Universidad de Valparaíso
CHILE

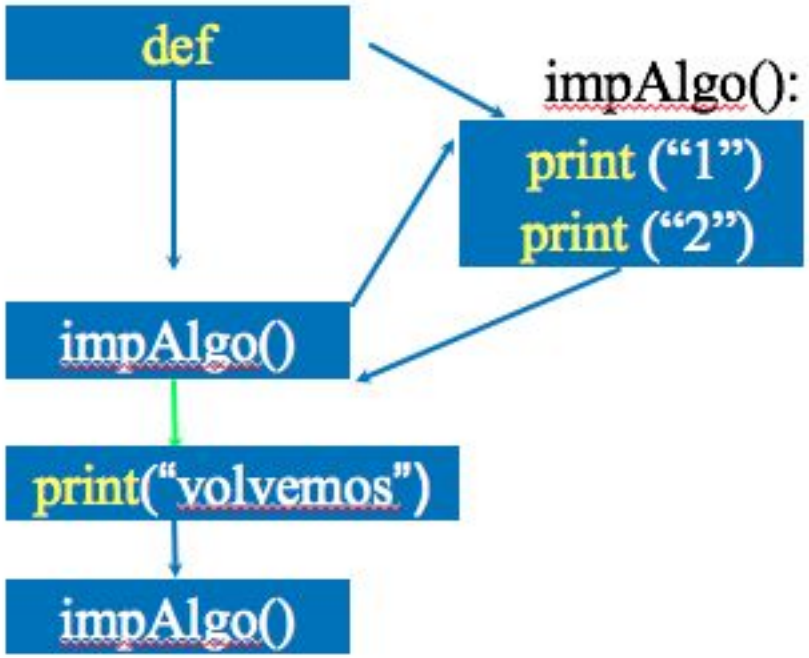
Fundamentos de Programación

Prof. Dr. Roberto Muñoz S. (Sec 1, Lab 502)
Prof. Dr. Rodrigo Olivares O. (Sec 2, Lab 401)
Prof. Víctor Ríos (Sec 3, Lab 303)
Prof. Pablo Olivares (Sec 4, Lab 301)

Escuela de Ingeniería en Informática
Facultad de Ingeniería
Universidad de Valparaíso

Funciones

Funciones




```
1  def impAlgo():
2      print ("1")
3      print ("2")
4
5  impAlgo()
6  print ("volvemos")
7  impAlgo()
```

Los grupos de código reutilizables se denominan **funciones**

- Existen **dos tipos** de funciones en Python
  - **Funciones internas**, que vienen incluidas en Python: **print**(), **input**(), **type**(), **float**(), **int**() ...
  - Funciones que las **define el programador** y luego **utiliza**.

- Tratamos los nombres de funciones internas (*built-in*) como "**nuevas**" palabras reservadas (*es decir, evitamos usarlas como nombres de variables*).

Definición de Funciones

- Una **función** en Python es un código reutilizable que toma **argumentos** como entrada, hace algún **cómputo** y **retorna** uno o más resultados.
- Se define una **función** usando la palabra reservada **def.**
- Llamamos o **invocamos** a la **función** escribiendo su nombre, paréntesis y **argumentos** en una expresión.

¿CÓMO?

- Creamos una nueva **función** mediante la palabra clave **def**, seguida por parámetros opcionales dentro de paréntesis
- **Indentamos** el cuerpo de la función
- Así se **define** la función, pero **no** se ejecuta el cuerpo de la función

```
def impParImpar(valor):
    if (valor % 2 == 0):
        print("Es par")
    else:
        print("Es impar")
```

Una vez que hemos **definido** una función, podemos **llamarla** (o **invocarla**) tantas veces queramos.

Seguimos, de este modo, el modelo de **almacenar** y **reutilizar.**

```
def impParImpar(valor):
    if (valor % 2 == 0):
        print("Es par")
    else:
        print("Es impar")

print("Ingrese un número:")
n = int(input())
while n != 0:
    impParImpar(n)
    print("Ingrese un número:")
    n = int(input())
print("Se acabó el programa!")
```

- Un **argumento** es un valor que pasamos (o enviamos) a la función a modo de entrada cuando invocamos la función
- Usamos **argumentos** para que la función haga cosas diversas cuando la llamamos desde distintos puntos o momentos
- Colocamos los argumentos en paréntesis después del nombre de la función

```
print ("volvemos")          parImpar(num)
```

Un **parámetro** es una variable que usamos en la definición de la función, a modo de "variable referencial", y que luego volvemos a utilizar en el cuerpo de la función para acceder a argumentos o valores a los cuales se aplicará la función en una invocación concreta

```
def parImpar(valor):
```

```
    def test(valor1, valor2):
```

Una función **tomará los argumentos** (si existen), ejecutará las instrucciones y **devolverá** (**retornará**) un valor, que se utilizará como resultado o valor de la función invocada en la expresión de llamada.

Se utiliza la palabra clave **return** para indicar el valor que se quiere retornar.

