Clase02_0Listas_ Funciones_modulos

March 18, 2024

1 Seminario de Lenguajes - Python

1.1 Cursada 2024

1.2 Clase 2: listas, funciones y módulos

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):
         word = input("Ingresá una palabra: ")
         if "r" in word:
             print(f"{word} tiene una letra r")
```

- ¿Se acuerdan qué representa **f"{word} 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".

La solución casi immediata:

```
[ ]: for i in range(4):
         word = input("Ingresá una palabra: ")
         if "r" in word:
             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.

[ ]:
```
number1 = int(input("Ingresá un número: "))
number2 = int(input("Ingresá un número: "))

number_max = number1 if number1 > number2 else number2
number_max
```

5 Escribimos otra solución al desafío

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

6 Evaluación del condicional

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

[ ]:
```
number1 = 1
number2 = 0

if True or number1/number2:
    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

random_number = random.randrange(5)
i_won = False
print("Tenés 2 intentos para adivinar un entre 0 y 4")
tries = 1

while tries < 3 and not i_won:
    entered_number = int(input("Ingresa tu número: "))
```

```
    if entered_number == random_number:
        print("Ganaste!")
        i_won = True
    else:
        print("Mmmm ... No.. ese número no es... Seguí intentando.")
        tries += 1
if not i_won:
    print("Perdiste :(")
    print(f"El número era: {random_number}")
```

9 La sentencia while

```
while condition:
    sentence
    sentence
```

Nuestro desafío: > 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

En homenaje a Akira Toriyama, vamos a trabajar con las películas de Dragon Ball.

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

¿Cómo sería un pseudocódigo de esto?

```
Ingresar la duración de las películas.
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.

[ ]:
```
movies_duration = [50, 48, 85, 93, 100]

type(movies_duration)
```

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

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

¿Cuántos elementos tiene la lista?

[ ]:
```
len(my_list)
```

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.

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

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

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

13 Recorriendo una lista

- ¿Qué estructura les parece que podríamos usar?

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

13.1 Pdríamos pensar en otra forma de recorrer la lista

[ ]:
```
list_length = len(my_list)
for i in range(list_length):
    print(my_list[i])
```

¿Qué forma les parece más legible?

14 Asignación de listas

Observemos el siguiente ejemplo:

[ ]:
```
rock = ["Riff", "La Renga", "La Torre"]
blues = ["La Mississippi", "Memphis"]
music = rock
print(music)
```

- Recordemos que las variables son **referencias a objetos**.

¿Cuál es el problema?

- Cada objeto tiene un identificación.

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

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

15 Observemos este código

[ ]:
```
more_music = rock[:]
print(id(rock))
print(id(more_music))
print(id(music))
```

[ ]:
```
more_music.append("Hermetica")
rock
```

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

Observemos este código:

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

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

16 Operaciones con listas

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

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

music = rock + blues
more_rock = rock * 3
```

[ ]:
```
more_rock
```

17 Algunas cosas para prestar atención

Analicemos el siguiente código:

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

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

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

18 Probemos ahora:

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

19 Probar en casa: más operaciones sobre listas

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

20 Algo muy interesante: comprensión de listas

(**list comprehension**)

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

[ ]:
```
import string

letters = string.ascii_uppercase
codes = [ord(n) for n in letters]
print(codes)
```

¿Y en este otro caso?

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

21 Retomamos a las pelis de Dragon Ball

Nuestro desafío era:

Queremos procesar las duraciones de las películas de Dragon Ball.

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

Ahora podemos plantear otra solución:

[ ]:
```
movies_duration = [50, 48, 85, 93, 100]
average = sum(movies_duration) / len(movies_duration)
max_average = [n for n in movies_duration if n > average]
max_average
```

22 DESAFÍO 4

Dada una lista de palabras, generar otra lista con aquellos verbos en infinitivo.

IMPORTANTE: solo vamos a comprobar que terminen en “ar”, “er” o “ir”.

[ ]:
```
words = ["c", "ir", "sol", "cantar","correr"]
verbs = [pal for pal in words if pal.endswith(("ar", "er", "ir"))]
verbs
```

23 Retomemos el desafío

```
Ingresar la duración de las películas.
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?

23.1 ¿Qué otra cosa nos falta?

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

24 Ahora si resolvamos este proceso

[ ]:
```
#Ingresar las duraciones
minutes = int(input("Ingresá la duración de una película de Dragon Ball (0 para␣
 ↪finalizar)"))
movies_duration  = []
while minutes != 0:
    movies_duration .append(minutes)
```

```
    minutes = int(input("Ingresá la duración de una película de Dragon Ball (0␣
 ↪para finalizar)"))

movies_duration
```

24.1 Calculamos el promedio

[ ]:
```
# Calculamos el promedio
total = 0
for movie in movies_duration:
    total += movie

average = total / len (movies_duration)
average
```

[ ]:
```
#Otra solución

average = sum(movies_duration) / len(movies_duration)
average
```

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

24.2 Tarea para el hogar: terminar el DESAFÍO 3

Datos según ChatGPT:

```
Película,Duración (minutos)
Dragon Ball: La leyenda de Shenron,50
Dragon Ball Z: Los guerreros de plata,48
Dragon Ball Z: La batalla de los dioses,85
Dragon Ball Z: La resurrección de Freezer,93
Dragon Ball Super: Broly,100
```

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

Veamos un pseudocódigo de la solución del **DESAFÍO 3**:

```
Ingresar la duración de las películas.
Calcular el promedio.
Calcular cuántas 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**.

