



Laboratorio 5 - Funciones

Objetivos

Parte 1: Funciones

Introducción

Actividad 1: Construyendo Funciones

Actividad 2: Variables Globales y Locales

Actividad 3: Prints vs Return

Actividad 4: Ejercitemos

Parte 2: Desafíos

Desafío 1: El mejor censista de Chile

Desafío 2: El viaje de dominio interestelar de Reinhardt

Desafío 3: Carreras

Desafío 4: Ajedrez Mágico

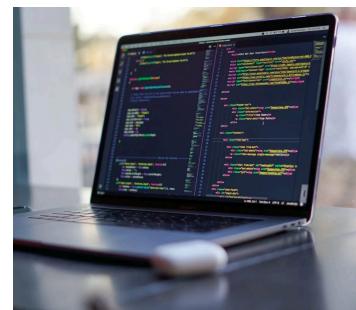
Objetivos

1. Implementar funciones que resuelvan subproblemas particulares dentro de un problema más amplio, utilizando una interfaz bien definida (parámetros y valor de retorno).
2. Comprender la diferencia entre print y return, en el contexto de una función.
3. Descomponer problemas en subproblemas que pueden ser abordados mediante funciones. Integrar la solución final.

Parte 1: Funciones

Introducción

Hasta ahora, hemos estado escribiendo programas que siguen una secuencia de instrucciones una tras otra. Esto está bien para resolver problemas pequeños, pero a medida que tus programas se vuelven más largos o necesitas repetir ciertas tareas muchas veces, **el código puede volverse desordenado, repetitivo y difícil de mantener.**



Aquí es donde entran las **funciones**.

¿Qué es una función?

Una **función** es un bloque de código que cumple una tarea específica. Puedes pensar en ella como una **máquina** que recibe una entrada, hace algo con ella, y entrega un resultado. Sirve para **organizar, reutilizar y entender mejor** tu código.

💡 ¿Por qué es útil usar funciones?

- 🔄 **Evitan repetir código:** Si una misma operación se repite varias veces, puedes definirla una sola vez como función.
- 🏗️ **Permiten dividir un problema grande en partes pequeñas:** Cada parte se puede resolver de forma separada y más sencilla.
- 🧠 **Hacen que el código sea más legible y mantenible:** Es más fácil leer "promedio(n1, n2, n3)" que entender un cálculo largo en cada lugar donde se use.
- 🔎 **Facilitan la búsqueda de errores (debugging):** Si algo no está funcionando, puedes revisar solo la función relacionada.
- 🔄 **Promueven la reutilización del código:** Una función bien escrita puede usarse en muchos programas distintos sin modificarla.

💡 ¿Ya usabas funciones sin saberlo?

¡Sí! Desde el comienzo del curso has estado utilizando funciones sin darte cuenta. Algunas funciones **ya vienen definidas en Python**, como:

- `len(texto)` → Calcula la longitud de un string.
- `int(numero_str)` → Convierte un string a entero.
- `float(valor)` → Convierte un string o entero a decimal.

- `str(numero)` → Convierte un número a string.
- `print(algo)` → Muestra algo en pantalla.
- `input("¿Cómo te llamas?")` → Pide un dato al usuario.

Todas estas tienen **nombre**, reciben **parámetros** (los datos que necesitan para trabajar), y algunas entregan un **resultado (retorno)**.

Por ejemplo, cuando usamos la función `len(texto)` la utilizamos llamándola de esta forma:

```
palabra = "Hola"  
cantidad = len(palabra)
```

En este caso sabemos que:

- El **nombre** de la función es `len`.
- Recibe como **parámetro** un *string*, en este caso le pasamos por ejemplo la variable `palabra`.
- Y el **retorno** que entrega es un número `int` que **representa** la cantidad de caracteres del *string*.

1. Ahora completa tú el **nombre**, **parámetros** y **retorno** de estas funciones:

```
edad_txt = "18"  
edad = int(edad_txt)
```

a. Nombre: _____

b. Parámetros: _____

c. Retorno: _____

```
nombre = input("¿Cómo te llamas?")
```

a. Nombre: _____

b. Parámetros: _____

c. Retorno: _____

Actividad 1: Construyendo Funciones

Para crear una función necesitamos definir sus 4 componentes principales:

```
def nombre_funcion (param1,param2,...,paramn):  
    sentencias a ejecutar en la función  
    return algo
```

Una función puede no tener parámetros o retorno

- El **nombre**: sirve para identificar la función y llamarla. El nombre **no debe contener espacios**. Cada función debe tener un nombre único.
- Los **parámetros**: valores que recibe la función como entrada, y que utilizará dentro de su código.
- El **código** de la función: operaciones que hace la función.
- El **resultado** (o valor de retorno): valor final que entrega la función.



