Estructuras de datos y funciones

Organización de un Proyecto en Python y modularización

{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?

- *Explica el sentido de utilizar funciones dentro de un programa distinguiendo su definición versus su invocación.*
- *Utiliza funciones preconstruidas y personalizadas por el usuario con paso de parámetros y que obtienen un retorno.*

{desafío} latam_

/* Organización de un Proyecto en Python */

Docstrings

Es muy probable que a medida que un proyecto en Python crece se generen muchas funciones, que van quedando guardadas, pero con el tiempo uno va olvidando su funcionalidad. Es por esto que siempre se recomienda como buena práctica que cuando un proyecto comience a crecer se vaya **documentando de manera apropiada**.

Los **docstrings** son una **documentación** que nosotros mismos podemos implementar dentro de nuestras funciones para poder recordar, cuando pasa el tiempo, cuál es la intención de la función, cómo funciona y qué parámetros son necesarios para que funcione de manera apropiada.

Docstrings

Se implementa al inicio de la función utilizando 3 pares de comillas (pueden ser simples o dobles):

```
def elevar(base, exponente):
    """Esta función tiene como objetivo elevar una base a un exponente"""
    return base**exponente
```

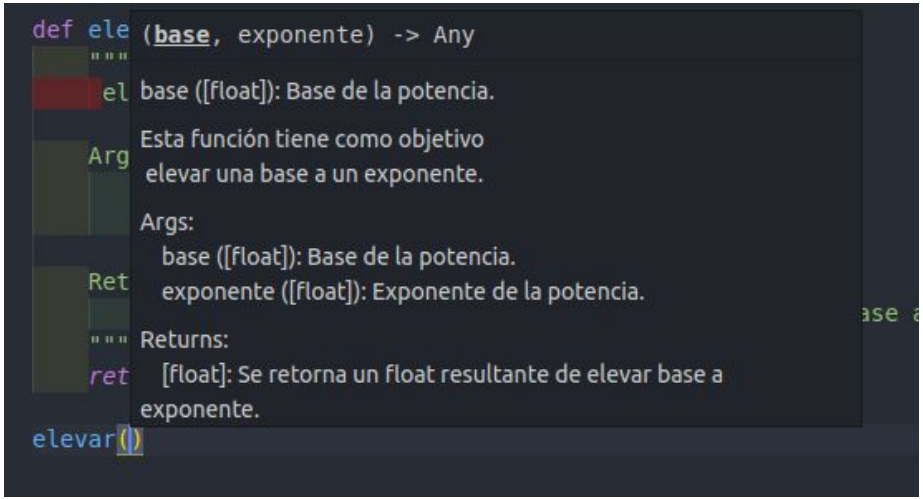
Si invocamos la función, los **editores de texto** serán capaces de mostrarnos el Docstring asociado, por lo que, dependiendo del grado de detalle que se dé a la documentación podremos entender de mejor manera nuestro código.

Tipos de Docstrings

Google

Fomenta generar un pequeño resumen de lo que hace la función, definir los parámetros con el tipo de datos que se espera de ellos, y una descripción y el tipo del retorno de la función.

```
def elevar(base, exponente):
    """[summary]
    Args:
        base ([type]): [description]
        exponente ([type]): [description]
    Returns:
        [type]: [description]
    """
    return base**exponente
```



