UNIVERSIDAD CATÓLICA DE TEMUCO
INGENIERÍA CIVIL EN INFORMÁTICA
FACULTAD DE INGENIERÍA

Programación I

Listas y funciones

Profesor: Ignacio Lincolao Venegas

Ingeniería Civil Informática, UCT

Las listas

Una lista es una colección de elementos los cuales tienen un orden particular, se pueden hacer listas que contengan alfabetos, palabras, dígitos, nombres. En python para indicar que será una lista se utiliza los [] y para separar los los elementos se utiliza “,”.
Ejemplos de lista:

```
autos = [‘BMW’, ‘Mercedes-Benz’, ‘Audi’, ‘Lexus’, ‘Renault’, ‘Ford’]
# Para imprimir una lista se puede utilizar la función print
print(autos)
# Mostrará
# [‘BMW’, ‘Mercedes-Benz’, ‘Audi’, ‘Lexus’, ‘Renault’, ‘Ford’]
```

Accediendo a elementos de una lista

Las listas son colecciones ordenadas por lo que se puede obtener acceso a cualquier elemento de una lista a través de su posición, o índice, del elemento de nuestro interés. Para acceder solo debemos escribir el nombre de la lista y ingresar su índice entre los corchetes, recordando que la primera posición comienza desde 0. Ejemplo

```
nombres = [‘Juan’, ‘Pedro’, ‘Diego’]
# Para imprimir una lista se puede utilizar la función print
print(nombres[0])
# Mostrará
# Juan
print(nombres[2])
# Diego
```

Accediendo a elementos de una lista

En python se consideran los primeros elementos de una lista como la posición 0 y no la posición 1. Esto sucede en la mayoría de los lenguajes de programación, Esto es debido a que las operaciones de las listas son implementadas a bajo nivel.

```
nombres = [‘Juan’, ‘Pedro’, ‘Diego’]
# Para imprimir una lista se puede utilizar la función print
print(nombres[1])
# muestra
# Pedro
print(nombres[3])
# Se genera un error
```

Uso de las listas

Uno puede utilizar los valores individuales como si fueran cualquier otra variable, por ejemplo son los casos de concatenación, las operaciones aritméticas, operaciones booleanas, etc..

```
smartphones = [‘LG’, ‘Huawei’, ‘Xiaomi’]
mensaje = “La marca popular del año es ”+ smartphones[2] + “.”
print(mensaje)
# Mostrará por pantalla
# La marca popular del año es Xiaomi.
```

Agregando, cambiando y eliminando elementos de una lista

Al construir una lista uno puede realizar operaciones de cambiar los elementos, agregar nuevos elementos a la lista e incluso eliminarlo. un ejemplo simple es cuando se solicita un programa que contenga una lista de invitados. Uno quiere agregar nuevos invitados, eliminar invitados, cambiar los nombres de los invitados en caso de ingresarlos mal, operaciones básicas.

Modificando elementos de una lista

Modificar un elemento de una lista es similar a la sintaxis que se utiliza para acceder a un elemento de la lista. Para modificar el elemento de la lista es debe ingresar el nombre de la lista seguido de su índice y utilizar la asignación sobre el elemento. Ejemplo:

```
autos = [‘BMW’, ‘Mercedes-Benz’, ‘Audi’, ‘Lexus’, ‘Renault’, ‘Ford’]
print(autos)
# Imprime
# [‘BMW’, ‘Mercedes-Benz’, ‘Audi’, ‘Lexus’, ‘Renault’, ‘Ford’]
autos[1] = ‘KTM’
print(autos)
# Imprime
# [‘BMW’, ‘KTM’, ‘Audi’, ‘Lexus’, ‘Renault’, ‘Ford’]
```

Agregando un elemento al final de la lista

Para agregar un nuevo elemento a una lista se le debe pasar el nuevo valor a través de la función append(). Esto agregara un nuevo elemento al final de la lista.

```
autos = [‘BMW’, ‘Mercedes-Benz’, ‘Audi’, ‘Lexus’, ‘Renault’, ‘Ford’]
print(autos)
# Imprime
# [‘BMW’, ‘Mercedes-Benz’, ‘Audi’, ‘Lexus’, ‘Renault’, ‘Ford’]
autos.append(‘Cadillac’)
print(autos)
print(len(autos))
# Imprime
# [‘BMW’,‘Mercedes-Benz’, ‘Audi’, ‘Lexus’, ‘Renault’, ‘Ford’, ‘Cadillac’]
# 7
```

