Clase2

March 23, 2021

1 Seminario de Lenguajes - Python

1.1 Cursada 2021

1.2 Clase 2

2 Un repaso antes de empezar

3 Ahora si, iniciamos la clase 2...

4 En el video de previa a esta clase planteamos el siguiente desafío:

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

[ ]:
```
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"**?

5 Primer desafío del día

5.0.1 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 expresiones condicionales.

6 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
```

7 Escribimos otra solución al desafío

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

8 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!!!!")
```

9 Segundo desafío del día

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

- ¿Podemos resolver esto con lo visto hasta el momento? ¿Qué estructura de control deberíamos utilizar para realizar esta iteración?

10 Iteración condicional

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

[ ]:
```
#Adivina adivinador....
import random

numero_aleatorio = random.randrange(5)
```

```python
gane = False

print("Tenés 5 intentos para adivinar un entre 0 y 99")
intento = 1

while intento < 6 and not gane:
    numero_ingresado = int(input('Ingresa tu número: '))
    if numero_ingresado == numero_aleatorio:
        print(f'Ganaste! y necesitaste {intento} intentos!!!')
        gane = True
    else:
        print('Mmmm ... No.. ese número no es... Seguí intentando.')
        intento += 1
if not gane:
    print(f'\n Perdiste :(\n El número era: {numero_aleatorio}')
```

11 La sentencia while

```
while condicion:
    instrucción
    instruccion
```

[ ]:
```python
# Solución al desafío
```

12 Tercer desafío del día

12.0.1 Necesitamos procesar las notas de los estudiantes de este curso. Queremos saber:

- cuál es el promedio de las notas;
- cuántos estudiantes están por debajo del promedio.

¿Cómo sería un pseudocódigo de esto?

```
Ingresar las notas
Calcular el promedio
Calcular cuántos tienen notas menores al promedio
```

¿Cómo obtenemos las notas menores al promedio? ¿Tenemos que ingresar las notas de nuevo?

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

13 Listas

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

[ ]:
```python
notas = [ 4, 6, 7, 3, 8, 1, 10, 4]
```

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

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

¿Cuántos elementos tiene la lista?

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

14 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] = 27
varios
```

15 Recorriendo una lista

- ¿Qué estructura les parece que podríamos usar?

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

15.1 Otra forma de recorrer:

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

16 Retomemos el desafío

```
Ingresar las notas
Calcular el promedio
Calcular cuántos tienen notas menores al promedio
```

Empecemos con el primer proceso: vamos a suponer que ingresamos datos hasta que ingrese una nota -1 - ¿Qué otra cosa nos falta?

[ ]:
```
lista = []
lista.append("algo")
lista.append("otro")
lista
```

17 Ahora si resolvamos este proceso

[ ]:
```
#Ingresar las notas
nota = int(input("Ingresá una nota (-1 para finalizar)"))
lista_de_notas = []
while nota != -1:
    lista_de_notas.append(nota)
    nota = int(input("Ingresá una nota (-1 para finalizar)"))
lista_de_notas
```

17.1 Tarea para el hogar: terminar el desafío

18 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 **":"**

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

- Si no se pone inicio o fin, se toma por defecto las posiciones de inicio y fin de la lista.

[ ]:
```
print( varios[ :2] )
print( varios[2:] )
```

19 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))
```

20 Observemos este código

[ ]:
```
otra_musica = rock[:]
print(id(rock))
print(id(otra_musica))
print(id(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).
- **Probar:** mas_musica = rock.copy()

21 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
musica
```

22 Algunas cosas para prestar atención

Analicemos el siguiente código:

[ ]:
```
lista =  [[1 , 2]] * 3
print (lista)
lista [0][1] = 'cambio'
print (lista)
```

- ¿Qué es lo que sucedió?
  - El operador * repite la misma lista, no genera una copia distinta; es el mismo objeto referenciado 3 veces.
- **Probar:** reemplazar la definición de la lista original con lista = [[1,2], [1, 2], [1, 2]]

23 Ahora analicemos este otro ejemplo:

[ ]:
```
lista = [[1,2], 8, 9]
lista2 = lista.copy()
print (lista, lista2)

lista[0][1]= 'cambio1'
lista[2]='cambia2'
print (lista)
print (lista2)
```

¿Qué sucedió?

24 Probar en casa: más operaciones sobre listas

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

25 ¿Se acuerdan de esta operación con cadenas?

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

Veamos de qué tipo es palabras ...

26 Algo muy interesante

26.1 Listas por comprensión

Observemos cómo definimos esta lista: ¿cuáles serían los elementos de esta lista?

[ ]:
```
import string

digitos = string.digits
codigos = [ord(n) for n in digitos]
codigos
```

¿Y en este otro caso?

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

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(tupla3)
```

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]
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.