Cuando una función **no retorna** un valor, se denomina una función "**void**”

```python
def impParImpar(valor):
    if (valor % 2 == 0):
        return 1
    else:
        return 0

print("Ingrese un número:")
n = int(input())
while n != 0:
    resultado = impParImpar(n)
    if (resultado == 1):
        print("Es par")
    else:
        print("Es impar")
    print("Ingrese un número:")
    n = int(input())
print("Se acabó el programa!")
```

```
def test(valor):
    if (valor %2 == 0):
        return ("Es par")
    else:
        return ("Es impar")

while(True):
    num = int (input("Ingrese un número, presione 0 para salir: "))
    if num == 0:
        break
    else:
        print (test(num))
print("Se acabó el programa")
```

Ejercicio

La agencia de seguridad nacional se ha contactado con Ud., para mejorar su sistema de comunicación segura entre sus aliados. Para ello, la agencia le ha solicitado que desarrolle un programa en Python que **encripte** una palabra (cadena), para luego enviarla a sus aliados. La palabra debe ser **ingresada desde teclado**, debe tener un **largo igual o inferior a 30** caracteres, **no debe contener espacios en blancos**, y el resultado de la encriptación debe ser devuelto donde fue invocado.

La encriptación se debe realizar utilizando la función **encript(palabra)** que **recibe** una cadena de caracteres (palabra) y **retorna** una nueva cadena (palabra) encriptada. Además incluya la función **buscaBlancos(palabra)**, que retorna **True** si es que existe al menos un espacio en blanco en la palabra.

Ejercicio: Use la siguiente “plantilla”

```
def encript(palabra):
    # Encripta bajo la regla definida
    return palabraEscriptada

def existenBlancos(palabra):
    # Valida que la palabra no contenga espacios en blancos
    # Regresa True o False
    return respuesta

respuesta = True
while respuesta != False:
    palabra = input("Ingrese una palabra a encriptar: ")
    respuesta = buscarBlancos(palabra)
    if respuesta == True:
        print("La palabra tiene espacios en blanco. Debes volver a ingresarla.")

print("La palabra encriptada es:", encript(palabra))
```

Listas en Python

Una lista es un tipo de colección

- Una colección permite poner varios valores en una "**variable**".
- Una colección es algo útil porque podemos poner varios valores agrupados convenientemente.

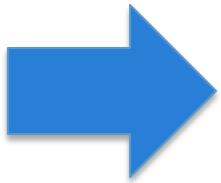
```
amigos = ["Pedro", 'Juan', "Diego" ]
juegos = ["LOL", "DOTA", 'FIFA', 'Candy Crush']
```

Constantes de una lista

- Las constantes de una lista se encierran entre corchetes o paréntesis, y se separan por comas.
- Cualquier objeto de Python puede constituir un elemento de una lista (**hasta una lista puede ser un elemento de otra lista**).
- Una lista puede estar **vacía**.

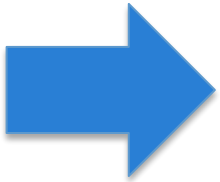
```
lista = [-51,4,0,24,32]
print(lista)

lista = ["red","blue","green"]
print(lista)

lista = ["yellow",-21,3.75]
print(lista)

lista = ["black",[2,-5],"gray",4]
print(lista)

lista = []
print(lista)
```

```
for i in [5,4,3,2,1]:
    print(i)
print("Salió del for!")
```



```
5
4
3
2
1
Salió del for!"
```

```
amigos = ["Pedro", "Juan", "Diego"]
for amigo in amigos:
    print("Hola", amigo)
print("Salió del for!")
```



```
Hola:  Pedro
Hola:  Juan
Hola:  Diego
Salió del for!
```

```python
amigos = ["Pedro", 'Juan', "Diego" ]
juegos = ["LOL", "DOTA", 'FIFA', 'Candy Crush']

for nombre in amigos:
    for j in juegos:
        print(nombre, "juega", j)
```

```
Pedro juega LOL
Pedro juega DOTA
Pedro juega FIFA
Pedro juega Candy Crush
Juan juega LOL
Juan juega DOTA
Juan juega FIFA
Juan juega Candy Crush
Diego juega LOL
Diego juega DOTA
Diego juega FIFA
Diego juega Candy Crush
>
```

Búsqueda en listas

- Al igual que con las cadenas, podemos acceder a cualquier elemento de una lista a partir del nombre de ésta y el índice del elemento entre corchetes.

```
amigos= ['Pedro', 'Juan', 'Diego' ]
print (amigos[1])
```

```
Juan
```

Largo de una lista

- La función **len()** toma una lista como parámetro y retorna el número de elementos de la lista.

- De hecho, **len()** nos indica el número de elementos de cualquier conjunto o secuencia (**por ejemplo de una cadena de caracteres**).

```
test = 'Hola Mundo'
print (len(test))

x = [ 1, 2.0, 'juan', 'a']
print (len(x))
```

Función Range

- La función range devuelve una lista de **guarismos**, desde cero hasta un número menos que el parámetro de la propia función.

