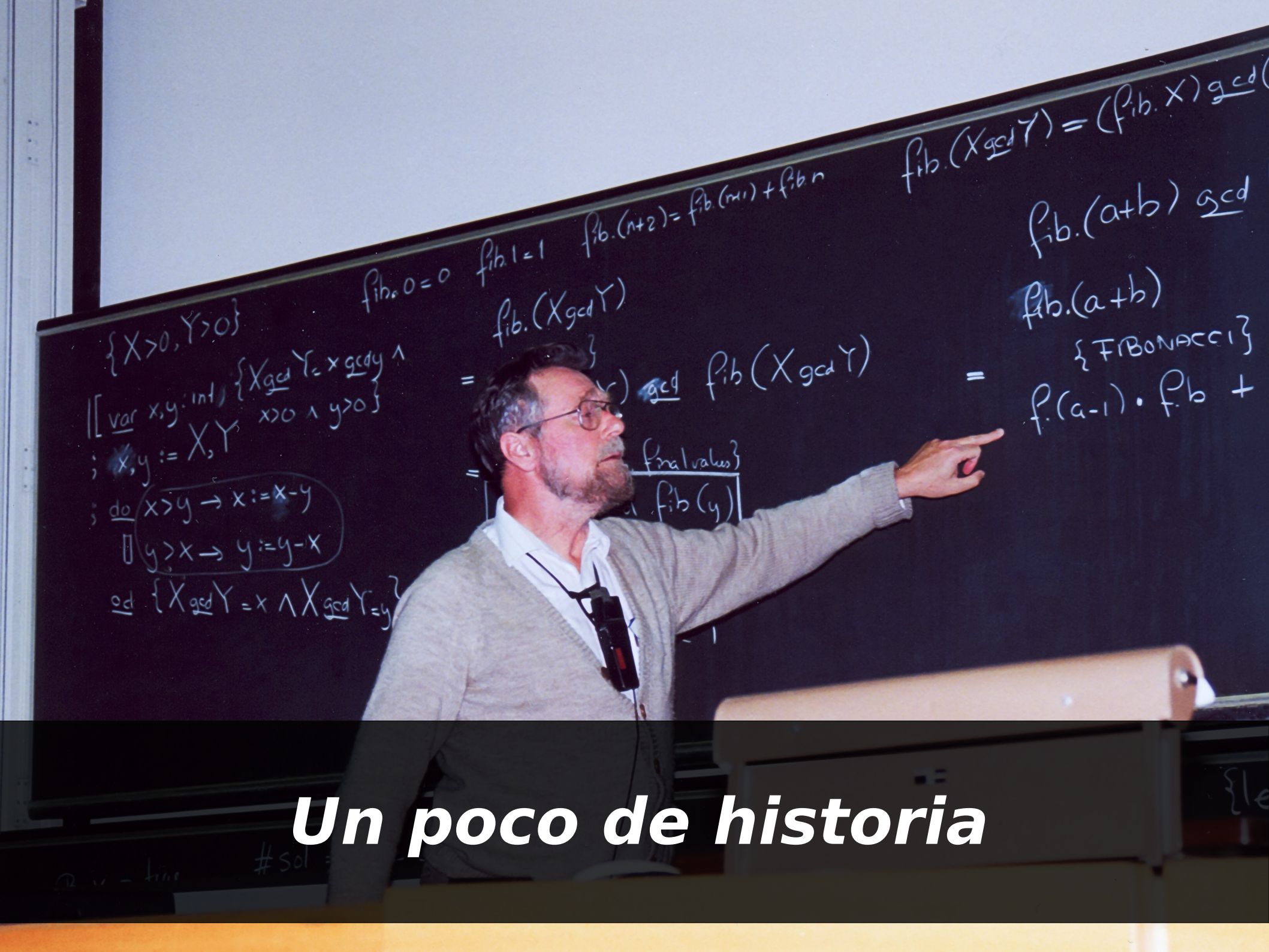
Programación I
Programación modular

Pablo Garaizar Sagarminaga
Borja Sanz Urquijo

Facultad de Ingeniería

Índice

- Un poco de historia.
- Subprogramas.



