Estructuras de datos y funciones

Funciones y variables (parte II)

{desafío} latam_

Utilizar estructuras de datos apropiadas para la elaboración de un algoritmo que resuelve un problema acorde al lenguaje Python.

Codificar un programa utilizando funciones para la reutilización de código acorde al lenguaje Python.

- Unidad 1:
  Introducción a Python
- Unidad 2:
  Sentencias condicionales e iterativas
- Unidad 3:
  Estructuras de datos y funciones

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Analizaremos tipos de argumentos y variables, considerando funciones como argumentos, variables locales y globales*

¿Has tenido problemas al querer cambiar el valor de una variable?

/* Tipos de argumentos y parámetros */

Tipos de argumentos

Como revisamos anteriormente, los **argumentos** son aquellos valores que toman los **parámetros** de una función para poder ejecutar el código al interior de ellas, y al interior, pueden ser de cualquier tipo de dato, pero también pueden corresponder a **estructuras de datos**.

```
precios = {'Notebook': 700000,
      'Teclado': 25000,
      'Mouse': 12000,
      'Monitor': 250000,
      'Escritorio': 135000,
      'Tarjeta de Video': 1500000}

def filtrar(diccionario, umbral):
    filtro = {k:v for k,v in diccionario.items() if v > umbral}
    return filtro

filtrar(precios, 12000)
```

```
{'Notebook': 700000,
 'Teclado': 25000,
 'Monitor': 250000,
 'Escritorio': 135000,
 'Tarjeta de Video': 1500000}
```

Funciones como argumentos

En Python **no estamos restringidos** a utilizar solo valores o estructuras de datos como argumentos, de hecho, podríamos utilizar incluso **funciones**.

En el caso de pasar una función como parámetro SÓLO se debe utilizar el nombre, sin paréntesis. Por ejemplo, generemos una función que entregue el tipo de dato de cada uno de los elementos en una lista.

```
lista_numeros = [1,2,3,4,5]
lista_string = ['a','b','c','d','e']

def sumar_contar_tipos(lista,funcion):
      tipos = [type(elemento) for elemento in lista]
      opcion = funcion(lista)
      return tipos, opcion

tipo, conteo = sumar_contar_tipos(lista_string, len)
print(tipo)
print(conteo)

tipo, suma = sumar_contar_tipos(lista_numeros, sum)
print(tipo)
print(suma)
```

Parámetros obligatorios

El carácter obligatorio se da porque en caso de no ingresar el argumento respectivo la función fallará.

En particular en funciones que tienen muchos parámetros (estas pueden ser definidas por el usuario o predefinidas en alguna librería de Python) el referenciar el parámetro es de gran utilidad para que otros usuarios puedan entender qué argumentos necesita la función.

```
def extremo_multiplicado(lista,factor):
    minimo = min(lista)
    maximo = max(lista)
    return factor*minimo, factor*maximo

print(extremo_multiplicado(4,[1,2,3,4]))
```

```
print(extremo_multiplicado([1,2,3,4], 4))
# se entregan en orden
print(extremo_multiplicado(factor = 4, lista = [1,2,3,4]))
```

```
(4,16)
```

Parámetros opcionales o por defecto

Existen casos en los cuales queremos utilizar un parámetro solo en ciertas ocasiones o para diferenciar el comportamiento de la función, es decir, el parámetro será opcional, y no siempre habrá que definirlo. Esto quiere decir que cuando no se defina, este tomará un **valor por defecto**.

```
def elevar(base, exponente, redondear = False):
    if redondear:
        valor = round(base**exponente,2)
    else:
        valor = base**exponente

    return valor
```

```
print(elevar(2, 3))
```

```
19.241905543136184
```

```
print(elevar(2, 3, redondear = True))
```

```
19.24
```

{desafío} latam_

*Args

Podemos definir funciones que utilicen una cantidad variable de argumentos sin definirlos a priori, utilizando ***args**.

```
def sumar_numeros(*args):
    suma = 0
    for num in args:
        suma += num
    return suma

print(sumar_numeros(1, 2, 3))  # Imprime 6
print(sumar_numeros(1, 2, 3, 4, 5))  # Imprime 15
print(sumar_numeros(1, 2))  # Imprime 3
print(sumar_numeros())  # Imprime 0, ya que no hay argumentos
```