Cosas importantes a tener en cuenta al momento de crear una función:

```
def nombre_función (param1,param2,...,paramn):  
    sentencias a ejecutar en la función  
    return algo
```

- Recuerda colocar `def` al inicio de la línea, ya que indica que se está definiendo una función.
- Puede recibir **muchos parámetros** separados comas.
- Las funciones también pueden no recibir ningún parámetro, y puede también no retornar nada.
- Recordar los dos puntos y paréntesis para definir sus parámetros.
- Recordar que una función debe ser creada antes de ser llamada.
- **Jamás crear funciones dentro de ciclos o condicionales**. Se recomienda crear todas las funciones antes del programa principal.

Un ejemplo de función simple que podemos crear es la función `suma` : recibe dos **parámetros** (los números `n1` y `n2`), y **retorna** la suma de ambos.

```
def suma(n1,n2):
    sum = n1 + n2
    return sum
```

Para llamar a la función `suma` de dos números **dentro de un programa**, se puede guardar la suma en una variable `resultado` de esta forma:

```
numero1 = int(input("Ingrese número 1: "))
numero2 = int(input("Ingrese número 2: "))
resultado = suma(numero1,numero2)
```

El programa completo quedaría así, siempre creando la función antes de llamarla:

```
def suma(n1,n2):
    sum = n1 + n2
    return sum

numero1 = int(input("Ingrese número 1: "))
numero2 = int(input("Ingrese número 2: "))
resultado = suma(numero1,numero2)
```

Ejercitemos la creación de funciones:

1. Analiza:

```
def cuadrado(n):
    return n * n

resultado = cuadrado(3)
print(resultado)
```

- a. ¿Qué imprime este programa? Verifica tu respuesta copiando y ejecutando el programa.

- b. Reconoce y anota los 4 componentes de la función.
2. Completa esta función para que retorne el triple de un número:

```
def triple(n):  
    # tu código aquí
```



Para probar una función debes usarla en tu programa. Por ejemplo, para probar la función `triple`, después de definirla puedes poner:

```
print(triple(4))
```

Y debería imprimir `12`.

3. Usa esta función ya definida:

```
def es_par(n):  
    return n % 2 == 0
```

Para hacer un programa que le pida un número al usuario, e imprima el mensaje `"Es par"` si el número es par, o `"Es impar"` en caso contrario.



Cuando te piden utilizar una función que ya está creada en tu programa, significa que **no la puedes modificar**, y que **sí o sí debe usarla en algún momento**.

En este sentido, primero tienes que entender bien cómo se comporta la función para luego usarla. Pruébala con distintos valores de parámetros y ve qué retorna.

En este caso por ejemplo, la función `es_par` recibe como parámetro `n`, que luego usa para hacer división entera con 2, por lo que se puede asumir que el parámetro `n` debe ser un número.

Actividad 2: Variables Globales y Locales

Sabemos que las funciones son pequeños programas o subrutinas. Las variables que existen **dentro de la función** no son accesibles a las demás funciones o al programa principal, aún cuando tengan el mismo nombre. A esto le llamamos **variables locales**, ya que son solo utilizables en su contexto específico.

Por ejemplo, si intentamos ejecutar el siguiente programa en el que se define la función llamada `funcion`, se puede ver que dentro de ella se crea la variable `a` que es local.

