Inspira Crea Transforma

UNIVERSIDAD EAFIT®

Programación de Computadores

ST0240

Semestre 2017-1

48 horas semestral

Fundamentos de Programación

ST0286

Semestre 2017-1

48 horas semestral

Agenda

2 clases por Semana

15m Contenido Previo (Tareas, Lecturas, Ejercicios propuestos o Anterior)

30m Contenido de Clase

30m Ejercicios Prácticos

15m Cierre: Trabajo Individual (Por fuera de Clase)

1 clase por Semana

30 m Contenido Previo (Tareas, Lecturas, Ejercicios propuestos o Anterior)

50 m Contenido de Clase

20 m Break

50 m Ejercicios Prácticos

30 m Cierre: Trabajo Individual (Por fuera de Clase)

Agenda

30 m Contenido Previo (Tareas, Lecturas, Ejercicios propuestos o Anterior)

60 m Arreglos, Cadenas, Listas, Tuplas y Diccionarios

10 m Break

40 m Ejercicios Prácticos

30 m Taller

10 m Cierre: Trabajo Individual (Por fuera de Clase)

Strings

- Python tiene una antigua biblioteca llamada str, pero no debe usarse.
- Un string no es más que una cadena de caracteres la cual puede contener caracteres especiales, se puede crear con comillas simples o comillas dobles. Cuando se utilizan comillas triples significa que el string es realmente una sola línea a pesar de tener varias en el código.
- Los strings en Python son inmutables, lo que quiere decir que no pueden ser modificados, sin embargo se pueden hacer nuevas construcciones a partir de ellos (p.e. por medio de la concatenación)

- Los strings en Python pueden ser accedidos por medio del operador `[]` y tienen índice basado en cero.
- Si el índice está por fuera de los límites del string, Python lanzará un error.
- Una función importante es `len(string)`, la cual devolverá la longitud del string. Esta función sirve para cualquier estructura de datos que sea una secuencia.

```
s = 'hi'

print s[1]          ## i

print len(s)        ## 2

print s + ' there'  ## hi there
```

String - Métodos

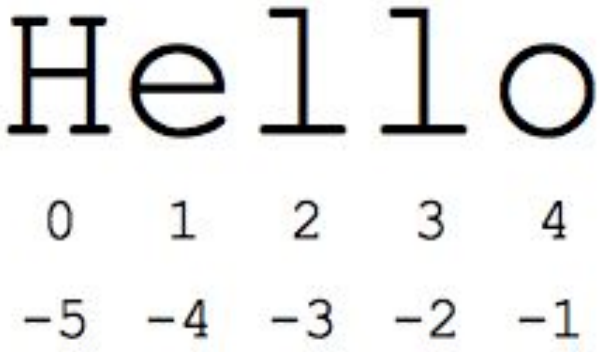
- s.lower, s.upper()
- s.strip()
- s.isalpha()/s.isdigit()/s.isspace()
- s.startswith('other'), s.endswith('')
- s.find('')
- s.replace('old', 'new')
- s.split('delim')
- s.join(list)

```
raw = r'this\t\n and that'

print raw      ## this\t\n and that

multi = """It was the best of times.

It was the worst of times."""
```

String Slices

- `s[1:4]` ➜ `'ell'`
- `s[1:]` ➜ `'ello'`
- `s[:]` ➜ `'Hello'`
- `s[:100]` ➜ `'Hello'`
- `s[-1]` ➜ `'o'`
- `s[:-3]` ➜ `'He'`
- `s[-3:]` ➜ `'llo'`

- `s[:n] + s[n:]` ➜ `s`



String %

```
# % operator

text = "%d little pigs come out or I'll %s and %s and %s" % (3, 'huff', 'puff', 'blow down')

# add parens to make the long-line work:

text = ("%d little pigs come out or I'll %s and %s and %s" %

(3, 'huff', 'puff', 'blow down'))
```

- Python permite formatear la salida por medio del operador %, dándole parámetros al string.
- %d ➜ `entero`
- `%c` ➜ `caracter`
- `%s` ➜ `string`

Ejercicios

El material y los ejercicios de esta clase están basados en Google Python Class. Realizar los ejercicios `string1.py` y `string2.py` de la carpeta `basic`.

https://goo.gl/uTfJfB

Arreglos

A diferencia de otros lenguajes, Python, no tiene arreglos, sin embargo tiene una estructuras de datos más generales que funcionan como arreglos. Python posee otros tipos de datos más complejos, que admiten una colección de datos:

- ➔ Tuplas;
- ➔ Listas;
- ➔ Diccionarios.