Agregando un elemento a la lista

Para insertar un nuevo elemento en cualquier posición de la lista se realiza usando el método insert(). Pero se debe especificar el índice en donde se agregara el nuevo elemento y el valor del nuevo elemento.

```
marcas_lacteos = [‘Colún’, ‘Líder’, ‘Loncoleche’, ‘Surlat’]
print(marcas_lacteos )
# Imprime
# [‘Colún’, ‘Líder’, ‘Loncoleche’, ‘Surlat’]
marcas_lacteos.insert(0,‘Soprole’)
print(marcas_lacteos)
print(len(marcas_lacteos))
# Imprime
# [‘Soprole’, ‘Colún’, ‘Líder’, ‘Loncoleche’, ‘Surlat’]
# 5
marcas_lacteos.insert(3,‘Quillayes’)
print(marcas_lacteos)
#[‘Soprole’, ‘Colún’, ‘Líder’, ‘Quillayes’ ,‘Loncoleche’, ‘Surlat’]
#     0         1        2          3             4            5
```

Eliminando un elemento de la lista

Si uno quiere eliminar un elemento de una lista se puede utilizar la declaración del, el cual eliminará el elemento de la posición indicada, permitiendo eliminar cualquier elemento de la lista

```
marcas_lacteos = [‘Colún’, ‘Líder’, ‘Loncoleche’, ‘Surlat’]
print(marcas_lacteos)
len(marcas_lacteos)
# Imprime
# [‘Colún’, ‘Líder’, ‘Loncoleche’, ‘Surlat’]
# 4
del marcas_lacteos[1]
print(marcas_lacteos)
len(marcas_lacteos)
#[‘Colún’, ‘Loncoleche’, ‘Surlat’]
# 3
```

Eliminando un elemento de la lista con pop()

Eliminando un elemento de la lista con el uso de pop(). En ocasiones se requiere eliminar un elemento de una lista, pero se quiere aún acceder a ese valor después de ser removida, un ejemplo: Cuando en una aplicación se quiere eliminar a un usuario de la lista de miembros activos pero se quiere agregar a la lista de miembros inactivos. El método pop() remueve de la lista al último elemento, pero este permite trabajar con él luego de removerlo.

```
miembros_activos = [“Rodolfo”, “Maria”,“Gabriel”, “Paulina”]
# Imprime
print(miembros_activos)
len(miembros_activos)
# [“Rodolfo”, “Maria”,“Gabriel”, “Paulina”]
# 4
miembro_inactivo = miembros_activos.pop()
print(miembros_activos)
print(miembro_inactivo)
# Imprime
# [“Rodolfo”, “Maria”,“Gabriel”]
# Paulina
print(“El último usuario removido fue”, miembro_inactivo)
# El último usuario removido fue Paulina
```

Eliminando un elemento de la lista con pop()

Con el método pop también se puede remover un elemento de una lista en cualquier posición, para eso se le debe ingresar por parámetro el índice del elemento

```
miembros_activos = [“Rodolfo”, “Maria”,“Gabriel”, “Paulina”]
print(miembros_activos)
len(miembros_activos)
# Imprime
# [“Rodolfo”, “Maria”,“Gabriel”, “Paulina”]
# 4
miembro_inactivo = miembros_activos.pop(2)
print(miembros_activos)
print(miembro_inactivo)
# [“Rodolfo”, “Maria”, “Paulina”]
# Gabriel
```

Eliminando un elemento de la lista

Algunas veces se quiere eliminar un elemento de la lista sin saber su posición y solo conocemos su valor, para esto se puede utilizar el método remove(). El método remove elimina al primer elemento que encuentra igual al parámetro ingresado, por lo que si existen valores iguales solo eliminará al primero que encuentre.

```
palabras = [“casa”, “gato”,“casa”, “gallina”]
print(palabras)
len(palabras)
# Imprime
# [“casa”, “gato”,“casa”, “gallina”]
# 4
palabras.remove(“casa”)
# Imprime
# [“gato”,“casa”, “gallina”]
# 3
```

Ejercicios en clases

Realiza las siguientes instruccione:

