Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

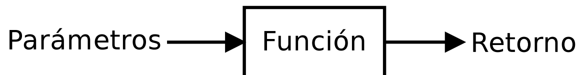
Clase 06: Scopes y Librerías

Rodrigo Toro Icarte (rntoro@uc.cl)

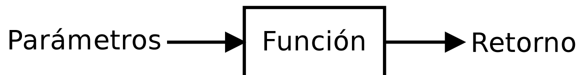
IIC1103 Introducción a la Programación - Sección 5

01 de Abril, 2015

¿Qué aprendimos la clase pasada?



¿Qué aprendimos la clase pasada?



Dos pasos:

- Definir la función.
- Llamar a la función.

¿Qué aprendimos la clase pasada?

1. **Definir la función**: Aquí definimos el comportamiento de la función (su código).

¿Qué aprendimos la clase pasada?

1. **Definir la función**: Aquí definimos el comportamiento de la función (su código).

Sintaxis: definir función

def nombre_función(param_1, param_2, ...)**:**
    inst_1
    ...
    inst_n
    return ret

¿Qué aprendimos la clase pasada?

Ejemplo:

```
"""
Esta función retorna la suma de los dígitos
del número 'n'
"""
def sumar_digitos(n): # <- Función recibe 1 parámetro
  # Código de la función
  suma = 0
  while(n!=0):
    suma+=n%10
    n//=10
  # Retornamos la suma de los dígitos
  return suma
```

Obs: Notar elementos de la función (parámetros, código y retorno).

¿Qué aprendimos la clase pasada?

2. **Llamar función**:
   - Desde tu código puedes *llamar* a funciones ya definidas.
   - Al llamarla debes dar valor a **todos** sus parámetros.
   - La función devolverá su valor de retorno.

¿Qué aprendimos la clase pasada?

2. Llamar función:

- Desde tu código puedes *llamar* a funciones ya definidas.
- Al llamarla debes dar valor a **todos** sus parámetros.
- La función devolverá su valor de retorno.

Sintaxis: llamar a una función

out_1 = nombre_función(in_1, in_2, ...)

¿Qué aprendimos la clase pasada?

Ejemplo:

```
1 a = int(input("Ingrese un número: "))
2 b = int(input("Ingrese otro número: "))
3 s_a = sumar_digitos(a)
4 s_b = sumar_digitos(b)
5 print("La multiplicación es:",s_a*s_b)
```

Observaciones

1. Una función podría no retornar ningún valor.

Observaciones

1. Una función podría no retornar ningún valor.

```
def print_mario():
  print("_▄▄▄▄▄▄▄▄▄▄▄▄░")
  print("▄▀▀▀▀█▌┌────┐░░")
  print("█▄▐▌▄█▄┘██")
  print("└▄▄▄▄▄▄┘█▌")
  print("███▒█▒█████▀")

print_mario()
```

Observaciones

2. print vs return.

Observaciones

2. print vs return.

"Cree una función que encuentre el máximo entre dos números."

```
# Imprime el máximo
def maximo_1(num1,num2):
  if(num1 < num2):
    print(num2)
  else:
    print(num1)
```

```
# Retorna el máximo
def maximo_2(num1,num2):
  if(num1 < num2):
    return num2
  else:
    return num1
```

Observaciones

2. print vs return.

"Cree una función que encuentre el máximo entre dos números."

```python
1 # Imprime el máximo
2 def maximo_1(num1,num2):
3   if(num1 < num2):
4     print(num2)
5   else:
6     print(num1)
```

```python
1 # Retorna el máximo
2 def maximo_2(num1,num2):
3   if(num1 < num2):
4     return num2
5   else:
6     return num1
```

¿Cómo calculo el máximo entre 4 números?

Observaciones

2. print vs return.

"Cree una función que encuentre el máximo entre dos números."

```
# Imprime el máximo
def maximo_1(num1,num2):
  if(num1 < num2):
    print(num2)
  else:
    print(num1)
```

```
# Retorna el máximo
def maximo_2(num1,num2):
  if(num1 < num2):
    return num2
  else:
    return num1
```

¿Cómo calculo el máximo entre 4 números?

```
print(maximo_2(maximo_2(16,10),maximo_2(30,8)))
```

Observaciones

3. Al *retornar*, la función se acaba.

Observaciones

3. Al *retornar*, la función se acaba.

```
"""
Esta función retorna true si n es primo
"""
def es_primo(n):
  # Si n==1 retorno False de inmediato
  if(n == 1):
    return False
  i = 2
  while(i<n):
    # Si encuentro un divisor exacto retorno False
    if(n%i==0):
      return False
    i+=1
  # Si llego acá es porque el número era primo
  return True
```

Debug

Jerga computina:

Debug

Jerga computina:

- Se denomina *bug* (bicho) a un error en un programa.

