principal como en Java, la verificación __name__ == '__main__' se usa para delimitar las expresiones que se ejecutan cuando se llama al archivo como un script desde la línea de comandos.

Guarde este script como *fruit.py* y ejecútelo:
[IA@lab ~]$ python fruit.py
```
That'll be 4.800000 please
Sorry we don't have coconuts
```

Ejercicio avanzado
Escriba una función quickSort en Python usando listas, es decir, ordenar una lista de menor a mayor. Puede encontrar la solución en quickSort.py.

Conceptos básicos de los objetos
Aunque esto no es una clase de programación orientada a objetos, tendrá que usar algunos objetos durante los proyectos que se realizarán, por lo que vale la pena cubrir los conceptos básicos de los objetos en Python. Un objeto encapsula datos y proporciona funciones para interactuar con esos datos.

Definición de clases
Aquí hay un ejemplo de cómo definir una clase llamada FruitShop:

```
class FruitShop:

    def __init__(self, name, fruitPrices):
        """ COMENTARIO
            name: Name of the fruit shop

            fruitPrices: Dictionary with keys as fruit
            strings and prices for values e.g.
            {'apples':2.00, 'oranges': 1.50, 'pears': 1.75}
        """
        self.fruitPrices = fruitPrices
        self.name = name
        print('Welcome to %s fruit shop' % (name))

    def getCostPerPound(self, fruit):
        """
            fruit: Fruit string
        Returns cost of 'fruit', assuming 'fruit'
        is in our inventory or None otherwise
        """
        if fruit not in self.fruitPrices:
            return None
        return self.fruitPrices[fruit]

    def getPriceOfOrder(self, orderList):
        """
            orderList: List of (fruit, numPounds) tuples

        Returns cost of orderList, only including the values of
        fruits that this fruit shop has.
        """
        totalCost = 0.0
        for fruit, numPounds in orderList:
```

```
            costPerPound = self.getCostPerPound(fruit)
            if costPerPound != None:
                totalCost += numPounds * costPerPound
        return totalCost

    def getName(self):
        return self.name
```

La clase FruitShop tiene algunos datos, el nombre de la tienda y los precios por libra de algunas frutas, y proporciona funciones o métodos sobre estos datos. ¿Qué ventaja tiene una clase?

1. Encapsular los datos evita que se modifiquen o utilicen de forma inapropiada.
2. La abstracción que proporcionan los objetos facilita la escritura de código de propósito general.

Usando objetos

Entonces, ¿cómo hacemos un objeto y lo usamos? Asegúrese de tener la implementación de FruitShop en shop.py. Luego, importamos el código de este archivo (haciéndolo accesible a otros scripts) usando import shop, ya que shop.py es el nombre del archivo. Luego, podemos crear objetos FruitShop que se encuentra en el fichero shopTest.py:

```
import shop

shopName = 'Natur'
fruitPrices = {'apples': 1.00, 'oranges': 1.50, 'pears': 1.75}
natur = shop.FruitShop(shopName, fruitPrices)
applePrice = natur.getCostPerPound('apples')
print(applePrice)
print('Apples cost $%.2f at %s.' % (applePrice, shopName))

otherName = 'Todo de Temporada'
otherFruitPrices = {'kiwis': 6.00, 'apples': 4.50, 'peaches': 8.75}
otherFruitShop = shop.FruitShop(otherName, otherFruitPrices)
otherPrice = otherFruitShop.getCostPerPound('apples')
print(otherPrice)
print('Apples cost $%.2f at %s.' % (otherPrice, otherName))
print("My, that's expensive!")
```

Ejecutando éste código, se obtiene lo siguiente:

[IA@lab ~]$ python shopTest.py

```
Welcome to Natur fruit shop
1.0
Apples cost $1.00 at Natur.
Welcome to Todo de Temporada fruit shop
4.5
Apples cost $4.50 at Todo de Temporada.
```

```
My, that's expensive!
```

En el fichero shopTest.py en la primera línea hemos indicado que importe (import shop) todas las funciones y clases de shop.py. La línea natur = shop.FruitShop(shopName, fruitPrices) construye una *instancia* de la clase FruitShop definida en *shop.py*, llamando a la función __init__ de esa clase. Tenga en cuenta que solo pasamos dos argumentos, mientras que __init__ parece tomar tres argumentos: (self, name, fruitPrices). La razón de esto es que todos los métodos de una clase siempre tienen *self* como primer argumento. El valor de la variable propia se establece automáticamente al mismo objeto; al llamar a un método, solo proporciona los argumentos restantes. La variable *self* contiene todos los datos (nombre y precio de la fruta) para la instancia específica actual (similar a Java). Más en documentos de Python.