Pueden almacenar colecciones de datos de diversos tipos y se diferencian por su sintaxis y la forma como los datos pueden ser manipulados

Tuplas

Es una variable que permite almacenar varios datos inmutables (no pueden ser modificados una vez creados) de tipos diferentes:

E.g.

```
mi_tupla = ('Pepito Perez', 15, 2.5, 'Pepita', 25)
```

Tuplas

Se puede acceder a cada uno de los datos mediante su índice correspondiente, siendo cero (0), el índice del primer elemento:

```
print mi_tupla[1] # salida: 15
print mi_tupla[0] # salida: Pepito Perez
```

Tuplas

También se puede acceder a una porción de la tupla, indicando (opcionalmente) desde el índice de inicio hasta el índice de fin:

```
print mi_tupla[1:4] # salida: (15, 2.5, 'Pepita')
print mi_tupla[3:] # salida: ('Pepita', 25)
print mi_tupla[:2] # salida: ('Pepito Perez', 15)
```

Tuplas

Otra forma de acceder a la tupla de forma inversa (de atrás hacia adelante), es colocando un índice negativo:

```
print mi_tupla[-1] # Salida: 25
print mi_tupla[-2] # Salida: Pepita
```

Listas

Una lista es similar a una tupla con la diferencia fundamental de que permite modificar los datos una vez creados

```
mi_lista = [‘Pepito Perez’, 15, 2.5, ‘Pepita’, 25]
```

A las listas se accede igual que a las tuplas, por su número de índice:

```
print mi_lista[1] # Salida: 15
print mi_lista[1:4] # Salida: [15, 2.5, 'Pepita']
print mi_lista[-2] # Salida: Pepita
```

Listas

Las listas NO son inmutables: permiten modificar los datos una vez creados:

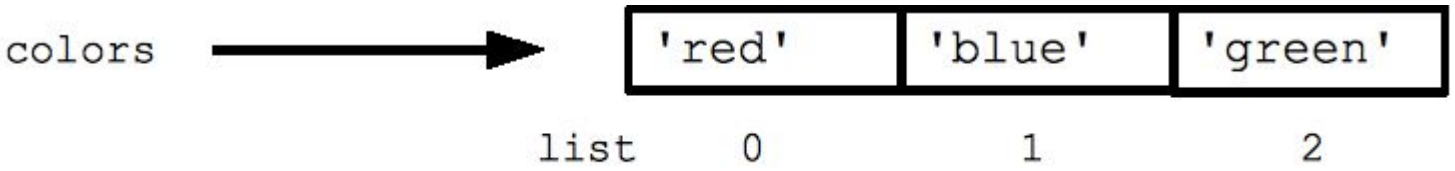
```
mi_lista[2] = 3.8 # el tercer elemento ahora es 3.8
```

Las listas, a diferencia de las tuplas, permiten agregar nuevos valores:

```
mi_lista.append('Nuevo Dato')
```

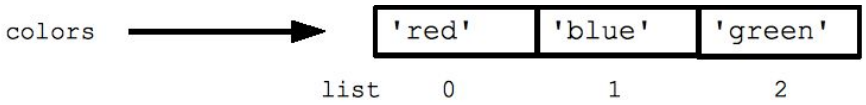
Listas - Ejemplos

```
colors = ['red', 'blue', 'green']

print colors[0]    ## red

print colors[2]    ## green

print len(colors)  ## 3
```



Listas - Ejemplos

```
colors = ['red', 'blue', 'green']

print colors[0]    ## red

print colors[2]    ## green

print len(colors)  ## 3
```



