</ Ayudantía 6:
Funciones
/>

Rolando Rojas

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Temas a tratar

{01}
Introducción a Funciones

{02}
Estructura de Funciones

{03}
Uso de Funciones

{04}
Modularidad y reuso

{05}
Ejemplos

{06}
Ejercicios

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Introducción a Funciones

En programación, una función es un bloque de código que realiza una tarea específica cuando se llama. Se define con un nombre que la identifica y, opcionalmente, puede aceptar parámetros de entrada que permiten modificar su comportamiento. Una función puede devolver un valor como resultado de su ejecución mediante la palabra clave ***return****. Si no se especifica un valor de retorno, la función devuelve* ***None*** *por defecto.*

```
def(parametroA,parametroB):
        …codigo
        …codigo
        …codigo
        return variableA
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Estructura de funciones

Considera la siguiente función matemática.
Esta función toma un número X como entrada y devuelve el resultado de multiplicar X por 3 y luego sumar 5.

$$f(x) = 3x + 5.$$

```python
def funcion_matematica(x):
    resultado = 3 * x + 5
    return resultado
```

En este ejemplo, la función funcion_matematica(x) toma un argumento x y calcula el resultado de la operación 3x+5.
El cálculo se almacena en la variable resultado y luego se devuelve usando return.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Uso de funciones

Aquí, la función sumar toma dos parámetros (a y b) y devuelve su suma. El valor de retorno se puede almacenar en una variable como resultado.

```python
def sumar(a, b):
    return a + b

resultado = sumar(2, 3)  # resultado es igual a 5
```

```python
# Definimos una función para calcular el índice de masa corporal (IMC)
def calcular_imc(peso, altura):
    # Calculamos el IMC dividiendo el peso por el cuadrado de la altura
    imc = peso / (altura ** 2)
    # Devolvemos el IMC calculado
    return imc

# Llamamos a la función con los valores de peso y altura de una persona
peso_persona = 70  # en kilogramos
altura_persona = 1.75  # en metros

# Calculamos el IMC usando la función
imc_persona = calcular_imc(peso_persona, altura_persona)

# Mostramos el resultado de la función
print(imc_persona)
```

Ejemplo de uso de una función que calcula el IMC de una persona, necesita los datos del peso y altura.
El resultado del ejemplo queda guardado en la variable imc_persona, por lo que se imprime mediante print.

0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Modularidad y reuso (1/2)

La modularidad y el reuso son útiles en Python porque ayudan a dividir el código en partes más pequeñas y manejables, cada una dedicada a una tarea específica. Esto hace que el programa sea más fácil de entender y de corregir cuando hay problemas. Además, puedes usar funciones en diferentes partes del programa o en otros proyectos, evitando repetir el mismo código. Esto mejora la calidad del software y te ahorra tiempo en el desarrollo

```python
# Definimos la función para calcular el cuadrado de un número
def calcular_cuadrado(numero):
    cuadrado = numero * numero
    return cuadrado

# Definimos la función para verificar si un número es impar
def es_impar(numero):
    return numero % 2 != 0

# Definimos la función para multiplicar dos números
def multiplicar(a, b):
    producto = a * b
    return producto

# Definimos algunos números para probar las funciones
numero = 3
a = 4
b = 6

# Calculamos el cuadrado del número usando la función
cuadrado = calcular_cuadrado(numero)
print("El cuadrado de", numero, "es:", cuadrado)

# Verificamos si el número es impar usando la función
if es_impar(numero):
    print(numero, "es un número impar.")
else:
    print(numero, "es un número par.")

# Multiplicamos dos números usando la función
producto = multiplicar(a, b)
print("El producto de", a, "y", b, "es:", producto)
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1

