Clase2_Secuencias_ Funciones

March 20, 2023

1 Seminario de Lenguajes - Python

1.1 Cursada 2023

1.2 Clase 2: listas, tuplas y diccionarios. Introducción a funciones

2 En el video de cadenas planteamos el siguiente desafío:

> Escribir un programa que ingrese 4 palabras desde el teclado e imprima aquellas que contienen la letra r.

Una posible solución:

[ ]:
```
for i in range(4):
    cadena = input("Ingresá una palabra: ")
    if "r" in cadena:
        print(f"{cadena} tiene una letra r")
```

- ¿Se acuerdan qué representa **f"{cadena} tiene una letra r"**?

3 Desafío 1

> Vamos a modificar el código anterior para que se imprima la cadena "TIENE R" si la palabra contiene la letra r y sino, imprima "NO TIENE R".

[ ]:
```
for i in range(4):
    cadena = input("Ingresá una palabra: ")
    if "r" in cadena:
        print("TIENE R")
    else:
        print("NO TIENE R")
```

Hay otra forma de escribir estas expresiones condicionales.

4 Expresión condicional

- La forma general es:

```
A if C else B
```

- Devuelve A si se cumple la condición C, sino devuelve B.

[ ]:
```
x = int(input("Ingresá un número"))
y = int(input("Ingresá un número"))

maximo = x if x > y else y
maximo
```

5 Escribimos otra solución al desafío

[ ]:
```
for i in range(4):
    cadena = input("Ingresá una palabra")
    print("TIENE R" if "r" in cadena else "NO TIENE R")
```

6 Evaluación del condicional

IMPORTANTE: Python utiliza la **evaluación con circuito corto** para evaluar las condiciones

[ ]:
```
x = 1
y = 0

if True or x/y:
    print("No hay error!!!!")
```

7 Desafío 2

Ingresar palabras desde el teclado hasta ingresar la palabra FIN. Imprimir aquellas que empiecen y terminen con la misma letra. - ¿Qué estructura de control deberíamos utilizar para realizar esta iteración? ¿Podemos utilizar la sentencia **for**?

8 Iteración condicional

- Python tiene una sentencia **while**. ¿Se acuerdan de este ejemplo?

[ ]:
```
## Adivina adivinador....
import random
numero_aleatorio = random.randrange(5)
gane = False
print("Tenés 2 intentos para adivinar un entre 0 y 4")
intento = 1

while intento < 3 and not gane:
    numero_ingresado = int(input('Ingresa tu número: '))
    if numero_ingresado == numero_aleatorio:
        print("Ganaste!")
```

```
        gane = True
    else:
        print('Mmmm ... No.. ese número no es... Seguí intentando.')
        intento += 1
if not gane:
    print("Perdiste :(")
    print("El número era:", numero_aleatorio)
```

9 La sentencia while

```
while condicion:
    instrucción
    instruccion
```

Nuestro desafio: > Ingresar palabras desde el teclado hasta ingresar la palabra FIN. Imprimir aquellas que empiecen y terminen con la misma letra.

[ ]:
```
# Solución al desafío
```

10 Desafío 3

Queremos procesar la duración de las películas de Harry Potter.

Queremos saber: - cuál fue la duración, en minutos, promedio; - cuántas películas duran más que el promedio, en minutos.

¿Cómo sería un pseudocódigo de esto?

```
Ingresar la duración de las peliculas.
Calcular el promedio.
Calcular cuántas películas duran más que el promedio.
```

¿Cómo hacemos el tercer proceso? ¿Tenemos que ingresar las duraciones nuevamente?

Obviamente no. **Necesitamos tipos de datos que nos permitan guardar muchos valores**.

11 Listas

- Una **lista** es una colección ordenada de elementos.

[ ]:
```
duracion_pelis = [152, 161, 142, 157, 138, 153, 146, 130]

duracion_pelis
```

11.0.1 Las listas son estructuras heterogéneas, es decir que pueden contener cualquier tipo de datos, inclusive otras listas.

[ ]:
```
varios = [1, "dos", [3, "cuatro"], True]
varios
```

¿Cuántos elementos tiene la lista?