1. Crea una lista tenga 5 nombres.
2. Imprima por pantalla el quinto nombre.
3. Modifique el primer nombre y luego imprime por pantalla la lista.
4. Elimina el segundo elemento y luego imprime por pantalla la lista.
5. Agrega un elemento al final de la lista y luego imprime el largo de la lista
6. Agrega un elemento al inicio de la lista y luego imprime el largo de la lista
7. Agrega un elemento a la lista en una posición que no sea ni la primera, ni la última.
8. Elimina el último elemento de la lista, y luego imprime el elemento eliminado.
9. Elimina un elemento de la lista por su nombre y imprime la lista

```
Recordatorio

Crear una lista
lista = [“Silla”,”Escritorio”]
Modificar un elemento de la lista
lista[1] = “Mueble”
Eliminar un elemento de la lista
del lista[1]
Agregar un elemento al final de la lista
lista.append(“Refrigerador”)
Agregar un elemento en cualquier posición de la lista
lista.insert(3,”mesa”)
Eliminar un elemento pero obtener el elemento eliminado
valor_eliminado = lista.pop()
Eliminar un elemento en base a su valor
lista.remove(“Refrigerador”)
```

Ejercicios en clases

Realiza las siguientes instruccione:

1. Crea una lista tenga 5 nombres.
2. Imprima por pantalla el quinto nombre.
3. Modifique el primer nombre y luego imprime por pantalla la lista.
4. Elimina el segundo elemento y luego imprime por pantalla la lista.
5. Agrega un elemento al final de la lista y luego imprime el largo de la lista
6. Agrega un elemento al inicio de la lista y luego imprime el largo de la lista
7. Agrega un elemento a la lista en una posición que no sea ni la primera, ni la última.
8. Elimina el último elemento de la lista, y luego imprime el elemento eliminado.
9. Elimina un elemento de la lista por su nombre y imprime la lista

```
lista = [“Jaime”,”Cristian”, “Claudio”,”Felipe”,”Rodrigo”]
print(lista[4])
# Rodrigo
lista[0] = “Paulina”
print(lista)
# [“Paulina”,”Cristian”, “Claudio”,”Felipe”,”Rodrigo”]
del lista[1]
print(lista)
# [“Paulina”,“Claudio”,”Felipe”,”Rodrigo”]
lista.append(“Maria”)
print(lista)
# [“Paulina”,“Claudio”,”Felipe”,”Rodrigo”,”Maria”]
lista.insert(0,‘Antonia’)
print(lista)
# [“Antonia”,“Paulina”,“Claudio”,”Felipe”,”Rodrigo”,”Maria”]
lista.insert(3,‘Juan’)
print(lista)
# [“Antonia”,“Paulina”,“Claudio”,”Juan”,”Felipe”,”Rodrigo”,”Maria”]
nombre_eliminado= lista.pop()
print(nombre_eliminado) # “Maria”
lista.remove(“Felipe”)
print(lista)
# [“Antonia”,“Paulina”,“Claudio”,”Juan”,”Rodrigo”]
```

Organizando una lista

Algunas ocasiones se quiere tener la información organizada o tener en un particular orden. Python provee algunos métodos que permitirán organizar tu lista dependiendo de la situación. El método sort() es un método que permite ordenar una lista alfabéticamente, este cambio de orden afecta permanentemente a la lista.

```
autos = ['Audi', 'Ford','BMW','Mercedes-Benz','Lexus', 'Renault']
autos.sort()
print(autos)
# ['Audi', 'BMW', 'Ford', 'Lexus', 'Mercedes-Benz', 'Renault']
##  Para ordenar de forma invertida se puede utilizar el parámetro reverse en sort
autos.sort(reverse=True)
print(autos)
# ['Renault', 'Mercedes-Benz', 'Lexus', 'Ford', 'BMW', 'Audi']
```

Organizando una lista

En otras ocasiones se requiere sólo mostrar la lista de forma ordenada, pero sin modificar el orden original. La función sorted() permite mostrar la lista en un orden particular pero sin afectar al orden de la lista actual

```
autos = ['Audi', 'Ford','BMW','Mercedes-Benz','Lexus', 'Renault']
print(sorted(autos))
print(autos)
# ['Audi', 'BMW', 'Ford', 'Lexus', 'Mercedes-Benz', 'Renault']
# ['Audi', 'Ford','BMW','Mercedes-Benz','Lexus', 'Renault']

##  Para ordenar de forma invertida se puede utilizar el parámetro reverse en sorted
print(sorted(autos,reverse=True))
print(autos)
# ['Renault', 'Mercedes-Benz', 'Lexus', 'Ford', 'BMW', 'Audi']
# ['Audi', 'Ford','BMW','Mercedes-Benz','Lexus', 'Renault']
```