Por lo que luego, como afuera de la definición de `funcion` se intenta imprimir `a`, el programa tirará error porque la variable `a` no existe en ese contexto:

```
def funcion():
    a = 7
    return a
funcion()
print(a)
```



```
Traceback (most recent call last):
  File "C:\Users\mauro\Desktop\ejemplo3.py", line 6, in <module>
    print(a)
NameError: name 'a' is not defined
```

Para arreglar ese error, si necesitáramos usar el valor `a` fuera de la función, lo que tendríamos que hacer es guardar el valor que retorna en otra variable, y ahí recién imprimir esta variable:

```
def funcion():
    a = 7
    return a
a = funcion()
print(a)
```

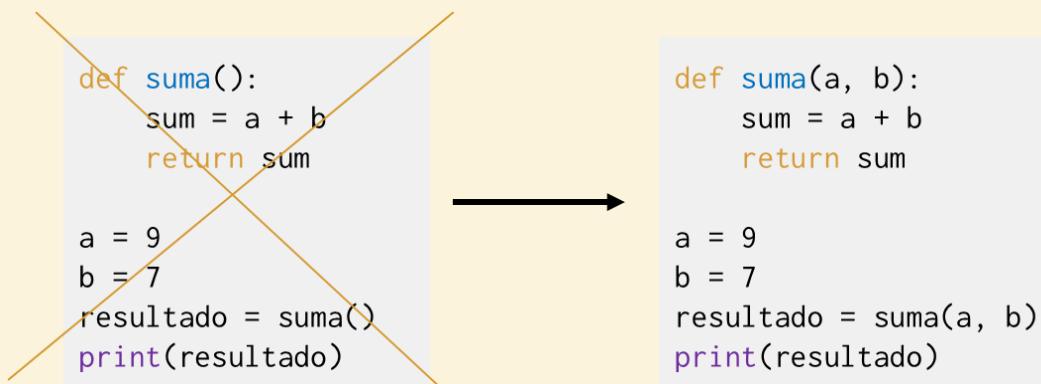


```
7
>>>
```

En este caso, la variable `a` que está definida dentro de la función es una variable **local**, y la variable `a` que está definida después que guarda el retorno sería una variable **global**. Son diferentes variables, aún cuando tienen el mismo nombre, porque están creadas en contextos diferentes.



A pesar de que las variables **globales** (las que están definidas en el programa principal y no en funciones) son accesibles por las funciones, se aconseja que **todo lo que necesite una función se le pase por parámetro**:



Veamos si es que entendemos los conceptos de variable local y global:

1. ¿Qué imprime este código? Analiza primero sin ejecutar el código, y luego comprueba tu respuesta.

```
def f():
    a = 3
    return a + 1

a = 10
print(f())
print(a)
```

Actividad 3: Prints vs Return

Como ya vimos, la sentencia `return` **no es lo mismo** que la sentencia `print`:

- `return`: Nos permite **retornar un valor** desde una función, es decir, devolver un resultado que puede ser **guardado en una variable, usado en cálculos posteriores, o enviado a otra función**.
- `print`: Solo **muestra información por pantalla**. Su propósito es **comunicarse con la persona usuaria**, no con el resto del programa. Lo que imprime no puede ser guardado ni procesado por el código.



Para própositos de este curso, por regla general vamos a imprimir y pedir inputs **sólo en el programa principal y NO en las funciones.**

¿La razón? Esto ayuda a que cada función tenga un propósito claro: **recibir datos, hacer un cálculo, y devolver un resultado**. Así podemos reutilizar nuestras funciones fácilmente, combinarlas, y probarlas sin depender de la interacción con el usuario.

Veamos un ejemplo:

```
# Esta función calcula el doble de un número y lo retorna
def doble(n):
    return n * 2
```

```
# En el programa principal:
resultado = doble(5)
print("El resultado es:", resultado)
```

📌 Aquí **la función no imprime nada**. Solo calcula y devuelve el resultado `10`. El `print` está en el programa principal.

🚫 En cambio, si hubiésemos usado `print` en vez de `return` en la función `doble`, el programa imprimirá `10` una vez, pero `resultado` **tendrá el valor `None`**, porque la función **no retornó nada**. Solo imprimió algo, y luego terminó:

```
def doble(n):
    print(n * 2)

# Programa principal:
resultado = doble(5)
print("Resultado guardado:", resultado)
```



¿Qué pasa después de un `return` en una función?

Cuando una función en Python ejecuta una instrucción `return`, **inmediatamente detiene su ejecución y sale** de la función. Esto significa que **ninguna línea que venga después del `return` será ejecutada**, incluso si está escrita dentro de la función.

Este comportamiento es importante, porque a veces uno puede olvidar que todo lo que está después del `return` es ignorado. Por eso, **si necesitas que algo se ejecute, debe ir antes del `return`**.