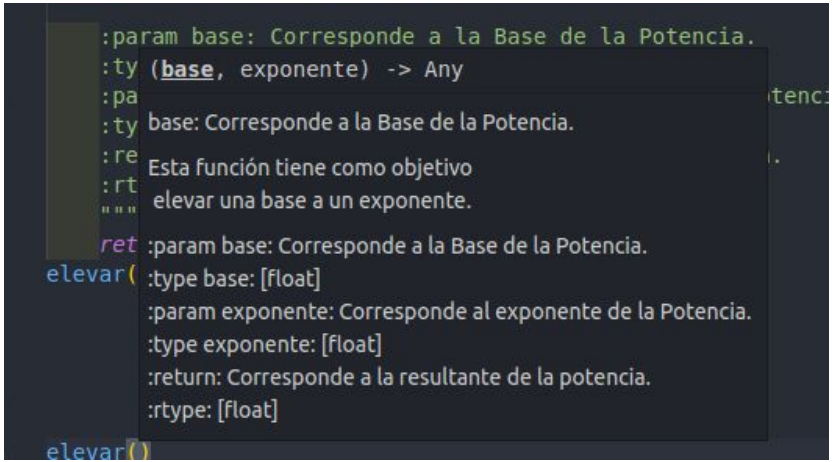
{desafío} latam_

Tipos de Docstrings

Sphinx

Herramienta especializada en la creación automática de documentación. En este caso, es muy similar a la de Google solo que definen los parámetros y sus tipos en líneas distintas.

```
def elevar(base, exponente):
    """[summary]
    :param base: [description]
    :type base: [type]
    :param exponente: [description]
    :type exponente: [type]
    :return: [description]
    :rtype: [type]
    """
    return base**exponente
```



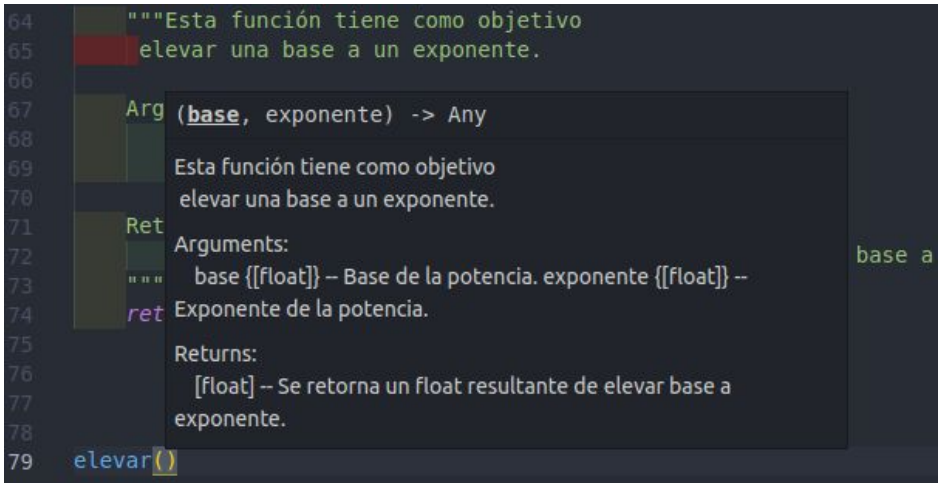
{desafío} latam_

Tipos de Docstrings

Docblockr

Casi idéntico que el de Google, pero con otro tipo de separadores. Si bien indica que se está esperando un argumento, en este caso el argumento base, no genera esa confusión y sensación de error de repetir el argumento al inicio.

```
def elevar(base, exponente):
    """[summary]
    Arguments:
        base {[type]} -- [description]
        exponente {[type]} -- [description]
    Returns:
        [type] -- [description]
    """
    return base**exponente
```



{desafío} latam_

Tipos de Docstrings

Numpy

Librería de Computación Científica muy popular en Ciencia de Datos. El formato utilizado es reconocido como texto enriquecido en editores como VS Code, por lo que destaca algunos títulos generando una documentación más elegante.


```numpy
def elevar(base, exponente):
    """Esta función tiene como objetivo
     elevar una base a un exponente.
    Parameters
    ----------
    base : [float]
        Base de la Potencia.
    exponente : [float]
        Exponente de la Potencia
    Returns
    -------
    [float]
        Retorna el resultado de elevar base a
exponente
    """
    return base**exponente
```

{desafío} latam_

/* Refactorización */

Refactorización

Al momento de crear una solución, esta no aparece de manera directa, sino por etapas, donde uno va haciendo pruebas hasta verificar que el código implementado efectivamente solucione el problema. Una vez alcanzada la solución muchas veces notamos que el código podría ser organizado y estructurado de manera mucho más eficiente de lo que lo tenemos actualmente.