Organizando una lista

También se puede invertir una lista sin ordenar los elementos a través de el metodo reverse

```
autos = ['Audi', 'Ford','BMW','Mercedes-Benz','Lexus', 'Renault']
print(list(reverse(autos)))
print(autos)
# ['Audi', 'Ford','BMW','Mercedes-Benz','Lexus', 'Renault']
# ['Renault','Lexus','Mercedes-Benz','BMW', 'Ford', 'Audi']
```

Uso del for

Algunas veces es necesario recorrer todo el contenido de las lista y como las listas pueden llegar a contener miles de datos, no es conveniente ir uno a uno por cada dato ejemplo print(data[0]) print(data[1]) print(data[3]), Para tener fácil acceso a sus valores lo podemos hacer con el uso del for, el cual es una instrucción que permite repetir el número de veces que nosotros le indiquemos, en el caso de python nos permite, no facilita aún más recorrer las listas debido a la forma de expresarlo:

```
autos = ['Audi', 'Ford','BMW','Mercedes-Benz','Lexus', 'Renault']

for auto in autos:
    print(auto)

# Imprime
#'Audi'
#'Ford'
#'BMW'
#'Mercedes-Benz'
#'Lexus'
#'Renault'
```

Uso del for

Otras forma de recorrer las lista es a través de su forma numérica, con el uso del range(a,b), donde generará una serie de números desde 1 a 5, el cual da como resultado 1,2,3,4

```
autos = ['Audi', 'Ford','BMW','Mercedes-Benz','Lexus', 'Renault']

for auto in range(1,5):
    print(auto)

# Imprime
# Ford
# BMW
# Mercedes-Benz
# Lexus
```

Funciones

Las funciones dentro de los lenguajes de programación nos permite reutilizar código, Las funciones son bloques de código que están asignadas para un trabajo específico, con esto podemos llevar a cabo tareas las cuales se han definido como función, solo y cuando estas sean llamadas, por lo que no se ejecutan hasta llamarlas. Para definir una función se realiza de la siguiente manera:

```
# Creo una función que imprima por pantalla hola
def saluda():
    print(“Hola”)

# Utilizo la función
saluda()
saluda()
saluda()
# Imprimirá
# Hola
# Hola
# Hola
```

Funciones y parámetros

Las funciones permiten el ingreso de parámetros, esto permite que pueda se le puedan pasar valores para utilizarlo dentro de la función, un ejemplo, La función saluda ahora no solo muestre un hola, sino que se quiere que salude a una persona en concreto, pasaremos el nombre de esa persona.

```
# Creo una función que imprima por pantalla hola
def saluda(nombre):
    print(“Hola”,nombre)

# Utilizo la función
saluda(“Jose”)
nombre = “Paulina”
saluda(nombre )
# Imprimirá
# Hola Jose
# Hola Paulina
```

Funciones y parámetros

Al llamar una función es importante ingresar los argumentos en el orden que estan los parámetros de la función, esto que quiere decir el primer argumento que entrega a la función representa al parámetro a y el segundo argumento representa al parámetro b

```
# La función resta 2 números
def resta(a,b):
    print(a - b)

resta(2,3)
# resultado es -1
resta(3,2)
# resultado es 1
resta(-4,3)
# resultado es -7
resta(3,-4)
# resultado es 7
```

Funciones y parámetros

Otra alternativa para pasar los argumentos es a través del nombre del parámetro, así no es necesario respetar el orden, lo importante es que el parámetro exista dentro de la función

```
# La función resta 2 números
def resta(a,b):
    print(a - b)

resta(a=2,b=3)
# resultado es -1
resta(b=3,a=2)
# resultado es -1
resta(a=-4,b=3)
# resultado es -7
resta(b=3,a=-4)
# resultado es -7
```