[ ]:
```
len(varios)
```

12 Accediendo a los elementos de una lista

- Se accede a través de **un índice** que indica la posición del elemento dentro de la lista **encerrado entre corchetes** [].
- **IMPORTANTE:** al igual que las cadenas los índices comienzan en 0.

[ ]:
```
varios = [ 17, "hola", [1, "dos"], 5.5, True]
print(varios[0])
print(varios[2][1] )
print(varios[-3])
```

- **Las listas son datos MUTABLES.** ¿Qué quiere decir esto?

[ ]:
```
varios[0] = ["El Dibu", 23]
varios
```

13 Recorriendo una lista

- ¿Qué estructura les parece que podríamos usar?

[ ]:
```
for elem in varios:
    print(elem)
```

13.1 Otra forma de recorrer:

[ ]:
```
# otra forma
cant_elementos = len(varios)
for indice in range(cant_elementos):
    print(varios[indice])
```

14 Retomemos el desafío

```
Ingresar la duración de las peliculas.
Calcular el promedio.
Calcular cuántas películas duran más que el promedio.
```

Empecemos con **el primer proceso**: vamos a suponer que ingresamos datos hasta que ingresemos una duración igual a 0 (esto si no sabemos cuántas películas vamos a procesar).

¿Cómo hacemos la iteración?

14.1 ¿Qué otra cosa nos falta?

[ ]:
```
lista = ["Leo", 10]
lista.append("Dibu")
lista.append(23)
lista
```

15 Ahora si resolvamos este proceso

[ ]:
```
#Ingresar las duraciones
cant_minutos = int(input("Ingresá la duración de una película de Harry Potter␣
 ↪(0 para finalizar)"))
duracion_pelis  = []
while cant_minutos != 0:
    duracion_pelis.append(cant_minutos)
    cant_minutos = int(input("Ingresá la duración de una película de Harry␣
 ↪Potter (0 para finalizar)"))

duracion_pelis
```

15.1 Calculamos el promedio

[ ]:
```
# Calculamos el promedio
suma = 0
for peli in duracion_pelis:
    suma += peli

promedio = suma / len (duracion_pelis)
promedio
```

[ ]:
```
#Otra solución

promedio = sum(duracion_pelis) / len(duracion_pelis)
promedio
```

¡IMPORTANTE! Acá no estamos chequeando que la lista esté vacía!!!

15.2 Tarea para el hogar: terminar el desafío

```
Datos:
Película                                    Duración (minutos)
Harry Potter y la piedra filosofal (2001) -152
```

```
Harry Potter y la cámara secreta (2002) - 161
Harry Potter y el prisionero de Azkaban (2004) -142
Harry Potter y el cáliz de fuego (2005) -157
Harry Potter y la Orden del Fénix (2007) -138
Harry Potter y el misterio del príncipe (2009)- 153
Harry Potter y las Reliquias de la Muerte - Parte 1 (2010)-146
Harry Potter y las Reliquias de la Muerte - Parte 2 (2011)-130
```

16 Slicing con listas

- Al igual que en el caso de las cadenas de caracteres, se puede obtener una porción de una lista usando el operador **":"**

[ ]:
```
vocales = [ "a", "e","i", "o", "u"]
print(vocales[1:3])
```

- Si no se pone inicio o fin, se toma por defecto las posiciones de inicio y fin de la lista.
- **Tarea para el hogar:** probar con índices negativos.

17 Asignación de listas

- Observemos el siguiente ejemplo:

[ ]:
```
rock = ["Riff", "La Renga", "La Torre"]
blues = ["La Mississippi", "Memphis"]

musica = rock

print(musica)
```

- Recordemos que las variables son referencias a objetos.
- Cada objeto tiene un identificación.

[ ]:
```
print(id(musica))
print(id(rock))
print(id(blues))
```

¿Cuál es el problema?

[ ]:
```
rock.append("Rata Blanca")
musica
```

18 Observemos este código

[ ]:
```
otra_musica = rock[:]
print(id(rock))
print(id(otra_musica))
print(id(musica))
```

[ ]:
```
otra_musica.append("Hermetica")
otra_musica
```

- **musica = rock**: musica y rock apunten al mismo objeto (misma zona de memoria).
- **otra_musica = rock[:]**: otra_musica y rock referencian a dos objetos distintos (dos zonas de memoria distintas con el mismo contenido).

Observemos este código:

[ ]:
```
# Otra forma
mas_musica = rock.copy()
id(mas_musica)
```

- ¿Qué podemos decir del método **copy**?

19 Operaciones con listas

- Las listas se pueden concatenar (+) y repetir (*)

[ ]:
```
rock = ["Riff", "La Renga", "La Torre"]
blues =["La Mississippi", "Memphis"]

musica = rock + blues
mas_rock = rock * 3
mas_rock
```

20 Algunas cosas para prestar atención

Analicemos el siguiente código:

[ ]:
```
lista = [[1,2]] * 3
lista
```

[ ]:
```
lista [0][1] = 'cambio'
lista
```

- ¿Qué es lo que sucedió?
  - El operador * repite la **misma lista**, no genera una copia distinta; es el mismo objeto referenciado 3 veces.

21 Probemos ahora:

[ ]:
```
lista = [[1,2], [1, 2], [1, 2]]
lista [0][1] = 'cambio'
lista
```

22 Probar en casa: más operaciones sobre listas

- Algunos métodos o funciones aplicables a listas: **extend()**, **index()**, **remove()**, **pop()**, **count()**.
- +Info en la documentación oficial.

23 ¿Se acuerdan de esta operación con cadenas?

[ ]:
```
palabras = "En esta clase aparecen grandes bandas".split(" ")
palabras
```

Veamos de qué tipo es palabras ...

[ ]:
```

```

24 Algo muy interesante: comprensión de listas

(**list comprehension**)

Observemos la siguiente definición: ¿cuáles serían los elementos de esta lista?

[ ]:
```
import string

letras = string.ascii_uppercase

codigos = [ord(n) for n in letras]
print(codigos)
```

¿Y en este otro caso?

[ ]:
```
cuadrados = [num**2 for num in range(10) if num % 2 == 0]
cuadrados
```

25 Volviendo al desafio 3

Queremos procesar las duraciones de las películas de Harry Potter.

Queremos saber: - cuál fue la duración, en minutos, promedio; - cuántas películas duran más que el promedio, en minutos.

Panteamos otra solución:

[ ]:
```
duracion_pelis = [152, 161, 142, 157, 138, 153, 146, 130]
promedio = sum(duracion_pelis) / len(duracion_pelis)
mayor_prom = [n for n in duracion_pelis if n > promedio]
mayor_prom
```

26 Desafío 4

Dada una alista de palabras, generar otra lista con aquellos verbos en infinitivo. (solo vamos a comprobar que terminen en "ar", "er" o "ir")

[ ]:
```
palabras = ["casa", "ir", "sol", "cantar","correr"]
verbos = [pal for pal in palabras if pal.endswith(("ar", "er", "ir"))]
verbos
```

27 Tuplas: otro tipo de secuencias en Python

- Al igual que las listas, son colecciones de datos ordenados.

[ ]:
```
tupla = 1, 2
tupla1 = (1, 2)
tupla2 = (1,)  # OJO con esto
tupla3 = ()
type(tupla2)
```

Observemos detallamente el código del desafío 4

[ ]:
```
palabras = ["casa", "ir", "sol", "cantar","correr"]
verbos = [pal for pal in palabras if pal.endswith(("ar", "er", "ir"))]
verbos
```

En este caso, estamos pasando una tupla como argumento al método **endswith**.

27.1 ¿Cuál es la diferencia con las listas?

Veamos las siguientes situaciones.

28 Tuplas vs. listas

[ ]:
```
tupla = (1, 2)
lista = [1, 2]

elem = tupla[0]
elem
#print(len(tupla))
```

- Se acceden a los elementos de igual manera: usando [] (empezando desde cero)

- La función **len** retorna la cantidad de elementos en ambos casos.

¿Entonces?

28.1 DIFERENCIA: las tuplas son INMUTABLES

- Su tamaño y los valores de las mismas NO pueden cambiar.

[ ]:
```
tupla = (1, 2)
lista = [1, 2]

#tupla[0] = "uno" # Esto da error
tupla
#tupla.append("algo") # Esto da error
```

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

28.1.1 Tenemos que acostumbrarnos a leer los errores.

29 Obteniendo subtuplas

[ ]:
```
tupla = (1, 2, 3, "hola")
print(tupla[1:4])
nueva_tupla = ("nueva",) + tupla[:2]
print(nueva_tupla)
```

[ ]:
```
# ¿por qué da error este código?
nueva_tupla = ('nueva') + tupla[1:3]
nueva_tupla
```

30 Desafío 5

Volvemos a procesar las películas de Harry Potter.

Queremos saber: - cuál fue la duración, en minutos, promedio; y - **qué** películas duran más que el promedio, en minutos.

¿Qué diferencia hay con el desafío 3?

- Deberíamos ingresar no sólo las duraciones, sino también los nombres de las películas.
- ¿Qué soluciones proponen?

31 ¿Qué les parece esta solución?

[ ]:
```
peli = input("Ingresa el nombre de una película de Harry Potter (<FIN> para␣
 ↪finalizar)")
lista = []
while peli != "FIN":
```

```
    cant_minutos = int(input(f"Ingresa la duración de la película {peli}"))
    lista.append((peli, cant_minutos))
    peli = input("Ingresa el nombre de una película de Harry Potter (<FIN> para␣
 ↪finalizar)")
lista
```

- ¿Qué estructura de datos estoy usando?
- Hay algo mejor...

32 Diccionarios en Python

- Un diccionario es un conjunto **no ordenado** de pares de datos: **clave:valor**.
- Se definen con **{ }**.

[ ]:
```
pelis = {"Harry Potter y la piedra filosofal (2001)":152,
         "Harry Potter y la cámara secreta (2002)": 161,
         "Harry Potter y el prisionero de Azkaban (2004)": 142,
         "Harry Potter y el cáliz de fuego (2005)": 157,
         "Harry Potter y la Orden del Fénix (2007)":138,
         "Harry Potter y el misterio del príncipe (2009)":153,
         "Harry Potter y las Reliquias de la Muerte - Parte 1 (2010)":146,
         "Harry Potter y las Reliquias de la Muerte - Parte 2 (2011)":130
        }
```

[ ]:
```
pelis["Harry Potter y la piedra filosofal (2001)"]
```

33 Las claves deben ser únicas e inmutables

- Las **claves** pueden ser cualquier tipo **inmutable**.
  - Las cadenas y números siempre pueden ser claves.
  - Las tuplas se pueden usar sólo si no tienen objetos mutables.

[ ]:
```
# Probar en casa cuáles de las siguientes instrucciones dan error

#dicci = {"uno":1}
#dicci = {1: "uno"}
#dicci = {[1,2]: "lista"}
#dicci = {(1,2): "tupla"}
#dicci = {([1],2): "tupla"}
#dicci
```

34 ¿Cómo accedemos a los elementos?

- Al igual que las listas y tuplas, se accede usando [ ] pero en vez de un índice que representa la posición, usamos la clave.
- Es un error extraer un valor usando una clave no existente.

[ ]:
```
meses = {"enero": 31, "febrero": 28, "marzo": 31}
meses
```

35 ¿Cómo agregamos elementos?

- Si se usa una clave que ya está en uso para guardar un valor, el valor que estaba asociado con esa clave se pierde, si no está la clave, se agrega.

[ ]:
```
meses["febrero"] = 29
meses["abril"] = 30
meses
```

36 Volviendo al desafío planteado ...

- Nos falta saber cómo definir un diccionario vacío para luego ir agregando los valores.

[ ]:
```
peli = input("Ingresa el nombre de una película de Harry Potter nombre (<FIN>␣
 ↪para finalizar)")
dicci = {}
while peli != "FIN":
    cant_minutos = int(input(f"Ingresa la duración de la película  {peli}"))
    dicci[peli] = cant_minutos
    peli = input("Ingresa el nombre de una película de Harry Potter nombre␣
 ↪(<FIN> para finalizar)")
dicci
```

37 ¿Cómo recorremos un diccionario?

