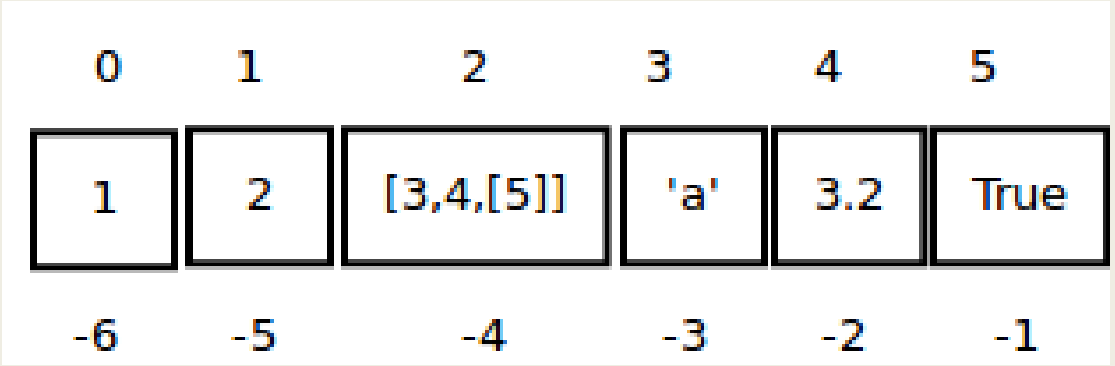
INTRODUCCIÓN A LA PROGRAMACIÓN

LISTAS

Rodrigo López A.

rilopez3@uc.cl

Listas en Python



- Python cuenta con una gran variedad de tipos de datos que permiten representar la información según cómo esté estructurada.
- La *listas* son utilizadas para modelar datos compuestos pero cuya cantidad y valor varían a lo largo del tiempo.
- Son secuencias variables/*mutables* y poseen un conjunto de métodos que facilitan el trabajo con grandes cantidades de variables.
- Las listas actúan como casilleros, en los cuales podemos guardar datos, la ventaja es que cada casillero tiene un numero único!!

- Las listas tienen un comportamiento muy similar a los String:
  - *Las listas tienen posiciones internas donde se almacenan datos, esto nos permite consultar los datos por su posición de manera sencilla* *mi_lista[0]*
  - *Las listas también largo que podemos consultar con la función* *len()*
  - *Podemos utilizar un ciclo for para recorrer todos los valores de la lista, tal como hacíamos con los strings, solo que ahora no obtendremos caracteres, sino que obtendremos cada dato almacenado.*
  - *Podemos utilizar el operador* *in* *tal como lo hacíamos con los strings, para verificar si un dato se encuentra dentro de la lista.*

Declarar una lista

- La notación para declarar/crear listas, es mediante una secuencia de valores/variables encerrados entre corchetes y separados por comas.
- La lista de los primeras 10 números primos seria

primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

len(primos) = > 10 primos[0] => 2

primos[3] => 7 primos[-1] => 29

lista_vacia = [] lista_vacia => []

Declarar una lista

primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

primos[2:5] => [5,7,11]

primos[2:2] => []

2 in primos => True
‘2’ in primos => False

```
for i in primos:
    print(i)
```

=>

```
2
3
5
7
11
13
17
19
23
29
```

Nuevos datos

- Para agregar nuevos datos, y por ende cambiar el tamaño de una lista, se usa el método mi_lista.append(nuevo_valor) esto agregara el nuevo valor al final de la lista.
- Ejemplo:

```
1   palabras = []
```

```
1   print(palabras)
2   print(len(palabras))
```

```
[]
0
```

```
1   palabras.append( 'hola' )
```

```
1   print(palabras)
2   print(len(palabras))
```

```
['hola']
1
```

```
1   palabras.append( 'mundo' )
```

```
1   print(palabras)
2   print(len(palabras))
```

```
['hola', 'mundo']
2
```

Unir/concatenar listas

- Es posible unir dos listas con el operador +
- Para unir listas es necesario que las variables/datos sean listas, la lista final es la concatenación de las listas en el orden en que se definan

```
1   lista1 = [1,2,3]
2   lista2 = ['a','b','c']
```

```
1   print(lista1 + lista2)
```

```
[1, 2, 3, 'a', 'b', 'c']
```

Extraer un índice de la lista

- Para realizar cambios en el contenido de una lista existen varios métodos útiles!!
- El método .pop() nos permite extraer un dato (se elimina de la lista) y guardarlo en una variable
- Si se entrega un índice especifico, se extraerá ese dato y se ajustaran los índices
- Si no se entrega ningún índice, se extrae el ultimo dato

```
1  mi_lista = ['a', 123, 'asd', 1.4]
2  x= mi_lista.pop(1)
```

```
1  print(x)
```

```
123
```

```
1  print(mi_lista)
```

```
['a', 'asd', 1.4]
```

```
1  mi_lista = ['a', 123, 'asd', 1.4]
2  x= mi_lista.pop()
```

```
1  print(x)
```

```
1.4
```

```
1  print(mi_lista)
```

```
['a', 123, 'asd']
```