**Kwargs

Los **kwargs permitirán un número indeterminado de argumentos, pero estos argumentos deben incluir un nombre ya que el nombre del argumento también pasará a la función.

```
def crear_perfil(**kwargs):
    perfil = {}
    for key, value in kwargs.items():
        perfil[key] = value
    return perfil

# Crear un perfil con diferentes configuraciones
perfil1 = crear_perfil(nombre='Alice', edad=25, ciudad='Barcelona')
perfil2 = crear_perfil(nombre='Bob', edad=30, ocupacion='Ingeniero')

print(perfil1)
print(perfil2)
```

Ejercicio guiado

"Expandiendo los diccionarios"

Expandiendo los diccionarios

Crear una función `get_multiple()` que permita extraer distintos valores añadiendo un número cualquiera de claves.

Paso 1

Creemos el script **get_multiple.py**

Paso 2

Definamos una función que permita tomar muchas claves.

Lo más apropiado para esto será utilizar un *arg.

```
def get_multiple(diccionario, *claves):
    pass
```

Expandiendo los diccionarios

Crear una función `get_multiple()` que permita extraer distintos valores añadiendo un número cualquiera de claves.

Paso 3

Iterar por el diccionario chequeando si existen todas esas claves.

```
def get_multiple(diccionario, *claves):
    return {clave: diccionario[clave] for clave in claves}
```

El diccionario resultante sólo llamara las claves que están ingresadas dentro del *arg *claves y contendrán la clave y el valor asociado a esa clave.

Expandiendo los diccionarios

Crear una función `get_multiple()` que permita extraer distintos valores añadiendo un número cualquiera de claves.

Paso 4

Finalmente, tomemos un diccionario cualquiera y probemos si es posible devolver varias de sus claves.

```
diccionario_prueba = {'manzana': 'verde',
                      'platano': 'amarillo',
                      'frutilla': 'roja'}
resultado = get_multiple(diccionario_prueba, 'manzana', 'platano')
print(resultado)
```

```
{'manzana': 'verde', 'platano': 'amarillo'}
```

{desafío} latam_

/* Variables */

Variables

Locales

Son aquellas definidas dentro de un contexto específico, en este caso en una función. Para la función, serán variables locales aquellas que se definan dentro de su bloque, este ambiente local se conoce como scope.

Globales

Cualquier variable que no se defina dentro dentro de un scope local será entonces una variable global, estas pueden ser accedidas desde cualquier parte en Python.

Alcance de variables

La disponibilidad de las variables tienen una cierta jerarquía. Al momento de llamar una variable, Python buscará si existe dicha variable en su propio **scope**; y en caso de existir utilizará dicho valor, en caso contrario escalará a un ambiente superior.

En este caso tenemos dos ambientes, un **ambiente local** dentro de la función **`get_continent()`** y el **ambiente global**, la variable continent ha sido definida en el ambiente global. Por otra parte, la función **`get_continent()`** imprime la variable continent, pero esta variable no está definida en el ambiente local. Por lo tanto, Python escalará al ambiente global donde sí está definida y la utilizará.

```
continent = 'South America'
def get_continent():
    print(continent)
print_continent()
```

```
South America
```

Una variable local no puede salir al ambiente global por sí sola.

La manera de poder utilizar una variable local en el ambiente global es como ya hemos mencionado anteriormente mediante return, la que viene a ser la conexión entre el scope local de una función y el ambiente local.

Modificando variables globales desde un entorno local

Existen ocasiones pero muy específicas en las que podríamos querer modificar una variable global desde un entorno local. Esta práctica es **poco común y no recomendada**, pero aún así es importante entender que es posible.

```
continent = 'South America'

def get_continent():
    global continent
    continent = 'Africa'

get_continent()
print(continent)
```

```
Africa
```

Precauciones con variables globales

Trabajar con variables globales puede resultar un poco confuso, ya que son consideradas una mala práctica, ya que podrían resultar en errores difíciles de entender en nuestro código.