[ ]:
```
musica = {"rock": ["Riff", "La Renga", "La Torre"],
          "blues": ["La Mississippi", "Memphis", "violeta"]}


for elem in musica:
    print(elem)
```

¿A qué referencia la variable **elem**?

Si queremos mostrar los valores:

[ ]:
```
musica = {"rock": ["Riff", "La Renga", "La Torre"],
          "blues": ["La Mississippi", "Memphis", "violeta"]}

for elem in musica:
    print(musica[elem])
```

38 Existen algunos métodos útiles:

```
[ ]: claves = musica.keys()
     valores = musica.values()
     items = musica.items()
```

```
[ ]: for elem in claves:
         print(elem)
```

39 El operador in en diccionarios

```
[ ]: musica = {"rock": ["Riff", "La Renga", "La Torre"],
               "blues": ["La Mississippi", "Memphis", "violeta"]}
     "Riff" in musica
```

- ¿Verifica en las claves o los valores?

40 Observemos este código

```
[ ]: meses = {"enero": 31, "febrero": 28, "marzo": 31}
     meses1 = meses
     meses2 = meses.copy()
     print(id(meses))
     print(id(meses1))
     print(id(meses2))
```

¿Qué significa?

```
[ ]: meses1["abril"] = 30
     meses2["abril"] = 43
     meses
```

41 Más operaciones

Probar en casa:

- **del**: permite borrar un par clave:valor
- **clear()**: permite borrar todo

```
[ ]: # Probamos del y clear
```

42 Otra forma de crear diccionarios

- Podemos usar **dict()**.

- Se denomia “constructor” y crea un diccionario directamente desde una secuencia de pares clave-valor.

[ ]:
```
dicci = dict([("enero", 31), ("febrero", 28), ("marzo", 31)])
dicci
```

- ¿Qué tipos de datos se pueden usar para la secuencia?
- ¿Y para los pares clave-valor?

43 Por comprensión

[ ]:
```
dict ([(x, x**2) for x in (2, 4, 6)])
```

[ ]:
```
import string

caracteres = dict([(n, ord(n)) for n in string.digits])
caracteres
```

44 Funciones en Python: una forma de definir procesos o subprogramas

Veamos un pseudocódigo de la solución del **desafío 5**:

```
Ingresar los nombres y duración  de las películas.
Calcular el promedio.
Mostrar qué  películas duran más que el promedio.
```

Podríamos pensar en dividir en tres procesos:

- En Python, usamos **funciones** para definir estos procesos.
- Las funciones pueden recibir **parámetros**.
- Y también retornan siempre un valor. Esto puede hacerse en forma implícita o explícita usando la sentencia **return**.

45 Ya usamos funciones

- float(), int(), str()
- len(), ord()
- input(), print()

En estos ejemplos, sólo **invocamos** a las funciones ya definidas.

46 Podemos definir nuestras propias funciones

```
def nombre_funcion(parametros):
    sentencias
    return <expresion>
```

- **IMPORTANTE:** el cuerpo de la función debe estar **indentado**.

47 La función para el primer proceso del desafío

[ ]:
```
def ingreso_pelis():
    """ Esta función retorna un diccionario con los nombres y duración de␣
 ↪películas """

    peli = input("Ingresa el nombre de una película de Harry Potter nombre␣
 ↪(<FIN> para finalizar)")
    dicci = {}
    while peli != "FIN":
        cant_minutos = int(input(f"Ingresa la duración de la película  {peli}"))
        dicci[peli] = cant_minutos
        peli = input("Ingresa el nombre de una película de Harry Potter nombre␣
 ↪(<FIN> para finalizar)")
    return dicci

pelis = ingreso_pelis()
pelis
```

- **¡IMPORTANTE!** Definición vs. invocación.

47.1 ¿Qué pasa si no incluyo el return?

[ ]:
```
def demo_funcion_sin_return():
    var = 10
    print(var)
demo_funcion_sin_return()
```

48 Tarea para el hogar: definir las otras dos funciones para completar el desafio

- Los que quieran, pueden subir el código a su repositorio en GitHub y compartir el enlace a la cuenta **@clauBanchoff**

49 Un artículo sobre las reglas de estilo

.

How to Write Beautiful Python Code With PEP 8

50 Seguimos la próxima ...