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.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[1:3]
print(nueva_tupla)
```

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

30 Cuarto desafío del día

30.0.1 Necesitamos procesar las notas de los estudiantes de este curso. Queremos saber:

```
- cuál es el promedio de las notas
- qué estudiantes están por debajo del promedio.
```

¿Qué diferencia hay con el desafío anterior?

- Deberíamos ingresar no sólo las notas, sino también los nombres.
- ¿Qué soluciones proponen?
- ¿Qué les parece esta solución?

[ ]:
```
nombre = input("Ingresa un nombre (<FIN> para finalizar)")
lista = []
while nombre != "FIN":
    nota = int(input(f"Ingresa la nota de {nombre}"))
    lista.append((nombre, nota))
    nombre = input("Ingresa un nombre (<FIN> para finalizar)")
lista
```

- Hay algo mejor...

31 Diccionarios en Python

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

[ ]:
```
notas = {"Janis Joplin":10, "Elvis Presley": 9, "Bob Marley": 5, "Tina Turner":␣
↪7}
notas
```

32 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 cuáles de las siguientes instrucciones dan error

dicci1 = {"uno":1}
dicci2 = {1: "uno"}
dicci3 = {[1,2]: "lista"}
dicci4 = {(1,2): "tupla"}
```

33 ¿Cómo accedemos a los elementos?

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

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

34 ¿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 = {"enero": 31, "febrero": 28, "marzo": 31}
meses["febrero"] = 29
meses["abril"] = 30
meses
```

35 Volviendo al desafío planteado ...

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

[ ]:
```
nombre = input("Ingresa un nombre (<FIN> para finalizar)")
dicci = {}
while nombre != "FIN":
    nota = int(input(f"Ingresa la nota de {nombre}"))
    dicci[nombre] = nota
    nombre = input("Ingresa un nombre (<FIN> para finalizar)")
dicci
```

36 ¿Cómo recorremos un diccionario?

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

#las claves
for elem in musica:
    print(elem)

# los valores
for elem in musica:
    print(musica[elem])
```

Existen algunos métodos útiles:

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

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

37 El operador in en diccionarios

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

- ¿Verifica en las claves o los valores?

38 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
meses2
```

39 Más operaciones

Probar en casa:

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

40 Otra forma de crear diccionarios

- Podemos usar **dict()**.
- Se denomia "constructor" y crea un diccionario directamente desde listas de pares clave-valor guardados como tuplas.

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

41 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
```

42 Modularizando nuestros programas

42.1 Definiendo funciones

Retomemos el pseudocódigo de la solución del tercer desafío:

```
Ingresar las notas
Calcular el promedio
Calcular cuántos tienen notas menores al 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**.

43 Ya usamos funciones

- float(), int(), str().
- len(), ord()
- range(), randrange()

44 Podemos definir nuestras propias funciones

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

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

45 La función para el primer proceso del desafío

[ ]:
```
def ingreso_notas():
    """ Esta función retorna un diccionario con los nombres y notas de␣
 ↪estudiantes """

    nombre = input("Ingresa un nombre (<FIN> para finalizar)")
    dicci = {}
    while nombre != "FIN":
        nota = int(input(f"Ingresa la nota de {nombre}"))
        dicci[nombre] = nota
        nombre = input("Ingresa un nombre (<FIN> para finalizar)")
    return dicci

ingreso_notas()
```

- Definición vs. invocación.
- ¿Qué pasa si no incluyo el **return**?

46 Tarea para el hogar

- Observar el juego del ahorcado que se presenta en el libro Invent Your Own Computer Games with Python.
- Prestar atención a:
  - Tipos de datos trabajados.
  - Funciones definidas
  - ¿Cómo define los niveles?
  - ¿Se respeta la PEP 8?
- ¿Se animan a modificarlo?
  - Agregar pistas sobre el tipo de la palabra a adivinar.

46.0.1 Subir el código modificado a su repositorio en GitHub.

- Compartir el enlace a la cuenta @clauBanchoff