Funciones y modularización

Sesión Conceptual 1

{desafío}
latam_


Inicio
Desarrollo
Cierre
Objetivo de la sesión
Activación de conceptos clave
Conceptualización
Ejercicios
Quiz
Cierre

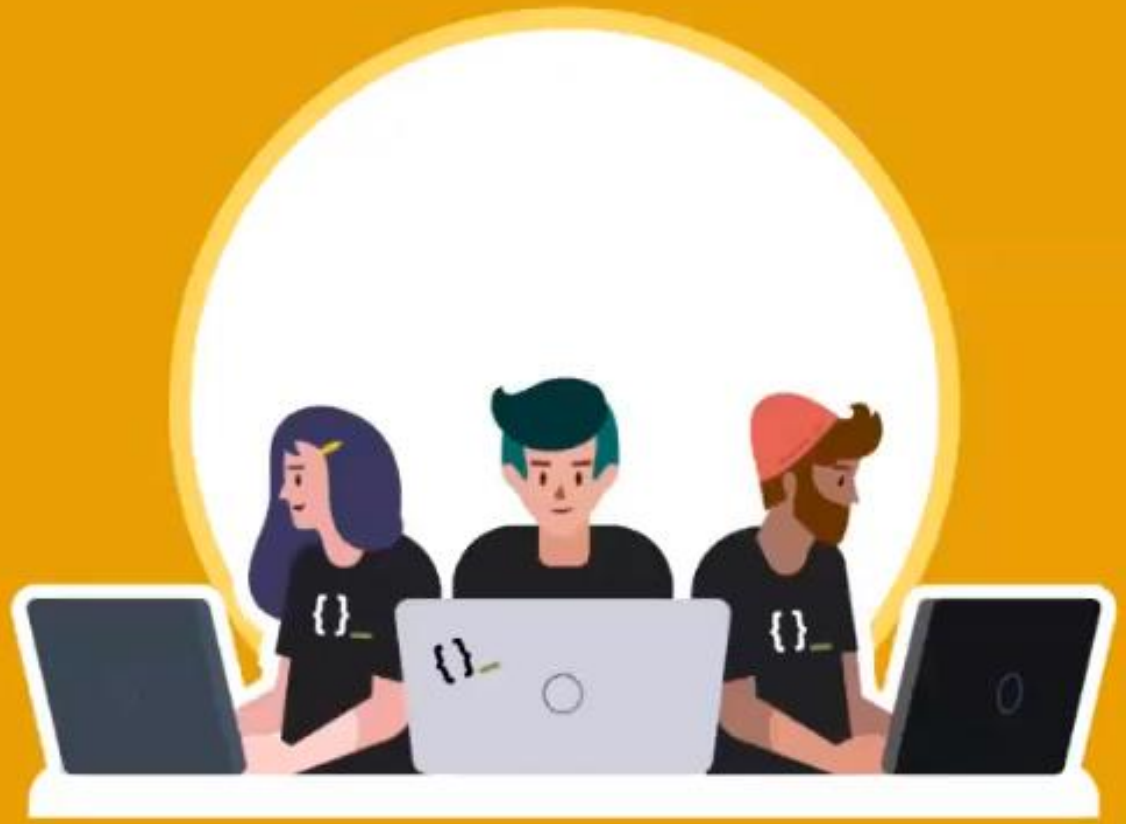
Inicio

10 minutos

{desafío} latam_

¿Para qué son útiles las funciones y cuáles son sus principales características?