Un poco de historia

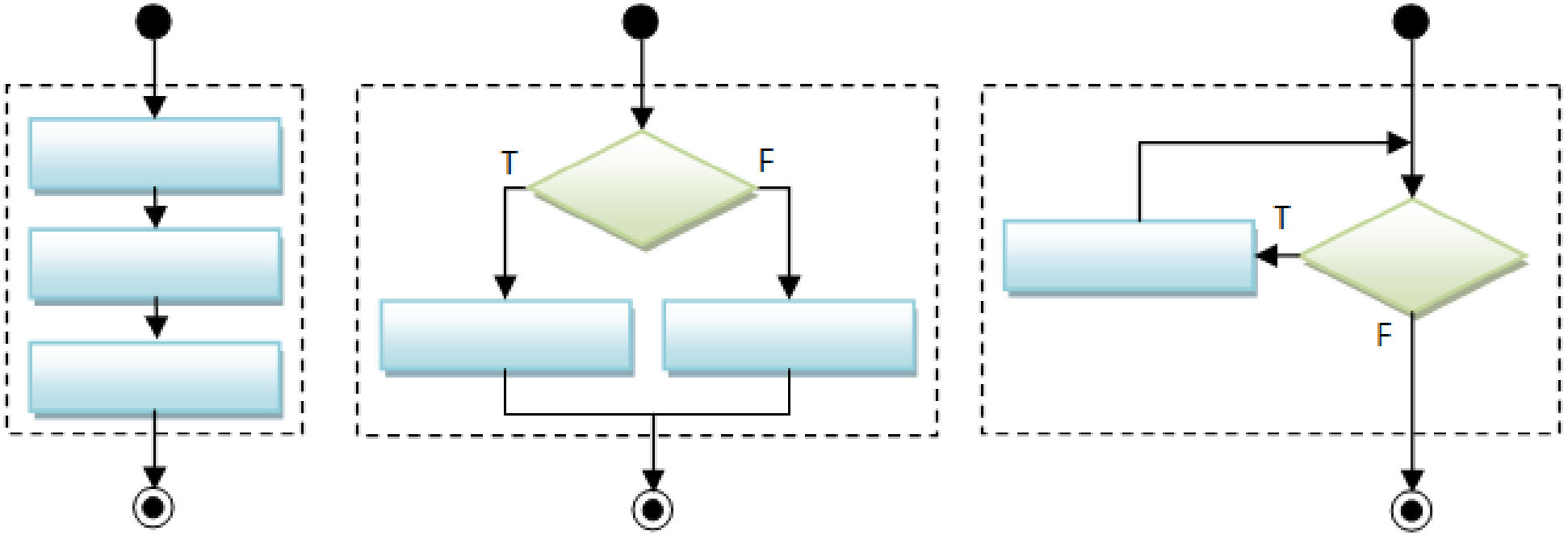
Historia

- Años 60 del s. XX:
  - Programación en **código máquina** (binario).
  - **Ensamblador** (mnemónicos traducidos a binario).
  - Lenguajes con **saltos incondicionales** (GOTO).
    - Código ***“spaghetti”*** → Difícil de mantener.

Historia

- 1968, Edgser Dijkstra: *“Go To Statement Considered Harmful”*.
  - **Programación estructurada** → Todo código tiene que ser:
    - **Secuencia**: ejecución de una instrucción tras otra.
    - **Selección**: ejecución de una secuencia u otra, según el valor de una variable booleana (if).
    - **Iteración**: ejecución de una secuencia mientras una variable booleana sea 'verdadera' (for, while, etc.).

Programación estructurada



Programación modular

- Evolución de la programación estructurada para resolver problemas **más grandes**.
  - ***"Divide y vencerás"***
    - Problema complejo → subproblemas más simples → dividir otra vez hasta que sea fácil resolver cada parte → **módulos**.
  - Evitar **copiar** código → *smells!*
  - Fomentar la **reutilización** de código.

Módulos

- Un módulo tiene **una** tarea definida.
- Puede que necesite de **otros** para llevarla a cabo.
- También llamados:
  - Subrutinas.
  - Subprogramas.
  - Procedimientos.
  - **Funciones**.
  - **Métodos** → nombre preferido en POO.
  - ,,,

Funciones

Funciones

- Elementos importantes:
  - **Ámbito**: ¿quién puede llamar a este módulo?
  - Tipo de dato de **retorno**: ¿qué devuelve?
  - **Argumentos**: ¿qué datos necesita?
  - **Código**: ¿cómo obtiene el resultado a partir de los argumentos?