Funciones y parámetros

También las funciones pueden asignarse parámetro por defecto en caso de no ingresarle un argumento

```
# La función resta 2 números
def resta(a=2,b=4):
    print(a - b)

resta(a=4)
# resultado es 0
resta(b=5)
# resultado es -3
resta(6)
# resultado es 2
resta(6,5)
# resultado es 1
```

Funciones y retorno

Las funciones también pueden retornar valores, para luego asignarlo a una variable dentro del programa principal

```
# Creo una función que imprima por pantalla hola
def suma(a,b):
    valor = a + b
    return valor

nuevo_valor = suma(5,6)
print(nuevo_valor) #  11
```

Funciones de utilidad para manejo de listas

La función range() tiene una función extra, el cual permite avanzar por el número de unidades que se le indica. Por lo que se la función range se compone de lo siguiente range(inicio,detiene,pasos), también es importante tener en cuenta que range solo recibe valores de tipo int, no acepta números flotante. Un ejemplo sería lo siguiente:

```
for x in range(2,21,2):
....print(x)

# imprime:  2 4 6 8 10 12 14 16 18 20

for x in range(2,21,1.5):
....print(x)

# Aparecerá por pantalla
#Traceback (most recent call last):
#  File "<stdin>", line 1, in <module>
#TypeError: 'float' object cannot be interpreted as an integer
```

Funciones de utilidad para manejo de listas

Otra de las funciones de utilidad es el uso de las funciones min() y max(), estas permiten obtener los valores mínimos y máximos, que son pasados como argumentos a la función

```
print(min([7,1,2,3,4,5,6]))
# Esto imprime 1
print(max([7,1,2,3,4,5,6]))
# Esto imprime 7
```

Funciones de utilidad para manejo de listas

Otras funciones de utilidad son split() y join(), La función split convierte una cadena de texto en una lista, y la función join convierte una lista en una cadena formada por los elementos de la separados por comas o el carácter que se le indique

```
# Ejemplo de spli
mensaje = “Hola mundo”
lista = mensaje.split()
print(lista)
# Imprime [“Hola”,”mundo”]
# También se le puede indicar el separador
mensaje  = “Buenos días, se encuentra bien.”
lista = mensaje.split(“, ”)
print(lista)
# [“Buenos días”, “Se encuentra bien”]
########################################
lista = [“Buenos días”, “Se encuentra bien”]
texto = “ ”.join(lista)
print(texto)
# imprime “Buenos días se encuentra bien”
# Donde “ ” contiene los caracter con los que unirá a los valores de la lista
```

Los diccionarios

Los diccionarios son estructuras de datos y a su vez un tipo de dato en python. Los diccionarios pueden almacenar una cantidad casi ilimitada de información. Los diccionarios nos pueden permitir crear representaciones como por ejemplo el de una persona, donde se puede almacenar su nombre, edad, ubicación, profesión y cualquier otro aspecto de la persona. Un ejemplo de diccionario

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Los diccionarios

Los diccionarios son colecciones de pares clave-valor. Las key están conectadas al valor, esto permite que usando la key puedas acceder al valor asociada a la key. los valores que se acceden por la clave puede ser un valor numérico, un string o una lista o otro diccionario. Para definir un diccionario se definen entre { } luego se ingresan valores los pares de key-valor dentro de las { }, dentro para separar los pares key-valor se realiza por comas.

```
persona = {‘nombre’: ‘Maria’, ‘edad’: 24, }
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Los diccionarios

Para acceder a un valor de los diccionario se puede acceder por el nombre de el diccionario

```
persona = {‘nombre’: ‘Maria’, ‘edad’: 24, }
print(persona[‘nombre’])
print(persona[‘edad’])
## Imprime por pantalla
# Maria
# 24
```

En ocasiones es necesario agregar nuevos pares de key-valor para eso simplemente se puede realizar agregando de la siguiente manera.