Ejemplo de modularidad

1 1 0 1

</ Modularidad y reuso [2/2]

```python
# Definimos una función para convertir una cadena a mayúsculas
def convertir_a_mayusculas(cadena):
    mayusculas = cadena.upper()
    return mayusculas

# Usamos un ciclo para convertir varias cadenas a mayúsculas
contador = 1
numero_cadenas = 3

# Procesamos las cadenas usando la función
while contador <= numero_cadenas:
    # Pedimos al usuario que ingrese una cadena
    cadena = input("Ingresa una cadena de texto: ")

    # Convertimos la cadena a mayúsculas usando la función
    cadena_mayusculas = convertir_a_mayusculas(cadena)

    # Mostramos la cadena en mayúsculas
    print("La cadena en mayúsculas es:", cadena_mayusculas)

    # Incrementamos el contador
    contador += 1
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 *Ejemplo de reuso* 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Ejemplos de funciones

Ya conocen muchos ejemplos, ya que llevan ocupando funciones desde el inicio del curso, como pueden ver en las funciones que trae Python por defecto:

1. **Manipulación de datos:**
   - `` `len(cadena)` ``: Devuelve la longitud de una cadena de texto.
2. **Conversión de datos:**
   - `` `int(valor)` ``: Convierte un valor a un entero.
   - `` `float(valor)` ``: Convierte un valor a un número de punto flotante.
   - `` `str(valor)` ``: Convierte un valor a una cadena de texto.
3. **Entrada/salida:**
   - `` `input()` ``: Solicita al usuario que ingrese datos por teclado y devuelve una cadena de texto con la entrada.
   - `` `print()` ``: Imprime uno o más valores en la consola.
4. **Manipulación de texto:**
   - `` `upper()` ``: Convierte una cadena de texto a mayúsculas.
   - `` `lower()` ``: Convierte una cadena de texto a minúsculas.
   - `` `strip()` ``: Elimina los espacios en blanco al principio y al final de una cadena de texto.
5. **Funciones matemáticas:**
   - `` `abs(numero)` ``: Devuelve el valor absoluto de un número.
   - `` `round(numero, decimales)` ``: Redondea un número al número de decimales especificado.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0

</ Ejercicio 1:

Crear una función codigo_palabra(codigo) que reciba un código encriptado de solo letras y entregue el mensaje desencriptado. La regla de desencriptación es la siguiente: la palabra desencriptada se obtiene recorriendo desde el final de la palabra hasta el comienzo, considerando solo las letras en ***ubicaciones impares****. Empezando desde la última letra. La intención es obtener el lugar de la reunión.*

```
print(codigo_palabra('aczaarltp'))
plaza
print(codigo_palabra('oagriatgrreov'))
vertigo
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Solución ejercicio 1:

```python
def codigo_palabra(codigo):
    descifrado = ""
    indice = len(codigo) - 1
    while indice >= 0:
        descifrado += codigo[indice]
        indice -= 2
    return descifrado
print(codigo_palabra('aczaarltp'))
print(codigo_palabra('oagriatgrreov'))
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Ejercicio 2: Ruteo

```
def f1(st):
    c=0
    h=len(st)
    s=0
    while c<h:
        if c!=h-1 and st[c+1]=='-':
            c+=2
        else:
            c+=3
        s+=1
    return s
n='1-5-13-18-19'
a = f1(n)
print(a)
```

global		f1			
n	a	st	c	h	s

1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Solución ejercicio 2:

global		f1			
n	a	st	c	h	s
"1-5-13-18-19"					
		"1-5-13-18-19"			
			0		
				12	
					0
			2		
					1
			4		
					2
			7		
					3
			10		
					4
			13		
					5
	5				

5

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</

Ejercicio 3: Ruteo

```
def f1(x):
    x = int(x)
    if x % 2 == 0:
        return True
    return False

def f2(b):
    i = 0
    while i < 10:
        if str(i) == b:
            return True
        i += 1
    return False

x = '1a2'
i = 0
p = 0
while i < len(x):
    if f2(x[i]):
        if f1(x[i]):
            p += 1
    i += 1

if p > len(x) / 2:
    print('Sirve')
else:
    print('No Sirve')
```

global			f1	f2	
x	i	p	x	b	i

1 0 1 1 0 1 1 0 1

0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</

Solución ejercicio 3:

global			f1	f2	
x	i	p	x	b	i
"1a2"					
	0				
		0			
				"1"	
					0
					1
			"1"		
			1		
	1				
				"a"	
					0
					1
					2
					3
					4
					5
					6
					7
					8
					9
					10
	2				
				"2"	
					0
					1
					2
			"2"		
			2		
		1			
	3				

No sirve

1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 1 1 0 1

Próxima semana:
Precertamen en horario de ayudantía

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Ayudantía 6: Funciones />

¿Dudas?
rolando.rojass@usm.cl

Rolando Rojas

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1