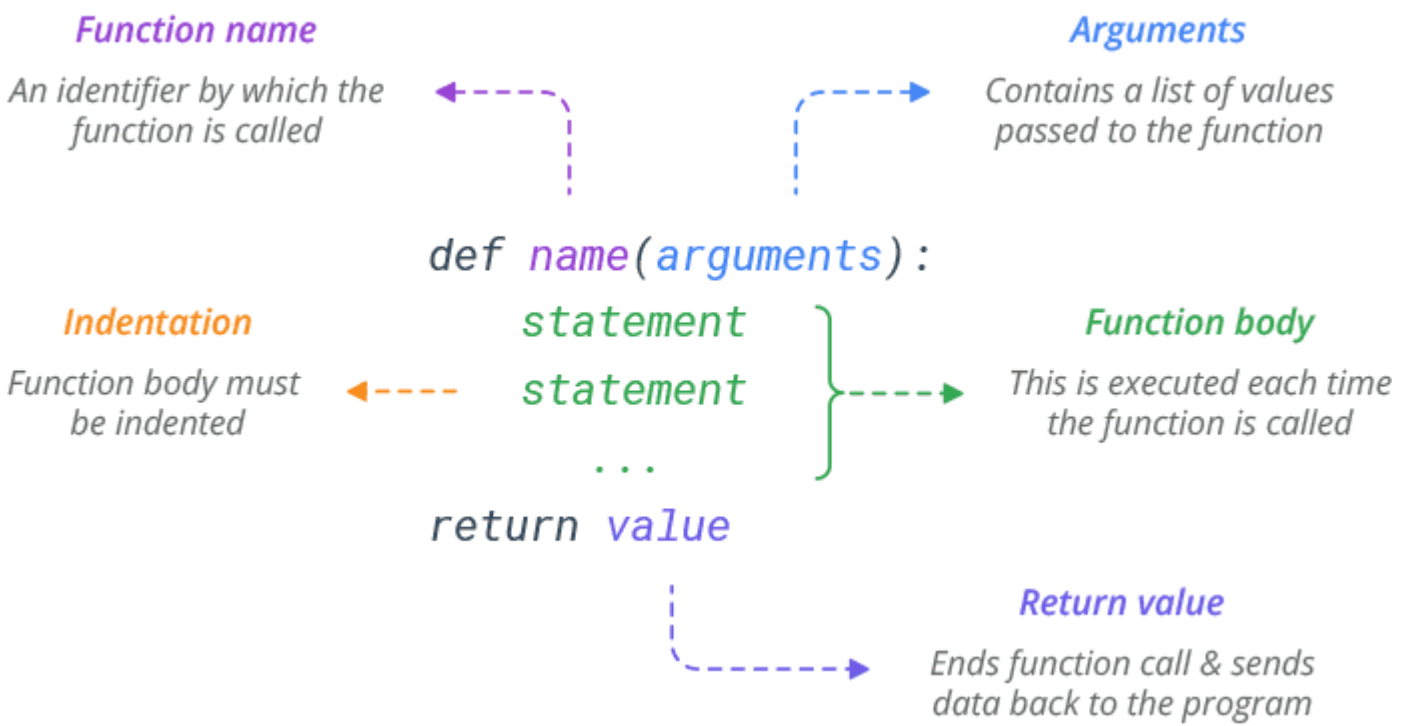
Funciones

- Ejemplo:

```
def promedio (a, b):
    return (a + b) / 2
```

  - **Ámbito**: público
  - **Tipo de datos de retorno**: float
  - **Argumentos**: a (int), b (int)

Funciones


Function name
An identifier by which the function is called
Arguments
Contains a list of values passed to the function
def name(arguments):
statement
statement
...
return value
Indentation
Function body must be indented
Function body
This is executed each time the function is called
Return value
Ends function call & sends data back to the program

Funciones: retorno

- Una función siempre devolverá **una** cosa:
  - Si la función no devuelve nada: no tiene **return** → devuelve **None**
  - Si devuelve **una cosa** (ej: un entero): **return entero**.
  - Si necesitamos que devuelva **más de un dato:**
    - Puede devolver **una lista** u otra estructura de memoria que permita guardar varias cosas (ejemplo: **tupla**).