¿Cómo se rompe un programa con variables globales?

Un programa puede tener miles de líneas de código e incorporar varias librerías (programas de terceros). En este aspecto, es sencillo que alguien llame **por error** a una variable global de su código de la misma forma que otra persona la llamó en su librería, y cualquier cambio en ella podría alterar el funcionamiento del código.

Ejercicio guiado

"Loto"

{desafío} latam_

Loto

Crear un programa que permita simular un sorteo del Loto.

Reglas: Se tiene un pool de números del 1 al 41, y uno a uno se tomarán 6 números al azar, y un séptimo que representará el comodín.

Paso 1

Creemos el archivo **`loto.py`**

Paso 2

Creemos el pool de números a escoger. Para evitar escribir los 41 números podemos utilizar un list comprehension de la siguiente manera:

```
pool = [n for n in range(1,42)]
```

Loto

Crear un programa que permita simular un sorteo del Loto.

Reglas: Se tiene un pool de números del 1 al 41, y uno a uno se tomarán 6 números al azar, y un séptimo que representará el comodín.

Paso 3

Escojamos un número al azar. Para ello podemos utilizar random.choice de la librería random:

```
elegido = random.choice(pool)
print("El primer número es", elegido)
```

Loto

Crear un programa que permita simular un sorteo del Loto.

Reglas: Se tiene un pool de números del 1 al 41, y uno a uno se tomarán 6 números al azar, y un séptimo que representará el comodín.

Paso 4

El mismo número que ha sido escogido podría ser removido del pool para la siguiente elección:

```
# sacamos el 2do, pero debemos evitar que se vuelva
# a sacar el número anterior
pool.remove(elegido)
elegido = random.choice(pool)
print("El segundo número es", elegido)
```

Loto

Crear un programa que permita simular un sorteo del Loto.

Reglas: Se tiene un pool de números del 1 al 41, y uno a uno se tomarán 6 números al azar, y un séptimo que representará el comodín.

Paso 5

Finalmente, el séptimo número es el comodín. Por lo tanto debe acompañarse de un mensaje distinto.

```
# El séptimo número es el comodín
pool.remove(elegido)
print(pool)
elegido = random.choice(pool)
print("El Comodín número es", elegido)
```

Loto

Crear un programa que permita simular un sorteo del Loto.

Reglas: Se tiene un pool de números del 1 al 41, y uno a uno se tomarán 6 números al azar, y un séptimo que representará el comodín.

Paso 6

Condensar todo en una función.

```
def sacar_numero(posicion):
    global pool
    elegido = random.choice(pool)
    pool.remove(elegido)
    print(f'El {posicion} es {elegido}')
```

En este caso posicion corresponderá al número que se está extrayendo en el sorteo.

Loto

Crear un programa que permita simular un sorteo del Loto.

Reglas: Se tiene un pool de números del 1 al 41, y uno a uno se tomarán 6 números al azar, y un séptimo que representará el comodín.

Paso 7

Al ejecutar el programa podemos notar cada uno de los números extraídos y además que el pool de número disminuye en cada iteración. Esto no es necesario y en este caso lo hacemos sólo para asegurarnos que el programa efectivamente haga lo que nos interesa.

```
└─ python loto.py
El primer número es 28
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]
El segundo número es 38
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 29, 30, 31, 32, 33, 34, 35, 36, 37, 39, 40, 41]
El tercer número es 25
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 26, 27, 29, 30, 31, 32, 33, 34, 35, 36, 37, 39, 40, 41]
El cuarto número es 37
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 26, 27, 29, 30, 31, 32, 33, 34, 35, 36, 39, 40, 41]
El quinto número es 24
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 26, 27, 29, 30, 31, 32, 33, 34, 35, 36, 39, 40, 41]
El sexto número es 16
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 26, 27, 29, 30, 31, 32, 33, 34, 35, 36, 39, 40, 41]
El comodín es 35
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 26, 27, 29, 30, 31, 32, 33, 34, 36, 39, 40, 41]
Sorteo Terminado
```

Próxima sesión...

- *Desafío evaluado.*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales