PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

INTRODUCCIÓN A LA PROGRAMACIÓN

Funciones y módulos

- Objetivos:
  - Presentar las nociones generales tras las **funciones**
  - Recordar funciones vistas
  - Cómo escribir nuestras propias funciones
  - Por qué usarlas
  - Introducción a los módulos
  - Cómo crear módulos

Nociones generales de funciones

Funciones

Una **función** es un trozo de código que tiene:

- Inputs: también llamados argumentos o parámetros (van dentro de los paréntesis)
  - Es posible que una función reciba cero argumentos
- Output: el valor que es retornado (return) por la función

Una función es un **mini-programa**, pues tiene los mismos elementos de un algoritmo. También y debido a esto, a veces se le llama **subprograma** o **subrutina** (aunque estos términos son más generales).

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Funciones ya conocidas (I)

Algunas funciones que ya hemos visto:

- **`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, …)`**: escribe los valores de x1, x2, ..., en la *salida estándar* (ej. la consola o terminal)

Los parámetros o argumentos van en los paréntesis.

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Funciones ya conocidas (II)

Y además hemos visto:

- **`round(x,d)`**: redondea el número **x**, dejando sólo **d** decimales
- **`abs(x)`**: obtiene el *valor absoluto* de **x**, o sea,
  - Si x es negativo, retorna **-x** (ej. **abs(-5)** retorna **5**);
  - Si no, retorna **x** (ej. **abs(4)** retorna **4**).
- **`max(x1, x2, …)`**: retorna el número más grande (el *máximo*) de los números x1, x2, ...,
- **`min(x1, x2, …)`**: retorna el número más pequeño (el *míximo*) de los números x1, x2, ...,

Escribiendo funciones
(implementación)

¿Cómo creamos una función?

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.

Ejemplo

```
def minimo3( x, y, z ):
    resp = x
    if y < resp:
        resp = y
    if z < resp:
        resp = z
    return resp
```

¿Cuáles son los parámetros o argumentos de la función?

¿Qué retorna la función?

¿Qué hace esta función?

Ejemplo

¿Qué imprime el siguiente código?

```
def minimo3( x, y, z ):
    resp = x
    if y < resp:
        resp = y
    if z < resp:
        resp = z
    return resp

u = minimo3( 25, 35, 15 )
print( u )
```

Ejemplo 2

Considere el siguiente código:

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

¿Cuáles son los parámetros o argumentos de la función?
¿Qué retorna la función?
¿Qué hace esta función?

¡IMPORTANTE!

Dentro de las funciones podemos definir variables, tener secuencias if-elif-else, tener ciclos/repeticiones while y for, etc.

```
def minimo3( x, y, z ):
    resp = x
    if y < resp:
        resp = y
    if z < resp:
        resp = z
    return resp
```

Esta función contiene una estructura de **ifs**

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

Esta función contiene una estructura de **for**

Podemos invocar funciones desde otras funciones

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

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

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

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

¿Y para qué sirven?

¿Y para qué sirven las funciones?

- Las funciones 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
- Podemos escribir código de forma **más ordenada**, abordando los problemas **por partes**

Haciendo más legible el código

```
seguir = True
while seguir:
    menu_opciones()
    x = input()
    if x == "salir":
        seguir = False
    elif x == "etc":
        hacer_etc()
```

Estrategias de programación

- **`Divide y Vencerás`**. Dividir el problema en subproblemas o partes más simples de resolver.
- ***`KISS`*`: Keep It Short & Simple`**. El código debe verse y estar escrito de forma simple. No hay que complicarse demás.
- ***`YAGNI`*`: You Ain't Gonna Need It`**. No agregar cosas que no se van a usar.
- ***`DRY`*`: Don't Repeat Yourself`**. Usar las funciones para evitar tener que escribir código idéntico varias veces.

Introducción a los Módulos

Módulos

¿Qué hace el siguiente código?

```
import math

print( math.sqrt(1) )
print( math.sqrt(4) )
print( math.sqrt(9) )
print( math.sqrt(16) )
```

Módulos

¿Qué hace el siguiente código?

```
import math

print( "Ingrese un numero:" )

a = float( input() )
b = math.sqrt( a )

print( "La raiz cuadrada de", a, "es", b )
```

Módulos

¿Qué hace el siguiente código?

```
import math

print( "Ingrese un numero positivo:" )

a = float( input() )
b = math.log( a )

print( "El logaritmo natural de", a, "es", b )
```

Módulos: random

¿Qué hace el siguiente código?

```
import random

print( "Diez numeros enteros entre 1 y 10:" )

for r in range(10):
    u = random.randint( 1, 10 )
    print( u )
```

Módulos: random

¿Qué hace el siguiente código?

```
import random

print( "Diez numeros decimales entre 1 y 10:" )

for r in range(10):
    u = 1 + 9 * random.random()
    print( u )
```

Creando módulos

¿Qué es un módulo?

Un módulo es un simplemente programa en Python.

Podemos **importar** un programa en Python dentro de otro programa en Python, de tal manera que podemos usar sus funciones, variables, etc.

Generalmente se usan como sinónimos:

- Módulo
- Librería
- Biblioteca

Creando módulos

¿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**.

And thus, we are **ready**
to take on Control 2

(September the 3rd)