Por ejemplo:

```
def saludo(nombre):
    return "Hola " + nombre
    print("Esto no se mostrará")

resultado = saludo("Sofía")
print(resultado)
```

¿Qué va a pasar aquí?

- La función `saludo` recibe el nombre `"Sofía"`.
- Ejecuta `return "Hola Sofía"` y **termina inmediatamente**.
- La línea `print("Esto no se mostrará")` **no se ejecuta**, porque está después del `return`.
- Finalmente, el programa imprime el resultado de la función, que es `"Hola Sofía"`.

Salida:

```
Hola Sofía
```

Veamos si es que entendimos la diferencia entre `print` y `return`:

1. Analiza:

```
def saludar(nombre):
    print("Hola", nombre)

mensaje = saludar("Martín")
print("Mensaje guardado:", mensaje)
```

- a. ¿Qué imprime este programa?
 - b. ¿Cuál es el valor de la variable `mensaje` ?
 - c. Comprueba tus respuestas copiando y ejecutando el código.
2. Este código da un error. ¿Por qué?

```
def cuadrado(n):
    print(n * n)

resultado = cuadrado(6)
doble_resultado = resultado * 2
```

- a. Reescribe la función **para que funcione bien usando `return`**.

Actividad 4: Ejercitemos

1. Completa la función para que retorne el promedio de 3 notas:

```
def promedio(n1, n2, n3):
    # tu código aquí
```



Recuerda comprobar que tu función está correcta llamándola, por ejemplo, de la siguiente manera:

```
print(promedio(100, 95, 73))
```

2. Completa la función para que se retorne `True` cuando se ingresa un número par, y `False` en caso contrario:

```
def es_par(n):
    if _____:
        return True
    else:
        _____
```

3. Crea una función que retorne la cantidad de dígitos de un número. Por ejemplo, si como parámetro se le pasa 1982, debe retornar 4.



El nombre de una función debe representar qué es lo que hace o calcula. En este caso, debes crear una función y decidir qué nombre ponerle. Recuerda usar un nombre que tenga sentido 🤔

▼ 🔍 Pista

Para este ejercicio podría ser muy útil transformar el número a un string.

4. Imagina que estás haciendo un programa para ayudar al portero de un centro de eventos, que debe decidir si pueden entrar los clientes a una fiesta, y hasta ahora va así:

```
def puede_entrar(edad):
    return edad >= 18

edad = int(input("Ingrese la edad: "))
if puede_entrar(edad):
    print("Puede entrar")
else:
    print("No puede entrar")
```

Te avisaron que además debes distinguir a 2 clientes VIP, cuyos nombres son **"Ana"** y **"Juan"**. Por esto es que debes modificar tu programa de la siguiente manera:

- a. Crea una función llamada **es_vip(nombre)** que retorne True si el nombre es **"Ana"** o **"Juan"**.

- b. Modifica el programa principal para que también pregunte el nombre del cliente, y usando la función `es_vip(nombre)` verifique si los clientes mayores de edad son VIP o no. Si es que el cliente es VIP debe imprimir "Es un cliente VIP".
5. Una empresa quiere crear un sistema para saber si un jugo es saludable o no. Un jugo es saludable si:

- No tiene azúcar agregada.
- Tiene menos de 100 calorías por porción.

Escribe un programa que le pregunte al usuario dos cosas:

- Si el jugo tiene azúcar (responde `'sí'` o `'no'`)
- Cuántas calorías tiene una porción

Y que determine si el jugo es saludable, usando una función `es_saludable(azucar, calorias)`.

Además, si el jugo es saludable, el programa debe calcular cuántas calorías tendría si se tomaran 3 porciones, usando una segunda función `calorias_totales(porcion, cantidad)`.

Guíate por los siguientes ejemplos de ejecución:

```
Ingrese si el jugo tiene azúcar agregada: sí  
Ingrese las calorías de una porción: 87  
El jugo no es saludable
```

```
Ingrese si el jugo tiene azúcar agregada: no  
Ingrese las calorías de una porción: 45  
El jugo es saludable  
Si te tomas 3 porciones, serán 135 calorías en total
```

```
Ingrese si el jugo tiene azúcar agregada: no  
Ingrese las calorías de una porción: 102  
El jugo no es saludable
```

6. En una tienda se ofrecen los siguientes descuentos:

- Si el precio es mayor a \$20.000, se aplica un **20% de descuento**.