Separar un string en una lista

- Es posible separar un string en una lista, dado un separador determinado!!
- Esto se logra utilizando el método .split(separador), el separador es lo que se utilizara para identificar cada dato de la lista
- El separador puede ser uno o más caracteres
- Los ítems obtenidos son almacenados como strings

```
1   dato = '1,2,3,4,5,6,7,8'
2   mi_lista = dato.split(',')
3   print(mi_lista)
```

```
['1', '2', '3', '4', '5', '6', '7', '8']
```

```
1   dato = 'esto es una oración'
2   mi_lista = dato.split(' ')
3   print(mi_lista)
```

```
['esto', 'es', 'una', 'oración']
```

```
1   dato = '123asd.456.asd.8.yuiasd1'
2   mi_lista = dato.split('asd')
3   print(mi_lista)
```

```
['123', '.456.', '.8.yui', '1']
```

Eliminar un valor de la lista

EXTRA

- El método .remove(valor_a _borrar) nos permite eliminar un dato de la lista según su valor
- Es necesario definir el dato que se desea borrar, y se eliminara el primer dato que tenga el valor entregado
- Tras eliminar el registro, se ajustan los índices de la lista

```
1   mi_lista = ['a', 123, 'asd', 1.4, 'asd']
2   mi_lista.remove( 'asd' )
```

```
1   print(mi_lista)
```

```
['a', 123, 1.4, 'asd']
```

EXTRA

Insertar un dato en una posición especifica

- El método .insert(indice , valor_a _insertar) nos permite insertar un dato en un índice especifico
- Es necesario definir el índice donde insertar el dato y el dato a insertar
- Después de insertar el dato se ajustan los índices de la lista

```
1   mi_lista = ['a', 123, 'asd', 1.4, 'asd']
2   mi_lista.insert( 1, 'test' )
```

```
1   print(mi_lista)
```

```
['a', 'test', 123, 'asd', 1.4, 'asd']
```

Copiar listas (OJO)

EXTRA

- Las listas son un tipo de dato MUTABLE, esto significa que es posible de modificar. PERO también significa que si copiamos una lista, las variables quedan vinculadas, lo cual provoca comportamientos EXTRAÑOS
- Para copiar una lista de manera correcta se debe usar el método .copy()

```
1  mi_lista = [1, 2, 3, 'a', 'b', 'c']
```

```
1  lista2 = mi_lista
```

```
1  lista2.append( 'dato nuevo' )
```

```
1  print(lista2)
```

```
[1, 2, 3, 'a', 'b', 'c', 'dato nuevo']
```

```
1  print(mi_lista)
```

EXTRAÑO

```
[1, 2, 3, 'a', 'b', 'c', 'dato nuevo']
```

```
1  mi_lista = [1, 2, 3, 'a', 'b', 'c']
```

```
1  lista2 = mi_lista.copy()
```

```
1  lista2.append( 'dato nuevo' )
```

```
1  print(lista2)
```

```
[1, 2, 3, 'a', 'b', 'c', 'dato nuevo']
```

```
1  print(mi_lista)
```

```
[1, 2, 3, 'a', 'b', 'c']
```

Tuplas

EXTRA

- Existe un tipo de dato especial que existe entre medio de números, strings y las listas: las **Tuplas**.
- Las **tuplas** son un tipo de dato similar a las listas, se diferencia en que las tuplas se declaran con paréntesis redondos y que una vez definidas no se pueden modificar.
- Se suele utilizar el nombre de **Dupla** para las **tuplas** de **2** valores.

```
1   numeros_en_lista = [1,2,3,4,5]
2   print(type(numeros_en_lista))
```

```
<class 'list'>
```

```
1   numeros_en_tupla = (1,2,3,4,5)
2   print(type(numeros_en_tupla))
```

```
<class 'tuple'>
```

```
1   numeros_en_lista[0] = 6
2   numeros_en_tupla[0] = 6
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-58-5aa1a96a3dae> in <module>()
      1 numeros_en_lista[0] = 6
----> 2 numeros_en_tupla[0] = 6

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

Diccionarios

EXTRA

- Existe otro estructura de datos similar a las listas, llamadas diccionarios.
- Los Diccionarios comparten los mismos métodos que las listas con la diferencia que los índices ya no son números, son strings.
- Esta forma de índices con strings permiten guardar información de manera mas fácil.
- Los diccionarios se definen con paréntesis de llave { ‘indice’ : valor }.

```
[2]  1  var = {}
     2
     3  var['hola'] = 'mundo'
     4
     5  print( var['hola'] )
     6

     mundo
```

```
[3]  1  var_2 = { '0' : 'cero', '1': 'uno' }
     2
     3  print( var_2 )
     4

     {'0': 'cero', '1': 'uno'}
```

```
1  for i in var_2:
2      print(i)
3      print(var_2[i])

0
cero
1
uno
```