Listas - Ejemplos

```
squares = [1, 4, 9, 16]

sum = 0

for num in squares:

    sum += num

print sum  ## 30
```

Listas - Ejemplos

```
list = ['larry', 'curly', 'moe']

if 'curly' in list:

    print 'yay'
```

Listas - Métodos

- `list.append(elem)`
- `list.insert(index, elem)`
- `list.extend(list2)`
- `list.index(elem)`
- `list.remove(elem)`
- `list.sort()`
- `list.reverse()`
- `list.pop(index)`

Listas - Métodos

```
list = ['larry', 'curly', 'moe']

list.append('shemp')          ## append elem at end

list.insert(0, 'xxx')         ## insert elem at index 0

list.extend(['yyy', 'zzz'])   ## add list of elems at end

print list  ## ['xxx', 'larry', 'curly', 'moe', 'shemp', 'yyy', 'zzz']

print list.index('curly')     ## 2

list.remove('curly')          ## search and remove that element

list.pop(1)                   ## removes and returns 'larry'

print list  ## ['xxx', 'moe', 'shemp', 'yyy', 'zzz']
```

Listas - Métodos

```
list = [1, 2, 3]

print list.append(4)    ## NO, does not work, append() returns None

## Correct pattern:

list.append(4)

print list  ## [1, 2, 3, 4]
```

Listas - Métodos

```
list = []           ## Start as the empty list

list.append('a')    ## Use append() to add elements

list.append('b')
```

Listas - Métodos

```
list = ['a', 'b', 'c', 'd']

print list[1:-1]    ## ['b', 'c']

list[0:2] = 'z'     ## replace ['a', 'b'] with ['z']

print list          ## ['z', 'c', 'd']
```

Ejercicio Listas

Realizar los ejercicios `list1.py` y `list2.py` de google python class

Diccionarios

Mientras que a las listas y tuplas se accede solo y únicamente por un número de índice, los diccionarios permiten utilizar una clave para declarar y acceder a un valor:

```
mi_diccionario = {'clave_1': valor_1, 'clave_2': valor_2, 'clave_7': valor_7}
print mi_diccionario['clave_2'] # Salida: valor_2
```

Diccionarios

Un diccionario permite eliminar cualquier entrada:

```
del(mi_diccionario['clave_2'])
```

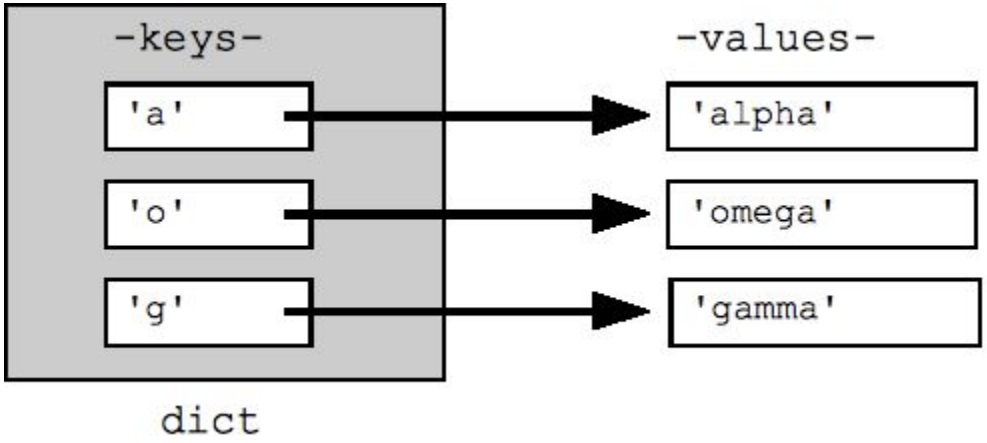
Al igual que las listas, el diccionario permite modificar los valores

```
mi_diccionario['clave_1'] = 'Nuevo Valor'
```