26 Ya usamos funciones

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

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

27 Podemos definir nuestras propias funciones

```
def my_function(param1, param2):
    sentences
    return <expression>
```

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

28 Una posible función para el primer proceso del desafío:

[ ]:
```
def input_movies():
    """ This function returns a list with the duration in minutes of the movies␣
↪"""

    minutes = int(input("Ingresá la duración de una película de Dragon Ball (0␣
↪para finalizar)"))
    movies_duration  = []
    while minutes != 0:
        movies_duration .append(minutes)
        minutes = int(input("Ingresá la duración de una película de Dragon Ball␣
↪(0 para finalizar)"))

    return movies_duration
```

[ ]:
```
movies = input_movies()
movies
```

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

29 El docstring

- Es una secuencia de caracteres que describe la función.
- Se sugiere siempre utilizar triples ”.
- **Son procesados por el intérprete.**

[ ]:
```
#help(print)
help(input_movies)
```

29.1 ¿Qué pasa si no incluyo el return?

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

print(function_without_return())
```

29.2 Ahora observemos este código que implementa una posible solución al segundo proceso

[ ]:
```
def average_calculation(movies_duration):
    """ This function calculates the average of the lengths of the movies␣
 ↪received by parameter.
    movies_duration: is a list with the duration in minutes of the movies
    """
    len_movies = len(movies_duration)
    average = 0 if len_movies == 0 else sum(movies_duration) / len_movies
    return average
```

[ ]:
```
average_calculation(movies)
```

- A diferencia de la función anterior, ésta tiene un parámetro.
- ¿Hay distintas formas de pasar parámetros en Python? ¿Cómo podemos probar esto?

30 DESAFÍO 5: tarea para el hogar

- Definir la función restante para completar el desafío.
- Escribir un programa que permita probar de qué forma se pasan los parámetros en Python.

  PISTA: probar con una función que reciba un entero y otra que reciba una lista.

31 ¿Qué son los módulos en Python?

#

Un módulo es un archivo (con extensión .py) que contiene sentencias y definiciones

#####

¿Algo nuevo?

32 Analicemos un ejemplo en el IDE

- ¿Cuántos archivos forman mi programa?
- ¿Cómo se relacionan?

33 La sentencia import

- Permite acceder a funciones y variables definidas en otro módulo.

[ ]:
```
import random
```

- Si el módulo contiene definiciones de funciones, sólo se importa estas definiciones, no las ejecuta.
- Para ejecutar una función definida en otro módulo **debo invocarla en forma explícita**.

[ ]:
```
random.randrange(10)
```

34 Otra forma de importar

```
from my_module import my_function
```

- Sólo se importa **my__function** de **my__module** (no el nombre del módulo).

```
from my_module import *
```

- En este caso, importamos **todos los ítems** definidos en **my__module**.
- **Esta forma no está recomendada**: podrían existir conflictos de nombres. ¿Por qué?

[ ]:
```
from random import choice
numbers = [x ** 2 for x in range(1, 15, 2)]
choice(numbers)
```

[ ]:
```
choice = 10
choice
```

35 Importando módulos

```
import utils

utils.one()
```

- La importación se realiza **sólo una vez por sesión del intérprete**.
- Veamos sobre el ejemplo anterior.
- Si necesito volver a importar podemos usar **reload()** incluido en el **módulo importlib**.

```
import importlib
importlib.reload(utils)
```

- +Info en la documentación oficial

35.0.1 Veamos qué pasa cuando queremos importar un módulo que no existe:

```
[ ]: import pp
```

35.0.2 ¿Dónde se busca los módulos?

- Directorio actual + otros directorios definidos en la variable de ambiente PYTHONPATH

```
[ ]: import sys
     sys.path
```

36 Los nombres de los módulos

- Es posible acceder al nombre de un módulo a través de **___name___**

```
[ ]: print(__name__)
```

36.1 ¿Cuándo un módulo se denomina ___main___?

- Las instrucciones ejecutadas en el nivel de llamadas superior del intérprete, ya sea desde un script o interactivamente, se consideran parte del módulo llamado **___main___**.

```
#módulo utils
print(f"El nombre de este módulo es {__name__}")

if __name__ == "__main__":
    one()
```

37 Agreguemos un menú al ejemplo anterior:

38 Primero: ¿qué hace?

39 Segundo: ¿en qué casos se muestra el menú?

¿Es posible utilizar las funciones sin invocar el menú?

```
[ ]: if __name__ == "__main__":
        print(""" Elegí una opción:
        (1) one()
        (2( two()
        (3) three()
        """)
        option = int(input())
        match option:
            case 1:
                one()
            case 2:
```

```
            two()
        case 3:
            three()
```

40 Ahora: ¿en qué casos se muestra el menú?

41 DESAFÍO 6: tarea para el hogar

Reimplementar el desafío completo definiendo las funciones en un módulo pensando que las funciones se podrían invocar desde el módulo donde están definidas o desde otros módulos.

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

42 Seguimos la próxima ...