- Si está entre \$10.000 y \$20.000, se aplica un **10% de descuento**.
- Si es menor a \$10.000, **no hay descuento**.

Crea una función `calcular_descuento(precio)` que retorne el valor del descuento, y una función `precio_final(precio, descuento)` que entregue el precio después del descuento.

Luego, escribe el programa principal que le pida ingresar precios de productos a un cliente hasta que ingrese como precio -1. Por cada uno de los productos, el programa usando las 2 funciones creadas le debe mostrar el precio final con el descuento aplicado, si es que el producto lleva descuento. Al final, se debe mostrar el monto total a pagar acumulado.

Guíate por el siguiente ejemplo de ejecución:

```

Ingresa el precio del producto: 27000
Precio final: 21600
Ingresa el precio del producto: 13990
Precio final: 12591
Ingresa el precio del producto: 4500
Precio final: 4500
Total a pagar: 38691

```

7. Aproximación del Seno y Coseno:

La funciones seno y coseno puede ser representadas mediante sumas infinitas:

$$\begin{aligned} \text{sen}(x) &= \frac{x^1}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots \\ \cos(x) &= \frac{x^0}{0!} - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots \end{aligned}$$

(Estas son las series de Taylor en torno a $x = 0$ de seno y coseno, que van a conocer de Mate 2).

Los términos de ambas sumas son cada vez más pequeños, por lo que tomando algunos de los primeros términos es posible obtener una buena aproximación.

- Escribe la función `factorial_reciproco(n)`, que retorne el valor de $1/(n!)$
- Escribe la función `signo(n)` que retorne `1` cuando n es par y `-1` cuando n es impar.
- Escribe las funciones `seno_aprox(x, m)` y `coseno_aprox(x, m)` que aproximen respectivamente el seno y el coseno usando los `m` primeros términos de las sumas correspondientes. Las funciones **deben** llamar a las funciones `factorial_reciproco` y `signo`.

8. Cadenas de ADN:

Un grupo de biotecnólogos externos a la USM, tienen demasiado trabajo analizando cadenas de ADN. Todo el trabajo lo realizan a mano ya que no tuvieron un buen ramo de programación. Por esto, le piden a los estudiantes del ramo de Programación que les ayuden.

Las cadenas de ADN están compuestas por 4 bases nitrogenadas (A: Adenina, C: Citosina, G: Guanina y T: Timina) agrupadas en bloques de 4.

Te piden ayuda en las siguientes operaciones:

- Escribe la función `valida(cadena)` que reciba un **string** con una cadena de ADN y **retorne** `True` si la cadena es válida o `False` si no lo es. Una cadena no es válida cuando aparecen bases nitrogenadas distintas a las antes descritas (asumir que el formato de cuatro letras seguidas de un espacio siempre se cumple). Guíate por el siguiente ejemplo:

```
>>> valida('CTGA CTGA AATT GGGC CTGG CCCC')
True
>>> valida('CTGA XCGA CGAT GGTA ACCC CCPC TTAA')
False
```

- Escribe la función `cantidad(cadena, base)` que reciba un string con una cadena de ADN y una base nitrogenada. La función debe retornar la cantidad de apariciones de la base en la cadena. Guíate por el siguiente ejemplo:

```
>>> cantidad('CTGA CTGA AATT GGGC CTGG CCCC', 'A')
4
```

- Los científicos encontraron un patrón de clasificación, que se deduce de la cantidad mayoritaria de un cierto par de bases:

- Si la suma de las cantidades de Citosina y Guanina es mayor a la de Adenina y Timina, es una especie vegetal.
- En caso contrario es una especie animal.

Escribe un programa que pregunte la cantidad de cadenas de ADN a evaluar, luego solicite las cadenas y finalmente muestre las cantidades de cada especie y cadenas no válidas.

El programa debe hacer uso de las funciones `valida(cadena)` y `cantidad(cadena, base)`.