```
persona = {‘nombre’: ‘Raul’, ‘edad’: 24, }
# Para agregar un nuevo key-valor
persona[‘color_ojos’] = ‘café’ # Se agrega key-valor {‘color_ojos’: ‘cafe’}
persona[‘estatura’] = 1.65 # Se agrega key-valor {‘estatura’: 1.65}
print(persona)
# Imprime = {‘nombre’: ‘Raul’, ‘edad’: 24, ‘color_ojos’: ‘café’, ‘estatura’: 1.65}
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Los diccionarios

En el caso de querer inicializar una variable como un diccionario, pero no se le quiere ingresar datos (Crear un diccionario vacío), solo se debe definir la variable con {}

```
escuela = {}

escuela[‘estudiantes’] = 1200
escuela[‘Profesores’] = 14
print(escuelas)
# Imprime { “estudiantes”: 1200, “profesores”: 14}
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Los diccionarios

Modificar un valor en los diccionarios, se realiza al igual que cuando se modifica un valor en una lista, pero esta vez en vez que el índice sea un número, este se llama por la key

```
escuela = {“estudiantes”: 1200, “profesores”: 14}

escuela[‘estudiantes’] = 1586
escuela[‘Profesores’] = 22
escuela[‘profesores’] = 24
print(escuelas)
# Imprime { “estudiantes”: 1586,“profesores”: 24,“Profesores”: 22}


pelota = {“pos_x”: 0, “pos_y”: 12, “velocidad”: “baja”}

pelota[“pos_x”] += 1
pelota[“pos_y”] += 1

print(pelota)
# pelota
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Los diccionarios

Hay ocasiones donde necesitas eliminar un par de key-valor del diccionario para esto se puede utilizar la instrucción **del**

```
escuela = {“estudiantes”: 1586, “profesores”: 14}

del escuelas[‘profesores’]
print(escuelas)
# Imprime { “estudiantes”: 1586 }
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Los diccionarios

Para recorrer un diccionario dentro de un loop, se realiza un for normal pero a diferencia que se separan en dos valores key y valor, donde se utiliza la función items(), el cual devuelve los valores separados. Ejemplo:

```
escuela = {“estudiantes”: 1586, “profesores”: 14}
print(escuela.items())
# Imprime dict_items([(‘estudiantes’,1586),(‘profesores’,14)])

for key, value in escuela.items():
    print(key) ## En el primer ciclo imprime estudiantes, en el segundo ciclo imprime profesores
    print(value) ## En el primer ciclo imprime 1586, en el segundo ciclo imprime 14
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Los diccionarios

También se puede recorrer el diccionario pero solo obteniendo los nombres de las key con las función keys(). Ejemplo:

```
escuela = {“estudiantes”: 1586, “profesores”: 14}
print(escuela.keys())
# Imprime dict_keys(['estudiantes', 'profesores'])
# dict_keys es una vista de las keys que se puede iterar por medio de un for

for key in escuela.keys():
    print(key) ## En el primer ciclo imprime estudiantes, en el segundo ciclo imprime profesores
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Los diccionarios

En ocasiones es necesario iterar un diccionario de forma ordenada, para eso se puede utilizar la función sorted() en el diccionario y usando la función keys(). Ejemplo

```
escuela = {“estudiantes”: 1586, “profesores”: 14}

for name in sorted(escuela.keys()): # Imprimirá las keys en orden alfabético
    print(name)
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Los diccionarios

En el caso de solo querer los valores del diccionario y no las key se puede utilizar la función values() en el diccionario, este retorna una vista iterable con los valores.

```
escuela = {“estudiantes”: 1586, “profesores”: 14}

for name in escuela.values():
    print(name)
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Los diccionarios

Otra de las funciones para manejar diccionarios es el uso de la función set() esta permite eliminar los elementos duplicados.

```
participantes= {
   ‘rodrigo’ : ‘A’,
   ‘Marcelo’ : ‘B’
   ‘Juan’    : ‘A’
}

for name in set(participantes.values()):
    print(name)
# Imprime A, B
## El A como se repite no aparece 2 veces ya que la función set lo elimino
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Los diccionarios

También pueden crear una lista de diccionarios

```
lista = [{“nombre”: “Marcelo”, “puntos”: 1},{“nombre”: “Maria”, “puntos”: 5},{“nombre”: “Rodrigo”, “puntos”: 2}]
```

También pueden crear una lista dentro de un diccionario

```
lista = {“nombre”: “Marcelo”, “comidas_favoritas”: [“Puré”, “Arroz”]}
```