- Podemos crear un bucle de índices usando for y un iterador (entero).

```
1   amigos= ['Pedro', 'Juan', 'Diego' ]
2
3   for amigo in amigos :
4       print ('Hola:',  amigo)
5
6
7   for i in range(len(amigos)) :
8       amigo = amigos[i]
9       print ('Hola:',  amigo)
10
```

```
Hola: Pedro
Hola: Juan
Hola: Diego
Hola: Pedro
Hola: Juan
Hola: Diego
```

Importante con listas

- Podemos crear una lista vacía y agregarle posteriormente elementos usando el método **append**.

- La lista establece un orden y se agregan los nuevos elementos al **final de la lista**.

Buscar algo en una lista

- Python proporciona dos operadores que permiten comprobar si un elemento está en una lista.

- Dichos operadores son de tipo lógico y devuelven **True** or **False**

Importante: No modifican la lista.

Buscar algo en una lista

```
lista = [-51,4,0,24,32]

if 4 in lista:
    print("Está!")

if -4 not in lista:
    print("No está!")
```

Una lista es una secuencia ordenada

- Las listas pueden contener muchos elementos, que mantienen el orden original hasta que se modifica éste.
- Se pueden reordenar las listas.
- El método **sort** (a diferencia de las cadenas) significa "**ordénate tú misma**".

```
amigos= ['Pedro', 'Juan', 'Diego' ]

for amigo in amigos :
    print ('Hola:',  amigo)
amigos.sort()
print(amigos)
```

Ejercicio

Universidad de Valparaíso
CHILE

Estadísticas

El Instituto de Estadísticas le ha solicitado al curso de Fundamentos de Programación que elabore un programa para calcular algunas métricas del curso. Dicha estadística requiere la edad y el nombre de un estudiante, y muestre la edad promedio, y la edad y nombre del(de la) mayor.

Archivos

Características

- Datos relacionados entre sí
- Almacenamiento persistente
- Nombre
- Ubicación
- Memoria secundaria

- **Abrir el archivo**
- **Leer el archivo**
- **Escribir**
- **Cerrar el archivo**

```
archivo = open(nombreArchivo, modo)
```

Modos

r	Lectura
r+	Lectura/Escritura
w	Sobreescritura. Si no existe archivo se creará
a	Añadir. Escribe al final del archivo
b	Binario
+	Permite lectura/escritura simultánea
U	Salto de línea universal: win cr+lf, linux lf y mac cr
rb	Lectura binaria
wb	Sobreescritura binaria
r+b	Lectura/Escritura binaria

Modos

r	Sólo Lectura. Debe existir el archivo.
r+	Lectura y Sobreescritura. Debe existir el archivo.
w	Sólo Sobreescritura. Si no existe archivo, se creará.
a	Sólo Añade. Escribe al final del archivo. Si no existe archivo, se creará.

Existen más modos de apertura...

```
archivo = open("InformaticaUV.txt", "r")
```

```
archivo = open("/Users/UV/InformaticaUV.txt", "w")
```

```
archivo = open(path, "a")
```

Abrir

Método **read**() permite leer un número de bytes.

Si no se indica número se leerá todo lo que reste.

```
1  archivo = open("carreras.txt",'r')  # abre archivo en modo lectura
2  cad1 = archivo.read(5)  # lee los 5 primeros bytes
3  cad2 = archivo.read()  # lee la información restaste
4  print(cad1)  # muestra la primera lectura
5  print(cad2)  # muestra la segunda lectura
6  archivo.close  # cierra archivo
```

readline() permite leer una línea completa

```
1 archivo = open("carreras.txt","r")   # abre archivo en modo lectura
2 while True:  # inicia bucle infinito para...
3  linea = archivo.readline()  # … leer línea a línea
4  if not linea: break  # … hasta que no haya más que leer
5  print(linea)  # muestra la línea leída
6 archivo.close()  # cierra archivo
```

readline() permite leer una línea completa

```
1 archivo = open("carreras.txt","r")
2 linea = archivo.readline()
3 while linea:
4     print(linea)
5     linea = archivo.readline()
6 archivo.close()
```

readlines() permite leer todas las líneas del archivo y almacenarlas en un arreglo.

conte = archivo.readlines()

```
archivo = open("ejemplo.txt", "r")
for linea in archivo.readlines():
    print(linea)
```

Procesar un archivo

```
archivo = open("arch.txt","r")
contenido = archivo.readlines()
print(contenido)

for linea in contenido:
    linea = linea.strip("\n")
    print(linea)
    palabras = linea.split(' ')
    for pal in palabras:
        print(pal)

archivo.close()
```

Leer

write(cadena) Escribe cadena dentro del archivo.

```
archivo = open("ejemplo.txt", "w")
archivo.write('Nueva linea')
archivo.close()
```

```
archivo = open("ejemplo.txt", "a")
archivo.write('Linea agregada')
archivo.close()
```

Universidad de Valparaíso
CHILE

Fundamentos de Programación

Prof. Dr. Roberto Muñoz S. (Sec 1, Lab 502)
Prof. Dr. Rodrigo Olivares O. (Sec 2, Lab 401)
Prof. Víctor Ríos (Sec 3, Lab 303)
Prof. Pablo Olivares (Sec 4, Lab 301)

Escuela de Ingeniería en Informática
Facultad de Ingeniería
Universidad de Valparaíso