Guíate por el siguiente ejemplo de ejecución:

```
Cantidad de cadenas de ADN: 3
Ingrese cadena 1: CGTA CAGT TTGG GGTA AATG CATG
Ingrese cadena 2: CACC CTGA GGAA ACAA XTFC ATGG
Ingrese cadena 3: TGTG TTGA ATGA CTAT ATTT
Cantidad animales: 2
Cantidad vegetales: 0
Cantidad no validas: 1
```

Parte 2: Desafíos

Desafío 1: El mejor censista de Chile

El glorioso señor Bodoque, un muy famoso censista chileno, está haciendo su turno laboral. Como cualquier censista, debe viajar por todo Chile en un solo día para realizar el censo de este año.

En la hoja del censo se requiere anotar al miembro mayor y al miembro menor de la familia. Sin embargo, Bodoque no es muy bueno con las matemáticas.



Figura 1: Bodoque en su hábitat natural

Mágicamente, todas las familias que Bodoque tiene que visitar hoy se componen de 3 personas.

Escribe un programa que implemente la función `max_familia(a, b, c)`, que retorne la edad del miembro mayor, y la función `min_familia(a, b, c)`, que retorne la edad del miembro menor, donde `a`, `b` y `c` son las edades de cada miembro de la familia.

Guíate por el siguiente ejemplo:

```
max_familia (66, 50, 23) #66  
min_familia (66, 50, 23) #23
```

Desafío 2: El viaje de dominio interestelar de Reinhardt

En un futuro distante alrededor del siglo XXXV, la Vía Láctea se encuentra repleta de mundos terraformados, habitados por seres humanos que viajan interestelarmente.

Reinhardt, un ambicioso genio militar, pretende unificar a toda la galaxia bajo su control. Cierto día del 2402 se descubre el primer planeta extrasolar habitable, que se encuentra en la órbita de la estrella Canopus.



Figura 2: El puente de mando de la nave espacial de Reinhardt

La flota de los científicos de Reinhardt tiene que viajar a este planeta para estudiarlo.

Ellos estiman que la distancia al planeta queda definida como:

$$distancia = \sqrt{\frac{(2x^2 + 4x^4 + 3x^8)}{y^2}}$$

donde x e y son las coordenadas del planeta.

- a. Escribe la función `distancia_galactica(x, y)`, que reciba como parámetros las coordenadas x e y de un planeta, y retorne la distancia a ese planeta redondeada al tercer decimal.

▼ 12 34 ¿No recuerdas cómo redondear un número?

La función de redondeo `round(x)` redondea el número x al entero más cercano:

```
redondeo = round(-57.6) #redondeo = -58
```

También se la puede utilizar como `round(x, y)` para redondear el número x con una cantidad de cifras y de decimales específica:

```
redondeo = round(1.35678, 2) #redondeo = 1.36
```

- b. Escribe un programa que reciba las coordenadas de un planeta extrasolar y muestre en pantalla la distancia a este utilizando la función `distancia_galactica(x, y)`. Guíate por el siguiente ejemplo de ejecución:

Ingrese la coordenada x: 3

Ingrese la coordenada y: 7

Distancia es: 20.216

- c. Como la misión de la tropa de Reinhardt fue tan exitosa estudiando el primer planeta extrasolar habitable, deciden realizar múltiples investigaciones al mismo tiempo en varios otros planetas candidatos.

Para eso te piden modificar tu programa: ahora debe recibir de una en una las coordenadas de diferentes planetas hasta que se ingrese la coordenada $(0,0)$. Por cada planeta, se debe imprimir la distancia correspondiente. Guíate por el siguiente ejemplo de ejecución:

```
Ingrese la coordenada x: 6
Ingrese la coordenada y: 8
Distancia es: 280.739
Ingrese la coordenada x: 2
Ingrese la coordenada y: 3
Distancia es: 9.661
Ingrese la coordenada x: 1
Ingrese la coordenada y: 1
Distancia es: 3.0
Ingrese la coordenada x: 4
Ingrese la coordenada y: 9
Distancia es: 49.399
Ingrese la coordenada x: 0
Ingrese la coordenada y: 0
¡Éxito en sus viajes!
```

Desafío 3: Carreras

Carreras es un juego de 2 contrincantes, que deben avanzar en una pista de 100 casillas. Cada jugador, en su turno, tira un dado de 12 caras para avanzar. Dependiendo del número de casilla en el que caiga, puede pasar:

- Si el número de casilla **es múltiplo de 6**, entonces el jugador debe retroceder 2 casillas.
- Si el número de la casilla es primo, se acumula 1 punto de poder. Si se juntan 3 puntos de poder, el jugador podrá lanzar dos veces en su próximo turno.
- Si es que el número de la casilla es distinto a lo señalado, el jugador se queda en esa casilla.