Diccionarios - Ejemplos

```
dict = {}

dict['a'] = 'alpha'

dict['g'] = 'gamma'

dict['o'] = 'omega'

print dict  ## {'a': 'alpha', 'o': 'omega', 'g': 'gamma'}

print dict['a']     ## Simple lookup, returns 'alpha'

dict['a'] = 6       ## Put new key/value into dict

'a' in dict         ## True

## print dict['z']                  ## Throws KeyError

if 'z' in dict: print dict['z']     ## Avoid KeyError

print dict.get('z')  ## None (instead of KeyError)
```

Diccionarios - Ejemplos



Diccionarios - Ejemplos

```
## By default, iterating over a dict iterates over its keys.
## Note that the keys are in a random order.
for key in dict: print key
## prints a g o

## Exactly the same as above
for key in dict.keys(): print key

## Get the .keys() list:
print dict.keys()  ## ['a', 'o', 'g']

## Likewise, there's a .values() list of values
print dict.values()  ## ['alpha', 'omega', 'gamma']

## Common case -- loop over the keys in sorted order,
## accessing each key/value
for key in sorted(dict.keys()):
  print key, dict[key]

## .items() is the dict expressed as (key, value) tuples
print dict.items()  ##  [('a', 'alpha'), ('o', 'omega'), ('g', 'gamma')]

## This loop syntax accesses the whole dict by looping
## over the .items() tuple list, accessing one (key, value)
## pair on each iteration.
for k, v in dict.items(): print k, '>', v
## a > alpha    o > omega     g > gamma
```

Diccionarios - Ejemplos

```
hash = {}
hash['word'] = 'garfield'
hash['count'] = 42
s = 'I want %(count)d copies of %(word)s' % hash  # %d for int, %s for string
# 'I want 42 copies of garfield'
```

Diccionarios - Ejemplos

```
var = 6
del var  # var no more!

list = ['a', 'b', 'c', 'd']
del list[0]     ## Delete first element
del list[-2:]   ## Delete last two elements
print list      ## ['b']

dict = {'a':1, 'b':2, 'c':3}
del dict['b']   ## Delete 'b' entry
print dict      ## {'a':1, 'c':3}
```

Ejercicios Diccionarios

Realizar los ejercicios `list1.py` y `list2.py` de google python class

Taller en Clase

Recursos Adicionales

https://spoj.com/

https://ideone.com/

https://www.pythonanywhere.com

http://python.swaroopch.com/

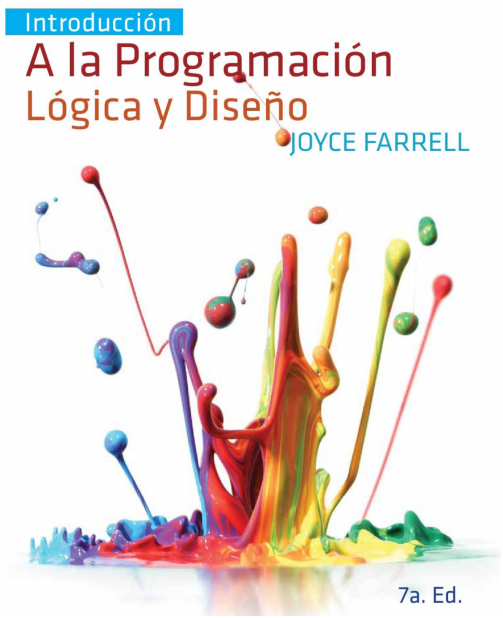
https://pragprog.com/book/gwpy2/practical-programming

https://coderbyte.com/course/learn-python-in-one-week

https://developers.google.com/edu/python/

Para la Próxima Clase/Semana

1. Leer Capítulos 4 y 5 del libro
2. Buscar un algoritmo para generar números primos
3. Consultar que es el triángulo de pascal
4. Exponer problemas que hayan tenido en el foro de la materia.
5. Realizar las tareas dejadas en clase



Referencias y Lecturas Adicionales

- Python Google Class - https://developers.google.com/edu/python/
- Hacker rank - https://www.hackerrank.com/dashboard