Previa_clase2

March 19, 2021

1 Seminario de Lenguajes - Python

1.1 Cursada 2021

1.1.1 Previa clase 2

2 Hagamos un repaso de la primera clase

[ ]:
```
## Adivina adivinador....
import random
numero_aleatorio = random.randrange(5)
gane = False

print("Tenés 3 intentos para adivinar un entre 0 y 99")
intento = 1

while intento < 4 and not gane:
    numero_ingresado = int(input('Ingresa tu número: '))
    if numero_ingresado == numero_aleatorio:
        print('Ganaste! y necesitaste {} intentos!!!'.format(intento))
        gane = True
    else:
        print('Mmmm ... No.. ese número no es... Seguí intentando.')
        intento += 1
if not gane:
    print('\n Perdiste :(\n El número era: {}'.format(numero_aleatorio))
```

3 ¿Qué tipos de datos vimos?

- Números: **int** y **float**
- Booleanos: **bool** (que mencinamos que eran números también)
- Cadenas de caracteres: **str**

La función **type()** nos permite saber de qué tipo es un determinado objeto referenciado por una variable.

[ ]:
```
x = 1
print(type(x))
```

```
x
```

4 También vimos que hay algunas conversiones de tipo implícitas y otras explícitas

[ ]:
```
num = 10
mitad = num / 2
type(mitad)
mitad
```

[ ]:
```
num = 10
mitad = int(num / 2)
type(mitad)
```

5 Las cadenas de caracteres

- Secuencia de caracteres encerrados entre comillas simples ' ' o comillas dobles " ".
- También se pueden definir con """ """.

[ ]:
```
mensaje_de_error = "ATENCION: la opción ingresada no es correcta."
menu = """ Menú de opciones:
            1.- Jugar
            2.- Configurar el juego
            3.- Salir
        """
print(menu)
print(mensaje_de_error)
```

6 Operaciones con cadenas de caracteres

- Concatenación: +
- Repetición: *
- Longitud de la cadena: **len()**

[ ]:
```
cadena = "Python "
otra_cadena = "es lo más!"
print(cadena + otra_cadena)
print(cadena * 5)
print(len(cadena))
```

7 Algo más sobre cadenas de caracteres

- Cada elemento de la cadena se accede mediante un índice entre []

[ ]:
```
cadena = "Python"
cadena[0]
```

- El índice puede ser negativo:

[ ]:
```
cadena[-2]
```

8 Subcadenas

[ ]:
```
cadena[1:3]
cadena[3: -1]
```

- El operador **:** permite obtener subcadenas. Esto se denomina **slicing**.
- El formato es **cadena[inicio:fin]**
- NO incluye al elemento cuyo índice es **fin**.
- [**:**] devuelve toda la cadena.
- Si los índices son negativos, se recorre de derecha a izquierda.

9 Probemos esto:

[ ]:
```
cadena[1] = 'm'
```

- Las cadenas son INMUTABLES.

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

9.0.1 Tenemos que acostumbrarnos a leer los errores.

10 Algo más sobre cadenas de caracteres

- Ya mencionamos que en Python, todos son objetos.
- Si bien retornaremos a esto más adelante, podemos mencionar que los objetos tienen propiedades y métodos.
- Volviendo a las cadenas, algunos métodos que podemos utilizar son:

[ ]:
```
cadena = "Python es lo más!"
cadena.upper()
cadena.lower()
cadena.islower()
cadena.isupper()
cadena.count("s")
cadena.split(" ")
```

- +Info

11 El operador in

- Este operador retorna True o False de acuerdo a si un elemento está en una colección o no.
- Las cadenas de caracteres son **secuencias de caracteres** por lo que puede utilizarse este operador.

[ ]:
```
palabra = input("Ingresá una palabra")
if "a" in palabra:
    print("Hay letras a")
else:
    print("No hay letras a ")
```

12 El módulo string

- Python tiene un módulo denominado string que contiene mucha funcionalidad para la manipulación de cadenas.
- Hay que usar la sentencia **import** previamente. Esto lo veremos en detalle más adelante.

[ ]:
```
import string
letras = string.ascii_letters
minusculas = string.ascii_lowercase
digitos = string.digits

digitos
```

13 Un desafío

- Escribir un programa que ingrese 4 palabras desde el teclado e imprima aquellas que contienen la letra "r".
- **Pensar:** ¿podemos usar la instrucción **for** tal cual la vimos la clase pasada?
- La sentencia **for** permite iterar sobre una **secuencia**.

```
for variable in secuencia:
        instrucción
        instrucción
        ...
        instrucción
```

[ ]:
```
cadena = "0123"
for elem in cadena:
    print(elem)
```

- Para nuestro problema, debemos generar una secuencia de 4 elementos: podrían ser de cualquier tipo, aunque, ¿cuál sería la mejor opción?

14 Alguien podría pensar en plantear esto:

[ ]:
```
for i in "1234":
    cadena = input("Ingresá una palabra")
    if "r" in cadena:
        print(cadena)
```

Pero.. ¿sería lo correcto? ¿Qué pasa si queremos ingresar 200 palabras?

15 La función range()

- Esta función devuelve una secuencia de números enteros.
- Puede tener de 1 a 3 argumentos:

`range(valor_inicial, valor_final, paso)`

- Es posible invocarla con uno, dos o los tres argumentos.

[ ]:
```
for i in range(3, 14, 3):
    print(i)
```

16 Entonces, una mejor forma sería:

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

17 ¿Qué nos devuelve range()?

[ ]:
```
secuencia = range(3)
secuencia
```

Buscar: ¿de qué tipo es secuencia?

18 Cadenas con formato

- Es posible definir cadenas con determinados formatos utilizando el método **format**.
- La forma general es:

`cadena.format(agumentos)`

- Observemos los siguientes ejemplos:

[ ]:
```
intento = 3
```

```
print('Hola {} !!! Ganaste! y necesitaste {} intentos!!!'.format("claudia",␣
↪intento))
```

[ ]:
```
for x in range(5):
    print("{0:2d} {1:3d} {2:4d}".format(x, x*x, x*x*x))
```

19 Los f-String

- Fueron introducidos a partir de la versión 3.6.
- Ver la PEP 498
- +Info en la documentación oficial
- Una forma más sencilla de usar el format:

[ ]:
```
intento = 3
nombre = "claudia"
print(f'Hola {nombre} !!! Ganaste! y necesitaste {intento} intentos!!!')
x = 4
print(f"{x:2d} {x*x:3d} {x*x*x:4d}")
```

20 Algunas cosas interesantes

[ ]:
```
cad1 = "Cadena alineada a izquierda"
cad2 = "Cadena alineada a derecha"
cad3 = "Cadena centrada"
print(f"\n{cad1:<30}\n{cad2:>30}")
print(f"\n{cad3:^30}")
print(f"\n{cad3:*^30}")
```

21 Un artículo sobre sistemas de codificación

-Unicode & Character Encodings in Python: A Painless Guide