Tuplas

- Una tupla es algo parecido a una **lista inmutable**:
  - Agrupa **varios** valores de manera **ordenada**.
  - Ejemplo:

```
posicion = (1.5, 3.1, 0.6)
```

Funciones: retorno

- Ejemplo de función que devuelve **varios** valores en **una** tupla:

```
def firstLast (text):
    return text[0], text[-1]

>>> print(firstLast('Hola mundo cruel'))
('H', 'l')
```

Funciones: argumentos

- Una función puede tener de 0 a N argumentos.
- Argumentos por defecto:

```
def saludar(nombre, saludo=”Hola”):
    print(f’¡{saludo} {nombre}!’)
```

- Argumentos por palabras clave (keywords):

```
def fecha(dia, mes, anyo):
    print(f’{dia}/{mes}/{anyo}’)

>>> fecha(mes=11, dia=22, anyo=2021)
```

Funciones: argumentos

- Si queremos que tenga un número **indefinido** de argumentos:

```
def sumatorio(*nums):
    suma = 0
    for num in nums:
        suma = suma + num
    return suma

# nums se comporta como una tupla
```

Funciones: ámbito

- Las variables definidas dentro de una función solo pueden accederse en ese ámbito:

```
def sumatorio(*nums):
    suma = 0
    for num in nums:
        suma = suma + num
    return suma

>>> print(suma)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'suma' is not defined
```

Funciones built-in

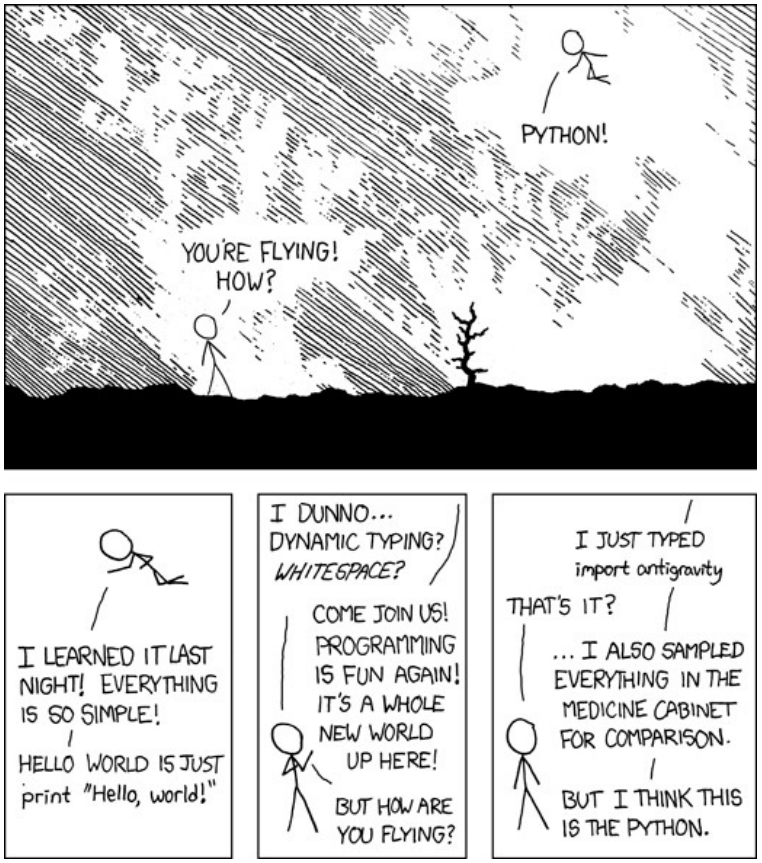
- En Python hay muchas funciones que no tenemos que programar, están ya programadas (built-in):

		Built-in Functions		
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	

Módulos

- Además de las funciones built-in, hay programadas muchas otras funciones dentro de **módulos** que podemos **importar**:

```
from nombreModulo import funcion
```



Llamadas encadenadas

- **Dentro** de una función, se puede llamar a **otras** funciones:
  - Ejemplo:

```
def multiplicar (a, b):
    resultado = 0
    while (b > 0):
        resultado = sumar(resultado, a)
        b--
```

Recursividad

- También llamada recursión o recurrencia.
- Un método puede llamarse **a sí mismo**:
  - Si se hace **mal** → desbordamiento de pila (***stack overflow***).
  - Si se hace **bien** → **recursividad**.
    - Difícil, pero elegante :)

Recursividad

- Ejemplo: factorial **iterativo**

```
def factorial(n) :
    resultado = 1
    for i in range(1, n+1):
        resultado *= i
    return resultado
```

Recursividad

- Ejemplo: factorial **recursivo**

```
def factorial(n):
  if (n==0):
    return 1
  else:
    return n * factorial(n-1)
```

Referencias

- John Sturtz, Real Python.
- John M. Zelle. Python Programming: An Introduction to Computer Science.
- Learnbyexample.org.
- Wikipedia.

Referencias

- Imágenes:
  - Wikipedia
  - Chua Hock-Chuan, Yet another insignificant… programming notes.