{desafío}
latam_



Desarrollo

70 minutos

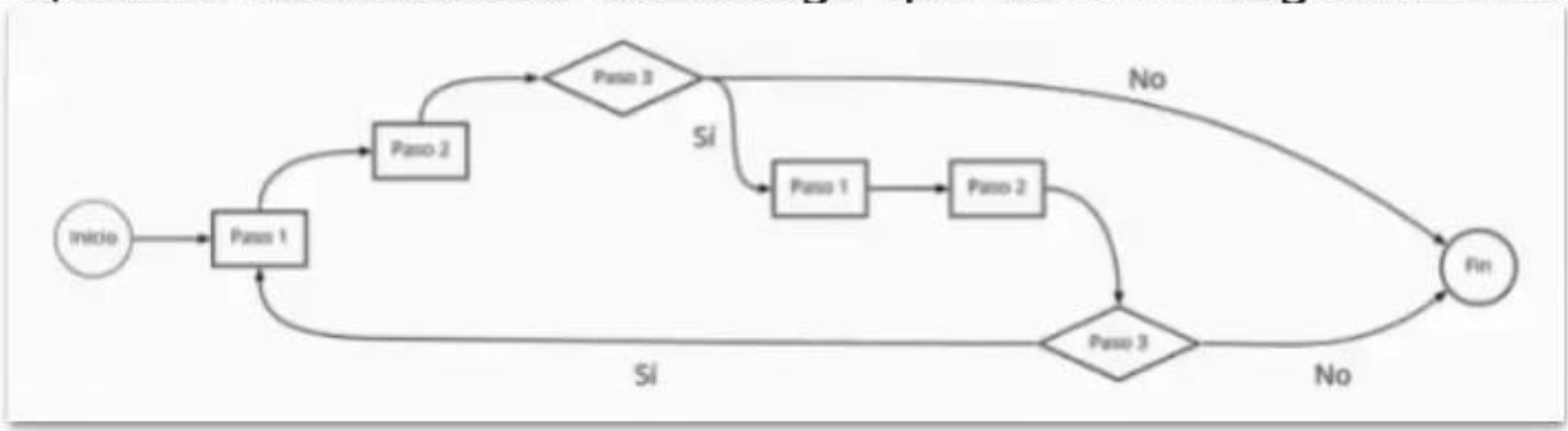
{desafío}
latam_

/* Condensando código */

{desafío}
latam_

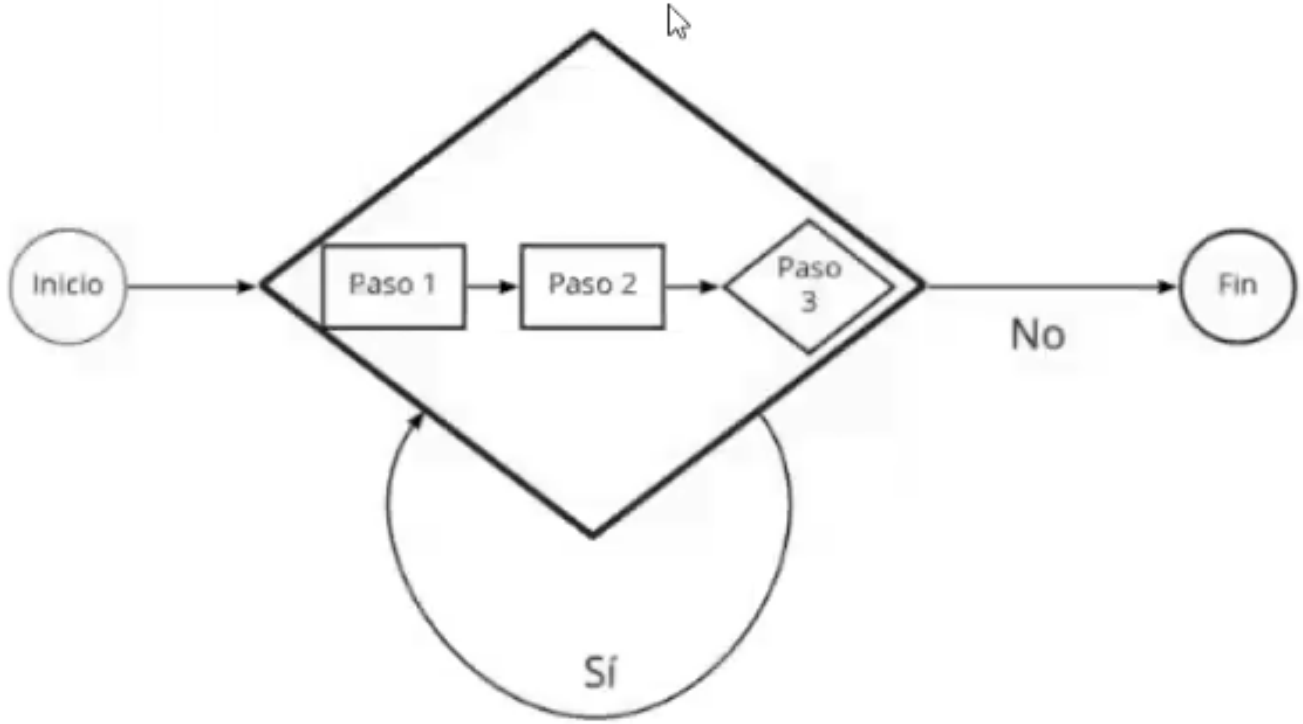
Necesidad de Funciones

Podemos notar que acá se da un fenómeno de duplicación o generación de código redundante. Los pasos 1, 2 y 3 están escritos dos veces y esto, normalmente, termina resultando en código que es más largo de lo necesario.



Necesidad de Funciones

Este problema se soluciona abstrayendo ese código dentro de una función. Esta función la llamaremos `todos_los_pasos`() y contendrá los pasos 1, 2 y 3 de manera



Principio DRY

Don’t Repeat Yourself:

“Si tengo que copiar y pegar un trozo de código 2 o más veces es muy probable que necesite crear una función.”

Ejercicio guiado

"La encuesta"

{desafío} latam_

Quiz

/* Anatomía de una función */

Parámetros

Un parámetro es un elemento que podrá ser utilizado dentro de la función para realizar sus cálculos. El uso de un parámetro, como dijimos anteriormente, permite crear funciones que son reutilizables en muchos casos.

```
def elevado_2(x):
    print(x**2)

elevado_2(2)
elevado_2(3)
elevado_2(4)
```

```
4
9
16
```

Argumentos

Un argumento corresponde a los valores que tomará el parámetro para ser utilizado dentro de la función.

```
def elevar(x,y):
    print(x**y)

elevar(2,2)
elevar(3,3)
elevar(4,2)

4
27
16
```

Ejercicio Guiado

"Revisitando la encuesta"

{desafío} latam_

Retorno

En ocasiones, las funciones necesita devolver un valor que pueda ser utilizado posteriormente, y a este valor se le conoce como retorno. Para evitar errores como el que acabamos de ver tenemos que añadir la palabra reservada `return`.

```
def elevar(base,exponente):
    print(base**exponente)

print(elevar(2,2) + elevar(3,3) + elevar(4,2))
```

```
47
```

Fin de la función

El retorno de una función también implica el término de la función. Esto quiere decir, que cualquier código que se encuentre después del `return` no será ejecutado.

```
def prueba_return():
    a = "Esta línea se va a imprimir"
    b = "Esta línea no se va a imprimir"
    return a # Punto de salida
    print(b)

print(prueba_return())
```

```
'Esta línea se va a imprimir'
```

Múltiples retornos

Una función puede retornar tantos valores como sea necesario. En caso de retornar más de un valor, las buenas prácticas indican que estos deben tener alguna relación, como por ejemplo, derivarse del mismo proceso.

{desafío} latam_

Ejercicio Guiado

"Estandarización"

{desafío} latam_

Quiz

{desafío}
latam_

/* Tipos de Argumentos y Parámetros */

{desafío}
latam_

Tipos de argumentos

Calcular el máximo y el mínimo valor de una lista:

```
def extremos(lista):
    minimo = min(lista)
    maximo = max(lista)
    print(f"El valor mínimo
es {minimo}")
    print(f"El valor mínimo
es {maximo}")

extremos([3,5,2,7])
```

```
El valor mínimo es 2
El valor mínimo es 7
```

Filtrar los elementos de un diccionario que cuestan más de cierto valor:

```
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)
```

{desafío} latam_

Funciones como argumentos

Es bueno mencionar que en Python no estamos restringidos a utilizar solo valores o estructuras de datos como argumentos, de hecho, podríamos utilizar incluso funciones. Para ello es importante hacer la siguiente salvedad, en el caso de pasar una función como parámetro SÓLO se debe utilizar el nombre, sin paréntesis.

Parámetros obligatorios

- Es importante recalcar que el carácter obligatorio se da porque en caso de no ingresarse el argumento respectivo la función fallará.
- Los argumentos correspondientes a parámetros obligatorios deben ser rellenados en el mismo orden que se definieron.
- En particular en funciones que tienen muchos parámetros, el referenciarlos es de gran utilidad para que otros usuarios puedan entender qué argumentos necesita la función.

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
```

{desafío}
latam_

*Args y **Kwargs

Los `*args` permiten utilizar tantos parámetros como sean necesarios sin la necesidad de que sean definidos a priori. Por ejemplo, podríamos crear la función suma() que sea la solución genérica:

```
def f(*args):
    return args

output = f(1,[2,3],'hola',{'clave':[4]})
print(type(output))

<class 'tuple'>
```

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 f(**kwargs):
      return kwargs

output = f(valor = 1, texto = 'hola',
lista_nombres = [4,5,6,7])
print(type(output))
```

{desafío} latam_

Ejercicio Guiado

"Expandiendo los diccionarios"

{desafío} latam_

Quiz

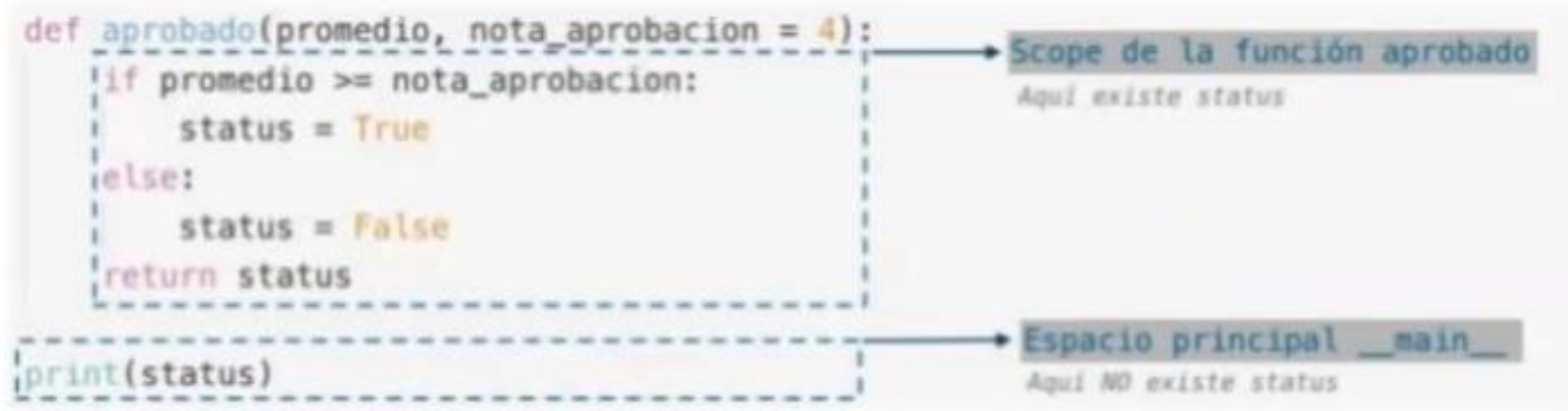
{desafío}
latam_

/* Alcance de variables */

{desafío}
latam_

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, y una variable definida en un scope local no puede ser accedida fuera de este.



Variables 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.

Variables globales

Alcance de variables

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.

Modificando variables globales desde un entorno Local

Cuando se quiera modificar una variable global desde un entorno local, se utiliza la palabra reservada `global` acompañada de la variable que se quiere modificar.

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.

Undefined

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_

Cierre

10 minutos

{desafío} latam_

Preguntas de cierre

- ¿Qué es una función?
- ¿Por qué es importante utilizar funciones en un código?
- ¿Cuál es la diferencia entre un parámetro y un argumento?
- ¿Qué tipo de parámetros existen?
- ¿Qué es el scope de variables y para qué sirven?

{desafío} latam_

{desafío} latam_

Academia de talentos digitales