
PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

# INTRODUCCIÓN A LA PROGRAMACIÓN


# Funciones y módulos

- Objetivos:
  - Recordatorio de funciones y módulos
  - Sobre **`return`**
  - Algunos programas de ejemplo:
    - El número secreto
    - Cachipún
    - El ahorcado

# Recordatorio: las funciones

- Son subprogramas: tienen entradas (argumentos) y generan una salida (valor retornado)
- Nos permiten **reutilizar código**
- Programación estructurada:
  - Ejecución de código sólo en algunas ocasiones: **alternativas**
    - Todo lo que sería **if-elif-else**
  - Repetir código de forma consecutiva: **repeticiones**
    - Todo lo que sería **for** y **while**
  - Reutilizar código en distintos lugares: **funciones**
    - Se definen con **def**, pero se invocan por su nombre

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

# Recordatorio: funciones ya vistas

- **`int(x)`**: recibe un valor e intenta convertirlo en un número entero (tipo int)
- **`float(x)`**: recibe un valor e intenta convertirlo en un número decimal (tipo float)
- **`str(x)`**: recibe un valor e intenta convertirlo en un texto (tipo str)
- **`input()`**: entrega un texto desde la *entrada estándar* (ej. el teclado)
- **`print(x1, x2, …)`**
- **`round(x,d)`**: redondea el número **x**, dejando sólo **d** decimales
- **`abs(x)`**: obtiene el *valor absoluto* de **x**
- **`max(x1, x2, …)`**: retorna el número más grande (el *máximo*) de los números x1, x2, ..., etc
- **`min(x1, x2, …)`**: retorna el número más pequeño (el *mínimo*) de los números x1, x2, ..., etc

# Recordatorio: nuevas funciones

Módulo: **`math`**

- Modulo con funciones matemáticas
- **`math.sqrt(x)`**: obtiene la raíz cuadrada de **x**
- Otras funciones (*no canon*)
  - `math.log`
  - `math.exp`
  - `math.cos`
  - `math.sin`

Módulo: **`random`**

- Módulo con funciones que generan números aleatorios
- **`random.randint(a,b)`**: genera un entero aleatorio entre los enteros `a` y `b`, extremos incluidos
- **`random.random()`**: genera un flotante entre 0 y 1, aleatorio

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

# Recordatorio: funciones propias

Las funciones se definen utilizando la instrucción def de la siguiente manera:

```
def nombre_de_la_función( argumento1, argumento2, … ):
    bloque de instrucciones
    (o mini procedimiento)
    que realiza lo que la
    función debe hacer
    return resultado
```

Las variables argumento1, argumento2, etc, se definen cada vez que se invoca a la función. La instrucción return permite indicar qué valor retornará la función.

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

# Recordatorio: funciones de funciones

```
def factorial( n ):
    resp = 1
    for k in range( 2, n+1 ):
        resp = resp * k
    return resp
```

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}$$

```
def combinacion( n, r ):
    numer = factorial( n )
    denom = factorial( r ) * factorial( n-r )
    return numer / denom

print( combinacion(8,2) )
print( combinacion(11,3) )
```


# Recordatorio: módulos propios

*¿Cómo creamos un módulo?*

Simplemente tenemos que guardar nuestro programa con su extensión **`.py`**.

*¿Cómo cargamos un módulo desde otro?*

Escribiendo **`import mi_modulo`**, si es que nuestro módulo se llamó **`mi_modulo.py`**.

*¿Cómo usamos una función de un módulo?*

Si la función se llama **`mi_funcion()`** y está en **`mi_modulo.py`**, entonces podemos llamar a **`mi_funcion`** escribiendo **`mi_modulo.mi_funcion()`** después de haber hecho **`import mi_modulo`**.


# Sobre
# `return`

# Más sobre `return`

- Permite indicar qué valor será retornado
- Una vez que **`return`** se ejecuta, la función se cierra y el valor se devuelve al contexto que llama
  - ¿Si hay return dentro de `if`? No importa, se acaba el `if`
  - ¿Si hay return dentro de `for`? No importa, se acaba el `for`
  - ¿Si hay return dentro de `while`? No importa, se acaba el `while`
- La instrucción `return` puede ir *vacía*, o sea, sin valor adjunto
  - Escribir **`return`** a secas es equivalente a escribir **`return None`**
- Si la función no lleva `return`, se retorna **`None`**

# Pregunta tipo control

Considere el siguiente programa en Python. ¿Qué es lo que imprime?

```
def misterio(z):
  if z < 0:
    return -1
  elif z > 0:
    return 1
  elif z == "hola":
    return "mundo"

x = misterio(17)
print(x)
```

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

# Pregunta tipo control

Considere el siguiente programa en Python. ¿Qué es lo que imprime?

```
def misterio(z):
  if z < 0:
    return -1
  elif z > 0:
    return 1
  elif z == "hola":
    return "mundo"

x = misterio(-5)
print(x)
```

# Pregunta tipo control

Considere el siguiente programa en Python. ¿Qué es lo que imprime?

```
def misterio(z):
  if z < 0:
    return -1
  elif z > 0:
    return 1
  elif z == "hola":
    return "mundo"

x = misterio(0)
print(x)
```

# Pregunta tipo control

Considere el siguiente programa en Python. ¿Qué es lo que imprime?

```
def misterio(z):
  if z < 0:
    return -1
  elif z > 0:
    return 1
  elif z == "hola":
    return "mundo"

x = misterio("hola")
print(x)
```

# Pregunta tipo control

Considere el siguiente programa en Python. ¿Qué es lo que imprime?

```
def misterio(u,v):
  for i in range(v):
    if i==u:
      return u
    else:
      return v

x = misterio(5,7)
print(x)
```


# Programando


# Programando

Veamos aplicaciones prácticas de funciones y módulos para escribir programas (juegos)

Juegos a implementar:

- **Adivine el Número Secreto**
- **El Cachipún**
- **El Ahorcado**