El juego termina cuando un jugador **superá** la casilla 100 al tirar los dados.



Figura 3: El famoso juego de Carreras. No se ve muy emocionante, pero hay ciertas personitas que se lo toman extremadamente en serio...

Un programador alcanzó a avanzar en el programa que simula el juego Carreras, pero no sabe cómo utilizar funciones para terminarlo. Te envió su código actual y dejó comentado todos los lugares en los que necesita que lo ayudes escribiendo y llamando a funciones.

Copia y pega el siguiente código, y luego **complétalo creando y usando las funciones** `lanzar_dado()` que retorna un número aleatorio entre 1 y 12, `multiplo_6(n)` que retorna si `n` es múltiplo de 6 o no, y `primo(n)` que retorna si `n` es un número primo o no:

```

posicion1 = 0
posicion2 = 0
punto_poder1 = 0
punto_poder2 = 0
while posicion1 <= 100 and posicion2 <= 100:
    print("-----")
    print("Turno de jugador 1: ")
    jugada = input("Apreta enter para lanzar los dados!")
    dado = #utilizar función lanzar_dados para lanzar los dados
    posicion1 += dado
    print("Te ha salido", dado)
    if #utilizar función multiplo_6 para verificar si posicion1 es múltiplo de 6:

```

```

print("Quedas en la posición", posicion1)
posicion1 -= 2
print("Caes en un múltiplo de 6, retrocedes 2 casillas.")
print("Quedas en la posición", posicion1)
elif #utilizar función primo para ver si posicion1 es primo:
    punto_poder1 += 1
    print("Quedas en la posición", posicion1)
    print("Caes en un primo, acumulas 1 punto de poder. Tienes", punto_poder1)
if punto_poder1 == 3:
    punto_poder1 = 0
    print("Has acumulado 3 puntos de poder, tiras los dados nuevamente")
    jugada = input("Apreta enter para lanzar los dados!")
    dado = #utilizar función lanzar_dados para lanzar los dados
    posicion1 += dado
    print("Te ha salido", dado)
    print("Quedas en la posición", posicion1)
else:
    print("Quedas en la posición", posicion1)

if posicion1 <= 100:
    print("-----")
    print("Turno de jugador 2: ")
    jugada = input("Apreta enter para lanzar los dados!")
    dado = #utilizar función lanzar_dados para lanzar los dados
    posicion2 += dado
    print("Te ha salido", dado)
    if #utilizar función multiplo_6 para verificar si posicion2 es múltiplo de 6:
        print("Quedas en la posición", posicion2)
        posicion2 -= 2
        print("Caes en un múltiplo de 6, retrocedes 2 casillas.")
        print("Quedas en la posición", posicion2)
    elif #utilizar función primo para ver si posicion2 es primo:
        punto_poder2 += 1
        print("Quedas en la posición", posicion2)
        print("Caes en un primo, acumulas 1 punto de poder. Tienes", punto_poder2)
        if punto_poder2 == 3:
            punto_poder2 = 0
            print("Has acumulado 3 puntos de poder, tiras los dados nuevamente")

```

```

jugada = input("Apreta enter para lanzar los dados!")
dado = #utilizar función lanzar_dados para lanzar los dados
posicion2 += dado
print("Te ha salido", dado)
print("Quedas en la posición", posicion2)

else:
    print("Quedas en la posición", posicion2)

print("-----")
print("FIN DEL JUEGO")
if posicion1 > 100:
    print("Jugador 1 gana")
else:
    print("Jugador 2 gana")

```

Para probar el programa final, guíate con el siguiente ejemplo de ejecución:

```

-----
Turno de jugador 1:
Apreta enter para lanzar los dados!
Te ha salido 3
Quedas en la posición 3
Caes en un primo, acumulas 1 punto de poder. Tienes 1 puntos
-----

Turno de jugador 2:
Apreta enter para lanzar los dados!
Te ha salido 3
Quedas en la posición 3
Caes en un primo, acumulas 1 punto de poder. Tienes 1 puntos
-----

Turno de jugador 1:
Apreta enter para lanzar los dados!
Te ha salido 2
Quedas en la posición 5
Caes en un primo, acumulas 1 punto de poder. Tienes 2 puntos
-----

Turno de jugador 2:
Apreta enter para lanzar los dados!

