

▼ Clase 2: Funciones y Receta de Diseño

Capítulo 2: Funciones

(Leer capítulo 2 del apunte para reforzar)

Hoy veremos como hacer funciones en Python.

Igual que en matemáticas, **las funciones en computación es una estructura que recibe valores de entrada y genera valores de salida.**

Las funciones nos **permiten encapsular tareas** o funcionalidades para así reutilizar el código sin tener que escribirlo nuevamente.

Ejemplo:

Cómo calcular el area de un círculo?

Conocemos bien la **relación entre el área de un círculo** y su **radio**. El área de un círculo está dado por su radio r . Entonces se calcula mediante la expresión:

```
r=5
pi=3.14
area=pi*r**2
```

```
area
```

si queremos calcular con $r = 2$?

```
#...completar
```

```
#...completar
```

- ▼ Hagamos una función que permita hacer esto para no tener que reescribir el código cada vez.

Es importante que cada función tenga un nombre que esté directamente relacionado con el objetivo que cumple, al igual que sus parámetros.

Palabras para declarar una función empezamos con `def` y terminamos con `:`.

Luego, todas **las instrucciones que pertenecen a la función deben ir indentadas** adentro de ésta. Esto **indica subordinación** a la instrucción que está un nivel más afuera.

Además, la palabra **return** indica el fin de la función.

```
def areaCirculo(r):  
    pi=3.14  
    area = pi*r**2  
    return area
```

la invocamos:

```
areaCirculo(5)
```

```
areaCirculo(12)
```

funciona para cualquier número, no tenemos que repetir código.

▼ Sintaxis de una función

```
def nombre(parametros):  
    instrucciones (cero o mas)  
    return expresion
```

- parametros
 - sintaxis: nombre, ... (cero o mas)
 - instrucciones y return se escriben indentadas en un margen/nivel más adentro que encabezamiento

```
def areaCirculo(r):  
    pi=3.14  
    return pi*r**2
```

- contraejemplo:

```
def f(x):  
return x
```

▼ Funciones pueden contener funciones

Una función puede estar compuesta de operadores básicos, variables y también de otras funciones que hayan sido definidas antes. Ej. Implementar una nueva función llamada:

```
areaAnillo(r_exterior, r_interior)
```

- Paso 1: Definir el algoritmo o pasos a seguir.
- Paso 2: Definir la nueva función.

```
def areaAnillo(r_exterior, r_interior):  
    return #...completar
```

```
areaAnillo(5,3)
```

▼ Indentación y subordinación de instrucciones

- cada indentación que separa la instrucción del margen izquierdo, indica subordinación a la función de más afuera.

```
def areaCirculo(radio):  
    pi = 3.14  
    return pi * radio **2
```

▼ Alcance de las variables

Ejemplo:

```
a = 100  
def sumaValorA(x):  
    return x + a  
  
sumaValorA(1)
```

Si redefinimos `a` adentro de la función, solo se modifica una copia local de la variable `a` y no a nivel global. Ejemplo:

```
a = 100  
def sumaValorA(x):  
    a = 5  
    return x + a
```

```
sumaValorA(a)
```

```
a # esto muestra el valor global de a (es decir afuera de la función)
```

Por otra parte, si el argumento de una función tiene el **mismo nombre que una variable definida fuera** de ésta, la función evaluará sus instrucciones con el valor del argumento, pues es la variable que está dentro de su alcance. Ejemplo:

```
a = 100
x = 50
def sumaValorA(x): # no importa qué valor tenía antes x
    return x + a

sumaValorA(4)
```

```
a = 100
x = 50
def sumaValorA(): # no definimos parametros de entrada
    return x + a

sumaValorA()
```

- ¿Qué pasa si llamo la función con el número incorrecto de parámetros?

```
sumaValorA(5,6)
```

▼ ¿Cómo traducir un problema expresado en palabras a una función?

Genera S.A. le paga 4.500 por hora a todos sus ingenieros de procesos recién egresados. Un empleado típicamente trabaja entre 20 y 65 horas por semana. La gerencia de informática le pide desarrollar un programa que calcule el sueldo de un empleado a partir del número de horas trabajadas.

Debemos fijarnos en 2 cosas:

- Qué debe producir la función?
- Qué recibe como parámetros?

```
def sueldo(horas):
    return 4500*horas
```

▼ Capítulo 3: Receta de diseño (Leer capítulo 3 apunte del curso)

Es una receta para escribir correctamente funciones. Se preocupa de ayudarnos a extraer la información importante de un problema y entenderlo. Se compone de la siguiente manera:

1. Entender el **propósito** de la función: ¿para qué sirve?
2. Dar **ejemplos** de uso de la función: ¿cómo se usa?
3. **Probar** la función: hacer tests
4. **Especificar** el cuerpo de la función: ahora escribimos la función

Entender el propósito de una función

Objetivo: diseñar una función que consume y produce información.