Supongamos el siguiente caso:

```
valor_entrada = 10
valor_1 = valor_entrada**2 + valor_entrada**3
valor_2 = valor_1*2 + valor_1*3 + valor_1*4
valor_3 = valor_2**2 + valor_2**3
valor_4 = valor_3*2 + valor_3*3 + valor_3*4
valor_5 = valor_4**2 + valor_4**3
valor_6 = valor_5*2 + valor_5*3 + valor_5*4
```

Tenemos el siguiente código, el cual realiza un cálculo muy difícil. Si lo miramos bien notamos ciertos patrones, como, por ejemplo, los valores 1, 3 y 5 siguen una misma lógica de sumar el cuadrado y el cubo de un número.
Por otro lado, los valores 2, 4 y 6 están sumando el doble, el triple y el cuádruple de un valor.

Refactorización

La refactorización consistirá en abstraer este código y generar funciones que se encarguen del cálculo de patrones similares, básicamente significa, aplicar el principio DRY.

Creemos las siguientes funciones:

```
def cuadrado_cubo(valor):
    return valor**2 + valor**3
def mult_234(valor):
    return valor*2 + valor*3 + valor*4
```

Al generar estas dos funciones el código puede refactorizarse de la siguiente manera:

```
def cuadrado_cubo(valor):
    return valor**2 + valor**3
def mult_234(valor):
    return valor*2 + valor*3 + valor*4
valor_entrada = 10
valor_1 = cuadrado_cubo(valor_entrada)
valor_2 = mult_234(valor_1)
valor_3= cuadrado_cubo(valor_2)
valor_4 = mult_234(valor_3)
valor_5 = cuadrado_cubo(valor_4)
valor_6 = mult_234(valor_5)
```

Refactorización

Mirando esto notamos que, de hecho, el código se puede refactorizar aún más. Podríamos crear la siguiente función:

```
def op_combinada(valor):
    var_intermedia = cuadrado_cubo(valor)
    return mult_234(var_intermedia)
```

Esta función abstrae pares de operaciones que se están aplicando, luego el código se puede refactorizar de la siguiente manera:

```
def cuadrado_cubo(valor):
    return valor**2 + valor**3
def mult_234(valor):
    return valor*2 + valor*3 + valor*4
def op_combinada(valor):
    var_intermedia = cuadrado_cubo(valor)
    return mult_234(var_intermedia)
valor_entrada = 10
valor_2 = op_combinada(valor_entrada)
valor_4 = op_combinada(valor_2)
valor_6= op_combinada(valor_4)
```

Refactorización

En este caso, sí se acorta el número de operaciones necesarias, pero hay que decir que esto se puede refactorizar aún más:

Podríamos crear la siguiente función:

```
def compose(f, n):
      def fn(x):
      for _ in range(n):
            x = f(x)
      return x
      return fn
```

{desafío} latam_

Permite repetir la ejecución de una función n de veces sobre el resultado que ella misma arroja.

```
def cuadrado_cubo(valor):
    return valor**2 + valor**3
def mult_234(valor):
    return valor*2 + valor*3 + valor*4
def op_combinada(valor):
    var_intermedia = cuadrado_cubo(valor)
    return mult_234(var_intermedia)
def compose(f, n):
      def fn(x):
          for _ in range(n):
              x = f(x)
          return x
      return fn
valor_entrada = 10
valor_6 = compose(op_combinada, 3)(valor_entrada)
```

Refactorización

- Uno puede refactorizar cuantas veces sea necesario, pero ¿es siempre esto lo óptimo? La respuesta es categórica: NO. En este caso particular este nivel de factorización no genera demasiados beneficios, e introduce una función compose() extremadamente compleja, ya que a muy pocos programadores se les ocurriría por sí solos.

- En conclusión, técnicas como la refactorización tienen que utilizarse con cautela, pues la idea de la factorización es siempre introducir funciones que faciliten el entendimiento del código, no que lo compliquen. Por lo tanto, dependiendo de las necesidades, es responsabilidad del desarrollador determinar cuántos niveles de refactorización utilizar y siempre pensando en facilitar la estructura del código y no introducir complejidades innecesarias.

Stackoverflow es la web más grande de ayuda a los programadores. Programadores de todo nivel comparten conocimiento de código acerca de cómo resolver un problema. Es común que ante cualquier duda que uno pueda tener, sin importar el lenguaje de programación, siempre existirá alguna sugerencia al respecto.

¿Qué es la refactorizar?

{desafío}
latam_

Próxima sesión...

- *Modularización y experiencia de usuario*

{desafío}
latam_

Academia de
talentos digitales