Introducción a la programación con Python 3

Conceptos fundamentales

Ing. Jose Eduardo Laruta Espejo

11 de febrero de 2021

Universidad La Salle

Contenido i

Estructuras de datos

Listas

- Las Listas almacenan una secuencia de objetos mutables.
- Los objetos almacenados no tienen que ser del mismo tipo.

```
>>> frutas = ['manzana', 'durazno', 'pera', 'banana']
>>> frutas[0]
'manzana'
>>> otras_frutas = ['naranja', 'papaya', 'kiwi']
>>> frutas + otras_frutas
['manzana', 'durazno', 'pera', 'banana', 'naranja', 'papaya'
    , 'kiwi']
```

Listas

Se puede acceder a un elemento de una lista usando el operador []

```
>>> frutas[0]
'manzana'
>>> frutas[2]
'pera'
>>> frutas[-1]
'banana'
```

Listas

Se puede acceder a varios elementos contiguos con el operador slice usando la siguiente forma general: `frutas[inicio, fin]` va a retornar los elementos en los indices `inicio, inicio+1, ..., fin-1`.

```
>>> frutas[0:2]
['manzana', 'durazno']
>>> frutas[:3]
['manzana', 'durazno', 'pera']
>>> frutas[2:]
['pera', 'banana']
```

Listas

Las listas pueden contener cualquier objeto de Python. Podemos tener listas de listas.

```
>>> matriz = [[1, 2, 3], [4, 5, 6]]
>>> matriz[1][2]
6
>>> lista = ['hola', 2, ['a', 2, 'adios'], 45]
>>> lista[-1]
45
```

Ejercicio: explorar métodos de una lista con el comando `dir(lista)` y consultar su documentación con `help(lista.metodo)`

Tuplas

Las tuplas son estructuras similares a las listas con la diferencia de que son inmutables, es decir, sus elementos no se pueden cambiar luego de crearse la tupla.

```
>>> par = (3, 4)
>>> par[0]
3
>>> x, y = par
>>> y
4
>>> par[1] = 6
```

Ejercicio: explorar métodos de una lista con el comando `dir(lista)` y consultar su documentación con `help(lista.metodo)`

Conjuntos(Sets)

Los conjuntos (Sets) son estructuras no ordenadas que no pueden tener items duplicados. Se puede crear un Set a partir de una lista.

```
>>> formas = ['circulo', 'cuadrado', 'rombo', 'triangulo']
>>> set_formas = set(formas)
>>> set_formas2 = {'circulo', 'cuadrado', 'rombo', '
    triangulo'}
```

Se pueden aplicar operaciones comunes de conjuntos (Unión, intersección) a los sets de Python.

Diccionarios

Una de las estructuras más útiles de Python es el Diccionario, el cual almacena un mapeo de un tipo de datos (key) a otro tipo (value). El key debe ser un tipo inmutable, mientras que el value puede ser cualquier tipo de python.

```
>>> estudiante = {'nombre':'jose', 'edad':22, 'notas':[78,
    90, 88]}
>>> estudiante['notas']
[78, 90, 88]
>>> estudiante.keys()
['nombre', 'edad', 'notas']
>>> estudiante.values()
['jose', 22, [78. 90, 88]]
>>> estudiante.items()
[('nombre','jose'), ('edad',22), ('notas',[78, 90, 88])]
>>> len(estudiante)
3
```

Al igual que con las listas anidadas, se pueden crear diccionaros de diccionarios.

Scripts de python

El intérprete de python puede ejecutar comandos de forma interactiva, pero para tareas de programación más serias y completas usaremos archivos de código fuente denominados `scripts`. Un script es un conjunto de sentencias de python que se ejecutan de forma ordenada.

```
1  # esto es un comentario
2  frutas = ['manzana', 'durazno', 'pera', 'banana']
3  for fruta in frutas:
4      print(fruta + ' a la venta')
5  frutasdic = ['manzana':15, 'durazno':20, 'pera':10, 'banana'
       :5]
6  for fruta, precio in frutasdic.items():
7      if precio < 12:
8          print('el precio de la {} es {}'.format fruta,
               precio)
9      else:
10         print('{} es una fruta muy cara'.format(fruta))
```

Scripts de python

Para ejecutar un script de python se debe invocar al intérprete ingresando el nombre del archivo como argumento.

```
$ python frutas.py
```

List Comprehension

Se puede construir una lista en python usando una forma de notación especial que se define en una expresión dentro de la lista.

```
1 nums = [1,2,3,4,5,6]
2 masuno = [x + 1 for x in nums]
3 impares = [x for x in nums if x % 2 == 1]
4 impares_mas_uno = [x + 1 for x in nums if x % 2 == 1]
```

List Comprehension (Ejercicio)

Dada la siguiente lista, escriba un list comprehension que genere una versión en minúsculas de cada cadena que tenga longitud mayor que 5.

```
nums = ['Hola',
        'escuDEro',
        'Buenos Dias',
        'ADIOS',
        'RinoceronTE',
        'algoritmos',
        'Inteligencia Artificial']
```

Funciones

Al igual que en Java o en C++, en Python podemos declarar nuestras propias funciones:

```
1  frutas = ['manzana':15, 'durazno':20, 'pera':10, 'banana':5]
2
3  def comprarFruta(fruta, kilos):
4      if fruta not in frutas:
5          print('Lo siento, no contamos con {}'.format fruta)
6      else:
7          costo = frutas[fruta] * kilos
8          print('Son {} Bolivianos, por favor'.format(costo))
9
10 comprarFruta('pera', 2)
11 comprarFruta('mango', 10)
```

Clases y Objetos

Un objeto encapsula datos y provee funciones para interactuar con los datos. En Python se pueden definir objetos personalizados usando Clases.

```
class Perro:
    def __init__(self, nombre, raza, edad):
        self.nombre = nombre
        self.raza = raza
        self.edad = edad
    def ladrar(self):
        print('guau')
    def saludar(self):
        print('guau, soy {} y soy de raza {}'.format(self.
            nombre, self.raza))
```

Encapsular datos previene su uso inadecuado.

La abstracción de objeto permite escribir código más sencillo de entender.

Clases y Objetos

Para aprovechar las ventajas de los objetos se debe instanciar los mismos en código.

```
1 cachuchin = Perro('cachuchin', 'chapi', 9)
2 cachuchin.ladrar()
3 cachuchin.saludar()
4
5 lassie = Perro('lassie', 'collie', 7)
6 lassie.saludar()
```

Encapsular datos previene su uso inadecuado.

La abstracción de objeto permite escribir código más sencillo de entender.

Práctica de Laboratorio 1

Laboratorio 1

El Laboratorio 1 tiene como objetivo introducir a la programación con Python, se cuenta con un proyecto autoevaluado sobre el cual el estudiante tendrá que realizar modificaciones segun los ejercicios. El archivo con lo necesario se encuentra en este link

Instrucciones i

Una vez descargado el archivo, descomprimir el contenido en alguna carpeta destinada a proyectos de programación. dentro de la carpeta encontrará varios archivos de python.

Para este Laboratorio, se tienen un conjunto de archivos que deberán editar:

- `addition.py`: ejercicio 1
- `buyLotsOfFruit.py`: ejercicio 2
- `shop.py`: ejercicio 3
- `shopSmart.py`: ejercicio 3

Instrucciones ii

Una vez editados los archivos correspondientes a los ejercicios, deberá ejecutar el autoevaluador para comprobar la solución.

El archivo para evaluar los ejercicios se llama `autograder.py`

Para ejecutar la evaluación de los ejercicios:

```
$ python autograder.py
```

Los archivos restantes se pueden ignorar, no los toque.

Ejercicio 1 (ejemplo) i

Abrir el archivo `addition.py` y ver la definición de add:

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

Modificar de la siguiente manera:

```
def add(a, b):
    "Retorna la suma de a y b"
    print('a={}, b={}, retornando a+b={}'.format(a,b,a+b))
    return a+b
```

Correr a evaluacion:

```
$ python autograder.py -q q1
```

Ejercicio 2

Añadir una función llamada `buyLotsOfFruit(orderList)` al archivo `buyLotsOfFruit.py` que tome como parámetro una lista de tuplas `(fruta, kilo)` y retorne el costo de la lista. Si aparece una fruta en la lista que no este presente en `fruitPrices` se debe imprimir un mensaje de error y retornar `None`.

No cambie la variable `fruitPrices`

Ejercicio 3

Complete la función `shopSmart(orders,shops)` en el archivo `shopSmart.py` que recibe un `orderList` (similar al del anterior ejercicio) y una lista de objetos `FruitShop` y retorne el objeto `FruitShop` que posea el menor costo total. No cambie los nombres de archivos o de variables.

Compruebe el funcionamiento usando el archivo de evaluación `autograder.py`