Debug

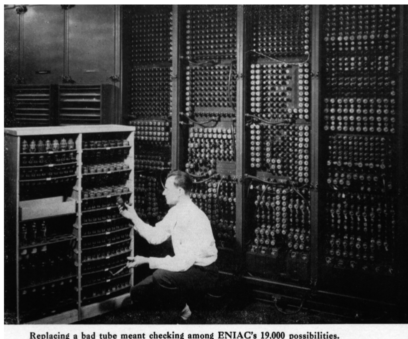
Jerga computina:

- Se denomina *bug* (bicho) a un error en un programa.
- Se conoce como *debuggear* al proceso de encontrar un bug (y arreglarlo).

Debug

Jerga computina:

- Se denomina *bug* (bicho) a un error en un programa.
- Se conoce como *debuggear* al proceso de encontrar un bug (y arreglarlo).



Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

Historia

Debug

Lo interesante: Existen herramientas para debuggear su código en python.

Debug

Lo interesante: Existen herramientas para debuggear su código en python.

Winpdb → http://winpdb.org/

Debug

Lo interesante: Existen herramientas para debuggear su código en python.

Winpdb → http://winpdb.org/

... veamos algunos ejemplos.

Debug

```
1  def es_primo(n):
2    if(n == 1):
3      return False
4    i = 2
5    while(i<n):
6      if(n%i==0):
7        return False
8      i+=1
9    return True
10
11 # sumo primeros "n" primos
12 n = int(input("n: "))
13 total = 0; i = 1
14 while(n!=0):
15   if(es_primo(i)):
16     total += i
17     n-=1
18   i+=1
19 print(total)
```

Debug

```
# Retorna el factorial de "num"
def factorial(num):
  f = 1; i = 1
  while(i < num+1):
    f *= i; i+=1
  return f

# Retorna C(m,n)
def binomial(m,n):
  return factorial(m)/(factorial(m-n)*factorial(n))

# Codigo principal
m = int(input("m: "))
n = int(input("n: "))
# Obtengo resultado final
print(binomial(m,n))
```

Variables (recordatorio)

Sólo se puede usar una variable si ya ha sido definida.

```
a = 4
print(a) # >>> 4

print(b) # >>> NameError: name 'b' is not defined
b = 3

x = 4*c  # >>> NameError: name 'c' is not defined
```

Variables (recordatorio)

¿Qué sucede en este caso?

```
a = int(input("Ingrese número: "))
if(a < 0):
  b = 5
else:
  c = 3

print(b)
print(c)
```

Variables (recordatorio)

¿Qué sucede en este caso?

```
a = int(input("Ingrese número: "))
if(a < 0):
  b = 5
else:
  c = 3

print(b)
print(c)
```

Dependiendo de 'a' fallará en línea 7 ó 8

Variables (recordatorio)

¿Qué sucede en este caso?

```
a = int(input("Ingrese número: "))
if(a < 0):
  b = 5
else:
  c = 3

if(a < 0):
  print(b)
else:
  print(c)
```

Variables (recordatorio)

¿Qué sucede en este caso?

```
a = int(input("Ingrese número: "))
if(a < 0):
  b = 5
else:
  c = 3

if(a < 0):
  print(b)
else:
  print(c)
```

Funciona!

Variables (recordatorio)

Buena práctica: Definir variables que *usaremos* fuera del if.

```
a = int(input("Ingrese número: "))
b = c = 0
if(a < 0):
  b = 5
else:
  c = 3

print(b)
print(c)
```

Scopes

Scope:

- Sección de código independiente.

Scopes

Scope:

- Sección de código independiente.
- Sus variables **NO** son visibles para el resto del programa.

Scopes

Scope:

- Sección de código independiente.
- Sus variables **NO** son visibles para el resto del programa.
- Cada función define su propio scope.

Scopes

"Lo que pasa en una función se queda en una función"

```python
def exp2(num):
  i = num**2
  print(i)

exp2(4)
print(i)    # >>> Error
```

Scopes

"Lo que pasa en una función se queda en una función"

```
1 def exp2(num):
2   i = num**2
3   print(i)
4
5 exp2(4)
6 print(i)    # >>> Error
```

... esto es bueno!