- Entonces, debemos darle a la función **un nombre significativo** que indique *qué es lo que hace*.
- Además, **indicar qué información consume** y **qué información produce**.
- Esto se llama el **contrato** de una función.
 - Por ahora los tipos de datos que consume y produce una función son los que conocemos (después agregaremos más): `int`, `float`, `num` y `str`.
- Luego escribimos el **propósito** de la función.
- Después agregamos **ejemplos** de cómo debe funcionar la función, que debemos calcular antes de escribir la función (a mano).
- Luego agregamos **tests**, utilizando la función `assert`.

TODO ESTO SE HACE ANTES DE ESCRIBIR LA FUNCIÓN

Ejemplo:

```
# areaRectangulo: num num -> num
# calcula el area de un rectangulo de medidas
# largo y ancho
# ejemplo: areaRectangulo(5, 3) debe producir 15
def areaRectangulo(largo, ancho):
    return largo * ancho

# Tests
assert areaRectangulo(5, 3) == 15
```

```
# areaCuadrado: num -> num
# calcula el area de un cuadrado de medida lado
# ejemplo: areaCuadrado(5) debe producir 25
def areaCuadrado(lado):
```

```
#...completar

# Tests

assert #...completar
```

En resumen hemos aprendido hasta ahora:

- A hacer funciones (cada función soluciona un problema acotado)
 - Entender cuál es el ámbito de las variables
 - Funciones de funciones
 - Crear recetas de diseño (pensar en abstracto sobre qué es lo que queremos hacer y luego aterrizarlo)

Funciones principales y funciones auxiliares

```
def areaAnillo(interior , exterior): # Buena practica
    return areaCirculo(exterior) - areaCirculo(interior)

def areaAnillo(interior , exterior): # Mala practica
    return 3.14 * exterior ** 2 - 3.14 * interior ** 2
```

debemos considerar el descomponer el problema en funciones, y éstas a su vez descomponerlas en funciones auxiliares hasta que cada una de ellas resuelva UN Y SOLO UN SUBPROBLEMA particular.

▼ Problema (hacer de manera individual):

“Una importante cadena de cines de Santiago tiene completa libertad en fijar los precios de las entradas. Claramente, mientras más cara sea la entrada, menos personas estarán dispuestas a pagar por ellas. En un reciente estudio de mercado, se determinó que hay una relación entre el precio al que se venden las entradas y la cantidad de espectadores promedio: a un precio de \$5.000 por entrada, 120 personas van a ver la película; al reducir \$500 en el precio de la entrada, los espectadores aumentan en 15. Desafortunadamente, mientras más personas ocupan la sala para ver la película, más se debe gastar en limpieza y mantenimiento general. Para reproducir una película, el cine gasta \$180.000. Asimismo, se gastan en promedio \$40 por espectador por conceptos de limpieza y mantenimiento. El gerente del cine le encarga determinar cuál es la relación exacta entre las

ganancias y el precio de las entradas para poder decidir a qué precio se debe vender cada entrada para maximizar las ganancias totales.”

Cuando nos vemos enfrentados a estas situaciones, **lo mejor es identificar las dependencias** y ver las relaciones una por una:

- Las **ganancias** corresponden a la diferencia entre los ingresos y los gastos.
- Los **ingresos** se generan exclusivamente a través de la venta de entradas. Corresponde al producto del valor de la entrada por el número de espectadores.
- Los **gastos** están formados por dos ítemes: un gasto fijo (\$180.000) y un gasto variable que depende del número de espectadores.
- Finalmente, el enunciado del problema también especifica cómo el número de espectadores depende del precio de las entradas.

Antes de escribir código, hay que formular el contrato, encabezado y propósito de la función, ejemplos y casos de prueba

```
# ganancias: int -> int
# calcular las ganancias como la diferencia entre los ingresos y
# los gastos dado precioEntrada
def ganancias(precioEntrada):

# ingresos: int -> int
# calcular el ingreso total, dado precioEntrada
def ingresos(precioEntrada):

# gastos: int -> int
# calcular los gastos totales, dado precioEntrada
def gastos(precioEntrada):

# espectadores: int -> int
# calcular el numero de espectadores, dado precioEntrada
def espectadores(precioEntrada):

def ganancias(precioEntrada):
    return #...completar

def ingresos(precioEntrada):
    return #...completar

def gastos(precioEntrada):
    return #...completar

def espectadores(precioEntrada):
    return 120 + (5000 - precioEntrada) * 15 / 500
```

Ejercicio 1:

Ahora que revisaste todo el material ¡No olvides realizar el **Ejercicio publicado en [U-Cursos!](#)**