Variables estáticas frente a variables de instancia

El siguiente ejemplo ilustra cómo usar variables estáticas y de instancia en Python. Cree el fichero person_class.py y pegue el siguiente código:

```
class Person:

    population = 0

    def __init__(self, myAge):
        self.age = myAge
        Person.population += 1

    def get_population(self):
        return Person.population

    def get_age(self):
        return self.age
```

Primero compilamos el script:
[IA@lab ~]$ python person_class.py

Ahora usa la clase de la siguiente manera:

```
>>> import person_class
>>> p1 = person_class.Person (12)
>>> p1.get_population ()
1
>>> p2 = person_class.Person (63)
>>> p1.get_population ()
2
>>> p2.get_population ()
2
>>> p1.get_age ()
12
```

```
>>> p2.get_age ()
63
```

En el código anterior, la edad es una variable de instancia y la población es una variable estática. La variable población es compartido por todas las instancias de la clase Person, mientras que cada instancia tiene su propia variable de edad.

Más consejos y trucos de Python

En este tutorial se ha repasado brevemente algunos aspectos importantes de Python. Aquí mas cosas útiles:

- Utilice range para generar una secuencia de números enteros, útil para generar índices tradicionales para bucles:

```
lst=[11,12,13,14]

for index in range(3):
    print(lst[index])
```

- Después de importar un archivo, si edita el archivo de origen, los cambios no se propagarán inmediatamente en el intérprete. Para esto, use el comando reload:

```
>>> reload(shop)
```

Solución de problemas

Estos son algunos problemas (y sus soluciones) que se encuentran comúnmente:

- **Problema:**

ImportError: No module named py

Solución:

Cuando utilice la importación, no incluya el ".py" del nombre del archivo.
Por ejemplo, debería decir: import shop
NO: import shop**.py**

- **Problema:**

NameError: name 'MY VARIABLE' is not defined
Incluso después de importar, puede ver esto.

Solución:

Para acceder a un miembro de un módulo, debe escribir MODULE NAME.MEMBER NAME, donde MODULE NAME es el nombre del archivo .py y MEMBER NAME es el nombre de la variable (o función) a la que está intentando acceder.

- **Problema:**

TypeError: 'dict' object is not callable

Solución:

Las búsquedas en el diccionario se realizan utilizando corchetes: [ ] NO paréntesis: ( ).

- **Problema:**

ValueError: too many values to unpack

Solución:

Asegúrese de que el número de variables que está asignando en un bucle for coincida con el número de elementos en cada elemento de la lista. De manera similar para trabajar con tuplas.
Por ejemplo, si par es una tupla de dos elementos (por ejemplo, par = ('manzana', 2.0)) entonces el El siguiente código provocaría el error "too many values to unpack error":

```
(a, b, c) = par
```

Otro ejemplo que daría error:

```
pairList = [('manzanas', 2.00), ('naranjas', 1.50), ('peras', 1.75)]
for fruit, price, color in pairList:
    print('%s fruit costs %f and is the color %s' % (fruit, price, color))
```

- **Problema:**

AttributeError: el objeto 'lista' no tiene atributo 'length' (o algo similar)

Solución:

Para obtener la longitud de una lista se realiza mediante len(NOMBRE_DE_LA_LISTA).

- **Problema:**

Los cambios en un archivo no tienen efecto.

Solución:

1. Asegúrese de guardar todos sus archivos después de cualquier cambio.
2. Si está editando un archivo en una ventana diferente a la que está usando para ejecutar python, asegúrese de volver a cargar "reload" (NOMBRE_DEL_MODULO) para garantizar que sus cambios sean visibles. "reload" funciona de manera similar a importar.

Más referencias

- El lugar para obtener más información sobre Python: www.python.org

Autocalificación:

Para que se familiarice con el fichero autograder (autocalificación), ha de realizar tres ejercicios. Todos los ficheros relacionados se pueden encontrar en el zip *tutorial.zip*. Descomprima este archivo y examine su contenido:

[IA@lab ~]$ unzip tutorial.zip
[IA@lab ~]$ cd tutorial
[IA@lab ~/tutorial]$ ls
addition.py
autograder.py
buyLotsOfFruit.py
grading.py
projectParams.py
shop.py
shopSmart.py
testClasses.py
testParser.py
test_cases
tutorialTestClasses.py

Contiene varios archivos que ha de editar y ejecutar:

- addition.py: archivo fuente para la pregunta 1
- buyLotsOfFruit.py: archivo fuente para la pregunta 2
- shop.py: archivo fuente para la pregunta 3
- shopSmart.py: archivo fuente para la pregunta 3
- autograder.py: secuencia de comandos de calificación automática (ver más abajo)

y otros que puedes ignorar:

- test_cases: el directorio contiene los casos de prueba para cada pregunta
- grading.py: código de autocalificador
- testClasses.py: código de autograder
- tutorialTestClasses.py: clases de prueba para este proyecto en particular
- projectParams.py: parámetros del proyecto

El comando python autograder.py califica su solución de los tres problemas. Si lo ejecutamos antes de editar cualquier archivo obtenemos una página o dos de salida:

[IA@lab ~/tutorial]$ python autograder.py

```
Starting on 1-21 at 23:39:51

Question q1
===========
*** FAIL: test_cases/q1/addition1.test
*** add(a,b) must return the sum of a and b
*** student result: "0"
*** correct result: "2"
```

```
*** FAIL: test_cases/q1/addition2.test
*** add(a,b) must return the sum of a and b
*** student result: "0"
*** correct result: "5"
*** FAIL: test_cases/q1/addition3.test
*** add(a,b) must return the sum of a and b
*** student result: "0"
*** correct result: "7.9"
*** Tests failed.
### Question q1: 0/1 ###

Question q2
===========
*** FAIL: test_cases/q2/food_price1.test
*** buyLotsOfFruit must compute the correct cost of the order
*** student result: "0.0"
*** correct result: "12.25"
*** FAIL: test_cases/q2/food_price2.test
*** buyLotsOfFruit must compute the correct cost of the order
*** student result: "0.0"
*** correct result: "14.75"
*** FAIL: test_cases/q2/food_price3.test
*** buyLotsOfFruit must compute the correct cost of the order
*** student result: "0.0"
*** correct result: "6.4375"
*** Tests failed.
### Question q2: 0/1 ###

Question q3
===========
Welcome to shop1 fruit shop
Welcome to shop2 fruit shop
*** FAIL: test_cases/q3/select_shop1.test
*** shopSmart(order, shops) must select the cheapest shop
*** student result: "None"
*** correct result: "<FruitShop: shop1>"
Welcome to shop1 fruit shop
Welcome to shop2 fruit shop
*** FAIL: test_cases/q3/select_shop2.test
*** shopSmart(order, shops) must select the cheapest shop
*** student result: "None"
*** correct result: "<FruitShop: shop2>"
Welcome to shop1 fruit shop
Welcome to shop2 fruit shop
Welcome to shop3 fruit shop
*** FAIL: test_cases/q3/select_shop3.test
*** shopSmart(order, shops) must select the cheapest shop
*** student result: "None"
*** correct result: "<FruitShop: shop3>"
*** Tests failed.
### Question q3: 0/1 ###

Finished at 23:39:51

Provisional grades
==================
Question q1: 0/1
Question q2: 0/1
Question q3: 0/1
------------------
Total: 0/3

Your grades are NOT yet registered. To register your grades, make sure to follow
your instructor's guidelines to receive credit on your project.
```

Para cada una de las tres preguntas, se muestran los resultados, nota y un resumen final. Como aún no ha resuelto las preguntas, todas las pruebas fallarán. A medida que resuelve cada pregunta, es posible que algunas pruebas pasen mientras que otras fallen. Cuando para una pregunta pasan todas las pruebas, se obtiene la máxima puntuación.
Si se observa los resultados de la pregunta 1, se puede ver que ha fallado tres pruebas con el mensaje de error "add(a,b) must return the sum of a and b". La respuesta que da su código es siempre 0, pero la respuesta correcta es diferente.