Scopes

Ejemplo:

```
1 def exp2(num):
2   i = num**2
3   print(i)
4 
5 # mostramos exp2 de números entre 1 y 10
6 i = 0
7 while(i < 10):
8   i+=1
9   exp2(i)
```

Scopes

Ejemplo:

```
1 def exp2(num):
2   i = num**2
3   print(i)
4 
5 # mostramos exp2 de números entre 1 y 10
6 i = 0
7 while(i < 10):
8   i+=1
9   exp2(i)
```

¿Qué pasaría si exp2 no definiera su propio scope?

Scopes

Variable local: Puede ser leída y modificada dentro de su scope.

Scopes

Variable local: Puede ser leída y modificada dentro de su scope.

```
1 def exp2(num):
2   i = num**2
3   print(i)
4
5 exp2(4)
6 print(i)    # >>> Error
```

Scopes

Variable local: Puede ser leída y modificada dentro de su scope.

```
def exp2(num):
    i = num**2
    print(i)

exp2(4)
print(i)    # >>> Error
```

Regla 1: Toda variable definida en una función es *local*.

Scopes

Regla 1: Toda variable definida en una función es *local*.

```
def funcion(num):
  ret = num**2
  print(ret)

ret = 5
funcion(4)
print(ret)
```

¿Cuál es la salida del programa?

Scopes

Regla 1: Toda variable definida en una función es *local*.

```
1 def funcion(num):
2   ret = num**2
3   print(ret)
4
5 ret = 5
6 funcion(4)
7 print(ret)
```

¿Cuál es la salida del programa? 16, 5

Scopes

Regla 2: Desde una función se puede leer, pero no modificar, el valor de una variable definida en su *scope superior*.

Scopes

Regla 2: Desde una función se puede leer, pero no modificar, el valor de una variable definida en su *scope superior*.

```
def funcion(num):
  otro = num**2 + ret
  print(otro)

ret = 5
funcion(4)
print(ret)
```

¿Cuál es la salida del programa?

Scopes

Regla 2: Desde una función se puede leer, pero no modificar, el valor de una variable definida en su *scope superior*.

```
def funcion(num):
  otro = num**2 + ret
  print(otro)

ret = 5
funcion(4)
print(ret)
```

¿Cuál es la salida del programa? 21, 5

Scopes

Regla 2: Desde una función se puede leer, pero no modificar, el valor de una variable definida en su *scope superior*.

```
1 def funcion(num):
2   ret += num**2
3   print(ret)
4
5 ret = 5
6 funcion(4)
7 print(ret)
```

¿Cuál es la salida del programa?

Scopes

Regla 2: Desde una función se puede leer, pero no modificar, el valor de una variable definida en su *scope superior*.

```
1 def funcion(num):
2   ret += num**2
3   print(ret)
4
5 ret = 5
6 funcion(4)
7 print(ret)
```

¿Cuál es la salida del programa? Error!

Scopes

Regla 2: Desde una función se puede leer, pero no modificar, el valor de una variable definida en su *scope superior*.

```
def funcion_1(num):
  a = num**2
  print(a)

def funcion_2(num):
  b = a + num
  print(b)

funcion_1(4)
funcion_2(4)
```

¿Cuál es la salida del programa?

Scopes

Regla 2: Desde una función se puede leer, pero no modificar, el valor de una variable definida en su *scope superior*.

```
1  def funcion_1(num):
2      a = num**2
3      print(a)
4
5  def funcion_2(num):
6      b = a + num
7      print(b)
8
9  funcion_1(4)
10 funcion_2(4)
```

¿Cuál es la salida del programa? 16, Error!

Scopes

Regla 2: Desde una función se puede leer, pero no modificar, el valor de una variable definida en su *scope superior*.

```
def funcion_1(num):
  ret = num**2
  print(ret)

def funcion_2(num):
  b = ret + num
  print(b)

ret = 5
funcion_1(4)
funcion_2(4)
```

¿Cuál es la salida del programa?

Scopes

Regla 2: Desde una función se puede leer, pero no modificar, el valor de una variable definida en su *scope superior*.

```
def funcion_1(num):
  ret = num**2
  print(ret)

def funcion_2(num):
  b = ret + num
  print(b)

ret = 5
funcion_1(4)
funcion_2(4)
```

¿Cuál es la salida del programa? 16, 9

Scopes

¿Cómo puedo modificar una variable externa dentro de una función?