```

Te ha salido 8

Quedas en la posición 11

Caes en un primo, acumulas 1 punto de poder. Tienes 2 puntos

Turno de jugador 1:

Apreta enter para lanzar los dados!

Te ha salido 8

Quedas en la posición 13

Caes en un primo, acumulas 1 punto de poder. Tienes 3 puntos

Has acumulado 3 puntos de poder, tiras los dados nuevamente

Apreta enter para lanzar los dados!

Te ha salido 11

Quedas en la posición 24

Turno de jugador 2:

Apreta enter para lanzar los dados!

Te ha salido 1

Quedas en la posición 12

Caes en un múltiplo de 6, retrocedes 2 casillas.

Quedas en la posición 10

...

Desafío 4: Ajedrez Mágico

Harry, Hermione y Ron se encuentran en una situación muy peligrosa: la tercera prueba de la Piedra Filosofal. Para superarla, deben ganar una partida de ajedrez mágico.

Las piezas del juego se controlan mediante comandos de voz. Cabe destacar que el tablero gigante, además de tener N filas y N columnas, se divide en casillas negras y casillas blancas. Para ganar la partida es importante seguir las reglas, lo que incluye moverse sólo por las casillas del color correcto.



Figura 4: La tercera prueba de la piedra filosofal

Para ayudar a Harry, Hermione y Ron, decides hacer un programa que les permita identificar de qué color es la casilla a la que se quieren mover. Considera que cada casilla de un tablero de ajedrez es representada por una coordenada (fila, columna). Por ejemplo, la casilla (1, 2) es de color blanco, y la casilla (6, 4) de color negro.

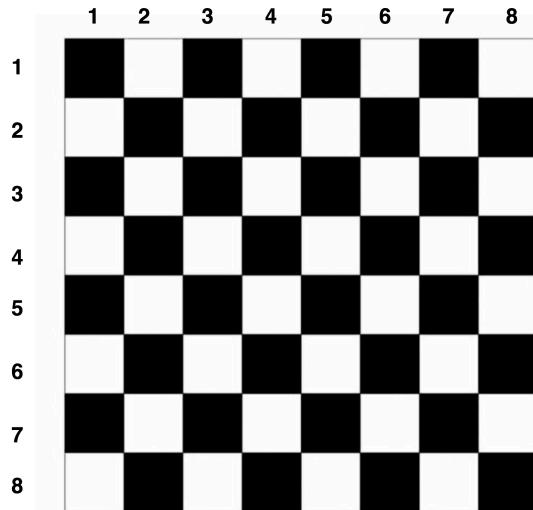


Figura 5: Un tablero de tamaño 8×8 .

- Escribe la función `color_casilla(fila, columna)`, que retorne "Blanco" o "Negro" dependiendo del color de la casilla con coordenadas (`fila`, `columna`).



Tip: ¿Hay alguna propiedad que indique el color de las casillas?
Revisa las casillas (1,1), (1,3), (2,2) y (4,4).

▼ **Pista**

Si la fila es impar, las columnas impares son negras.

Si la fila es par, las columnas pares son negras.

Para probar tu función guíate con los siguientes ejemplos:

```
color_casilla(2, 2)
```

'Negro'

```
color_casilla(5, 6)
```

'Blanco'

- b. Escribe un programa asumiendo un tablero general (tamaño `N` desconocido). El programa debe recibir uno por uno el número de una fila y el número de una columna. Por cada coordenada ingresada debe mostrar en pantalla de qué color es la casilla indicada (usando la función `color_casilla(fila, columna)`). El programa debe terminar si alguno de los números ingresados es negativo.

Guíate con el siguiente ejemplo de ejecución:

```
Ingrese la fila: 2
```

```
Ingrese la columna: 7
```

Blanco

```
Ingrese la fila: 6
```

```
Ingrese la columna: 6
```

Negro

```
Ingrese la fila: 4
```

```
Ingrese la columna: 3
```

Blanco

```
Ingrese la fila: -1
```

```
Ingrese la columna: -1
```

Terminando!