Otro cosa que se puede hacer es tener un diccionario dentro de un diccionario

```
lista = {
‘dormitorio’:{
    ‘cama’: “2 plazas”

    },
‘cocina’:{
    ‘lavalozas’: true,
    ‘esponja’: true
    }

}
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Ejercicio en clases

1- Crea un diccionario vacío

2- Agrega 2 pares de key-value(key-valor) con la información que estime conveniente

3- Elimina un par de key-value.

4- Crea un par de key-value donde el valor sea un lista.

5- Muestra por pantalla el valor de unos de los lista

6- Muestra por pantalla los valores de las keys del diccionario

7- Muestra por pantalla los valores del diccionario

8- Crea un par de key-value donde el valor sea un diccionario

9- Muestra por pantalla el valor de una de las par de key-value que está dentro del diccionario agregado al diccionario anteriormente

10- Crea un lista que contenga 2 diccionarios

11-Muestra por pantalla al valor de uno de los diccionario agregado a la lista anteriormente

Recordatorio
Valores vacio
```
test = {}
```
Eliminar un valor
```
del test['key']
```
Cambiar un valor
```
test['key'] = value
```
Agregar un valor
```
test['key'] = value
```
Agregar un lista al diccionario
```
test['key'] = [value1,value2,value3]
```
Agregar un diccionario a la lista
```
lista = [{'key':value, 'key2': value}]
```
Recorre un diccionario
```
for key, value in test:
```
Recorre el diccionario obteniendo los valores
```
for value in test.values():
```
Recorre el diccionario obteniendo las keys
```
for value in test.keys()
```

Introducción a los loops while

El bucle for toma una colección de elementos y ejecuta un bloque de código una vez para cada elemento de la colección. Por el contrario, el ciclo while se ejecuta mientras se cumpla una determinada condición. La estructura básica del while es la siguiente:

```
while condición:
    ejecuta mientras se cumpla condición
Cuando no cumple termina while
```

Un ejemplo más concreto es un while que cuente de 1 hasta 10

```
count = 1
while count <= 10:
    print(count)
    count+=1
# Imprime
# 1
# 2
# 3
# ...
# 9
# 10 Cuando count alcanza 10 al final del ciclo se le suma 1
# while verifica que count sea <=10 y como en este caso ahora es 11
# termina los ciclos while
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Introducción a los loops while

Una forma básica de detener el while es que la condición del ciclo while esté sujeta a una acción nuestra, como en el siguiente ejemplo:

```
mensaje= ‘’
while (mensaje!=‘salir’):
    mensaje = input(“Ingrese una palabra, si desea terminar escriba salir”)
    if(mensaje== ‘salir’):
        print(“El usuario a finalizado”)
    else:
        print(mensaje)
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Introducción a los loops while

Otro caso común para determinar cuándo terminar un ciclo while es a través del uso de banderas, las banderas normalmente son variables de tipo booleanas, al cumplir una determinada condición cambian su valor en este caso de true pasa false:

```
Ok = True
while Ok:
    mensaje = input(“Ingrese una palabra, si desea terminar escriba salir”)
    if(mensaje== ‘salir’):
        Ok = False
        print(“El usuario a finalizado”)
    else:
        print(mensaje)
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Introducción a los loops while

Otra forma de terminar un loop, es con el uso de la instrucción break, aunque esta solo se debe usar con precaución y solamente cuando se estime necesaria, ya que cuando se ejecuta termina inmediatamente el loop y no continua con lo que faltaba para terminar el bloque de código, un ejemplo es:

```
while True:
    mensaje = input(“Ingrese una palabra, si desea terminar escriba salir”)
    if(mensaje== ‘salir’):
        break
    print(mensaje)
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Introducción a los loops while

Existen otras instrucción que permiten saltar el ciclo actual, pero sin terminar el while, para esto se puede utilizar la función continúe, un ejemplo es el siguiente:

```
count = 0
while count <=10:
    count+=1
    if(count %2 == 0):
        continue
    print(count)
# Imprime
# 1
# 3
# 5
# 7
# 9
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.

Introducción a los loops while

Una forma fácil de recorrer una lista dentro de un while puede ser de la siguiente manera

```
numeros_impar = [1,3,5,7]
numeros = []
while numeros_impar :
    valor = numeros_impar.pop()
    numeros_impar.append(numeros_impar )

## Agrega los valores de la lista numeros_impar a la lista numeros
## While cuando detecta que la lista está vacía se considera falso
# pueden comprobarlo ejecutando
print(bool([]))
# Retornara falso
```

Fuente: Matthes, Eric. Python Crash Course: A Hands-on, Project-Based Introduction to Programming. 2016.