Scopes

¿Cómo puedo modificar una variable externa dentro de una función?

Variable global: Puede ser leída y modificada desde cualquier scope.

Scopes

¿Cómo puedo modificar una variable externa dentro de una función?

Variable global: Puede ser leída y modificada desde cualquier scope.

Regla 3: Se pueden definir variables *globales* con el tag *global*.

Scopes

Regla 3: Se pueden definir variables *globales* con el tag *global*.

```
def funcion(num):
  global ret
  ret += num**2
  print(ret)

ret = 5
funcion(4)
print(ret)
```

¿Cuál es la salida del programa?

Scopes

Regla 3: Se pueden definir variables *globales* con el tag *global*.

```
def funcion(num):
  global ret
  ret += num**2
  print(ret)

ret = 5
funcion(4)
print(ret)
```

¿Cuál es la salida del programa? 21, 21

Scopes

Regla 3: Se pueden definir variables *globales* con el tag *global*.

```
1 def funcion(num):
2   global ret
3   ret = num**2
4   print(ret)
5
6 funcion(4)
7 print(ret)
```

¿Cuál es la salida del programa?

Scopes

Regla 3: Se pueden definir variables *globales* con el tag *global*.

```
def funcion(num):
  global ret
  ret = num**2
  print(ret)

funcion(4)
print(ret)
```

¿Cuál es la salida del programa? 16, 16

Scopes

Regla 3: Se pueden definir variables *globales* con el tag *global*.

```python
def funcion_1(num):
  global ret
  ret = num**2
  print(ret)

def funcion_2(num):
  b = ret + num
  print(b)

ret = 5
funcion_1(4)
funcion_2(4)
```

¿Cuál es la salida del programa?

Scopes

Regla 3: Se pueden definir variables *globales* con el tag *global*.

```
def funcion_1(num):
  global ret
  ret = num**2
  print(ret)

def funcion_2(num):
  b = ret + num
  print(b)

ret = 5
funcion_1(4)
funcion_2(4)
```

¿Cuál es la salida del programa? 16, 20

Scopes: Resumen

Variables pueden ser *locales* o *globales*.

- **Local**: Puede ser leída y modificada dentro de su scope.
- **Global**: Puede ser leída y modificada desde cualquier scope.

Scopes: Resumen

Variables pueden ser *locales* o *globales*.

- **Local**: Puede ser leída y modificada dentro de su scope.
- **Global**: Puede ser leída y modificada desde cualquier scope.

Reglas:

1. Toda variable definida en una función es *local*.
2. Desde una función se puede leer, pero no modificar, el valor de una variable definida en su *scope superior*.
3. Se pueden definir variables *globales* con el tag *global*.

Scopes: Resumen

Variables pueden ser *locales* o *globales*.

- **Local**: Puede ser leída y modificada dentro de su scope.
- **Global**: Puede ser leída y modificada desde cualquier scope.

Reglas:

1. Toda variable definida en una función es *local*.
2. Desde una función se puede leer, pero no modificar, el valor de una variable definida en su *scope superior*.
3. Se pueden definir variables *globales* con el tag *global*.

Importante: Nunca usen variables globales.

Import

Import: Permite usar código definido en otros archivos (módulos o librerías).

Import

Import: Permite usar código definido en otros archivos (módulos o librerías).

Sintaxis 1

import nombre_archivo

Sintaxis 2

from nombre_archivo **import** nombre_método_a_importar

Import

Ejemplo

Import

Ejemplo

```
import sudoku

# Cargamos el tablero 1
sudoku.cargarTablero(1)

# Mostramos lo que hay en la casilla (0,0) -> 0
print(sudoku.obtener(0,0))
# Mostramos lo que hay en la casilla (2,0) -> 8
print(sudoku.obtener(2,0))
```

Import

¿Por qué existen los imports?

Import

¿Por qué existen los imports?

Ventajas:

- Defino variables y métodos una vez, y los llamo desde distintos programas.

Import

¿Por qué existen los imports?

Ventajas:

- Defino variables y métodos una vez, y los llamo desde distintos programas.
- Separamos en archivos componentes lógicas.

Import

¿Por qué existen los imports?

Ventajas:

- Defino variables y métodos una vez, y los llamo desde distintos programas.
- Separamos en archivos componentes lógicas.
- Permiten compartir componentes.

Import

¿Qué cosas puedo importar?

Import

¿Qué cosas puedo importar?