Pregunta 1: Addition
Abra addition.py y observe la definición de add:

```
def add (a, b):
    "Return the sum of a and b"
    "*** YOUR CODE HERE ***"
    return 0
```

Modificamos:

```
def add (a, b):
    "Return the sum of a and b"
    print("Passed a=%s and b=%s, returning a+b=%s" % (a,b,a+b))
    return a+b
```

Ahora vuelva a ejecutar el autograder (omitiendo los resultados de las preguntas 2 y 3):

[IA@lab ~/tutorial]$ python autograder.py -q q1

```
Starting on 1-21 at 23:52:05
Question q1
===========
Passed a=1 and b=1, returning a+b=2
*** PASS: test_cases/q1/addition1.test
*** add(a,b) returns the sum of a and b
Passed a=2 and b=3, returning a+b=5
*** PASS: test_cases/q1/addition2.test
*** add(a,b) returns the sum of a and b
Passed a=10 and b=-2.1, returning a+b=7.9
*** PASS: test_cases/q1/addition3.test
*** add(a,b) returns the sum of a and b
### Question q1: 1/1 ###

Finished at 23:41:01

Provisional grades
==================
Question q1: 1/1
Question q2: 0/1
Question q3: 0/1
------------------
Total:1/3
```

Ahora pasa todas las pruebas y obtiene la máxima puntuación en la pregunta 1. Observe las líneas "*** PASS: ...".

Pregunta 2: función buyLotsOfFruit

Agregue una función buyLotsOfFruit (orderList) en buyLotsOfFruit.py que toma una lista de tuplas (fruta, libra) y devuelve el coste de su lista. Si hay alguna fruta en la lista que no aparece en fruitPrices, debería imprimir un mensaje de error y devolver Ninguno. Por favor no modifique la variable fruitPrices.

Ejecute python autograder.py hasta que la pregunta 2 pase todas las pruebas y obtenga la máxima puntuación. Cada prueba confirmará que buyLotsOfFruit (orderList) devuelve la respuesta correcta dadas varias posibles entradas. Por ejemplo, para el caso de test_cases/q2/food_price1.test, la prueba sería el siguiente:

```
Cost of [('apples', 2.0), ('pears', 3.0), ('limes', 4.0)] is 12.25
```

Pregunta 3: función shopSmart

Complete la función shopSmart(orders, shops) en shopSmart.py, que toma una orderList (igual que en ejemplo pasado FruitShop.getPriceOfOrder) y una lista de FruitShop y devuelva la FruitShop donde su pedido cueste la menor cantidad en total. No cambie el nombre del archivo ni nombres de variables. Tenga en cuenta que se proporciona la implementación de shop.py como un archivo de "soporte".

Ejecute python autograder.py hasta que la pregunta 3 pase todas las pruebas y obtenga la máxima puntuación. Cada prueba confirmará que shopSmart (orders, shops) devuelve la respuesta correcta dadas varias posibles entradas. Por ejemplo, con las siguientes definiciones de variables:

```
orders1 = [('apples',1.0), ('oranges',3.0)]
orders2 = [('apples',3.0)]
dir1 = {'apples': 2.0, 'oranges':1.0}
shop1 = shop.FruitShop('shop1',dir1)
dir2 = {'apples': 1.0, 'oranges': 5.0}
shop2 = shop.FruitShop('shop2',dir2)
shops = [shop1, shop2]

test_cases/q3/select_shop1.test tests whether:
      shopSmart.shopSmart(orders1, shops) == shop1
and test_cases/q3/select_shop2.test tests whether:
      shopSmart.shopSmart(orders2, shops) == shop2
```