- Funciones.
- Variables (constantes).

Import

Ejemplo: Creemos una librería que permita hacer operaciones matemáticas básicas.

Import

Ejemplo: Creemos una librería que permita hacer operaciones matemáticas básicas.

Pasos:

- Crear archivo python (calculadora.py)
- Definir funciones para sumar, restar, multiplicar y dividir.
- Definir la constante Pi.

Import: Ejemplo

calculadora.py

```
1  # Operaciones aritméticas
2  def sumar(n1,n2):
3    return n1+n2
4  def restar(n1,n2):
5    return n1-n2
6  def dividir(n1,n2):
7    return n1/n2
8  def multiplicar(n1,n2):
9    return n1*n2
10
11 # Pi aproximado
12 pi = 3.1415
```

Import: Ejemplo

Para usar la librería:

- Crear nuevo archivo (user.py)
- Poner calculadora.py en la misma carpeta que user.py
- Importar calculadora
- Llamar a métodos y constantes de calculadora.py

Import: Ejemplo

user.py

```
1  from calculadora import sumar, multiplicar
2  from calculadora import restar, dividir, pi
3
4  # 3*4-5
5  print(restar(multiplicar(3,4),5))  # >>> 7
6
7  # 3*2+4/5-2/3
8  n1 = multiplicar(2,3)
9  n2 = dividir(4,5)
10 n3 = dividir(2,3)
11 print(restar(sumar(n1,n2),n3))     # >>> 6.13
12
13 # pi
14 print(pi)                          # >>> 3.1415
```

Import: Módulos de python

Python posee muchas librerías que pueden importar.

Import: Módulos de python

Python posee muchas librerías que pueden importar.

Entre ellas:

- **math** → funciones y constantes matemáticas.
- **random** → funciones para producir números random.
- Otros → **link**

Import: math

Módulo.Paquete	Retorno
math.exp(x)	e elevado a x
math.log(x)	log natural de x
math.pi	Constante π
math.e	Constante e
math.sin(x)	Seno de x
math.cos(x)	Coseno de x
math.tan(x)	Tangente de x

Más funciones math **aquí**

Import: math

```
from math import exp,cos,pi,log

print(exp(5.5))
print(cos(pi/2))
print(log(exp(4)))
```

```
import math

print(math.exp(5.5))
print(math.cos(math.pi/2))
print(math.log(math.exp(4)))
```

Import: random

Random:

Import: random

Random:

`random.random()`: Retorna un float entre 0 y 1.
`random.randint(i,j)`: Retorna un int en [i,j].
`random.randrange(i=0,j,k=1)`: Retorna un int en [i,j-1] c/k.

Import: random

Random:

`random.random()`: Retorna un float entre 0 y 1.
`random.randint(i,j)`: Retorna un int en [i,j].
`random.randrange(i=0,j,k=1)`: Retorna un int en [i,j-1] c/k.

Más funciones random **aquí**

Import: random

```
from random import random, randint, randrange

print(random())             # float entre 0 y 1
print(randint(1,2))         # random entre [1,2]
print(randrange(4))         # random entre [0,1,2,3]
print(randrange(1,4))       # random entre [1,2,3]
print(randrange(1,4,2))     # random entre [1,3]
```

```
import random

print(random.random())
print(random.randint(1,2))
print(random.randrange(4))
print(random.randrange(1,4))
print(random.randrange(1,4,2))
```

Import: random

Ejemplo: Programe un jugador de piedra, papel o tijera.

Import: random

Ejemplo: Programe un jugador de piedra, papel o tijera.

```
from random import randint

v = 0; l = 0
while(True):
  # [1-piedra 2-papel 3-tijera]
  u = int(input("Ingrese jugada: "))
  pc = randint(1,3)
  print("jugada pc:",pc)
  if((pc+1)%3==u):
    v += 1
  if((u+1)%3==pc):
    l += 1
  print("v",v,"-",l,"l")
```

Ejercicios

1) ¿Cuál es la salida de los siguientes códigos?

```
a = 3; i = 0
while(i < a):
  global b
  b = i*a
  i+=1
print(i)
print(a)
print(b)
```

```
def sumar2():
  global suma
  suma += 2

suma = 0
sumar2()
sumar2()
print(suma)
```

2) ¿Qué pasa si quitamos *suma = 0*?

3) *Enchule* el piedra, papel o tijera (su despliegue en consola).

Ejercicios

4) Programa un jugador de: rock, paper, scissors, lizard, spock.