

A blue circuit board pattern is visible along the left edge of the slide.

FACULTAD DE
INGENIERÍA DEL
EJERCITO

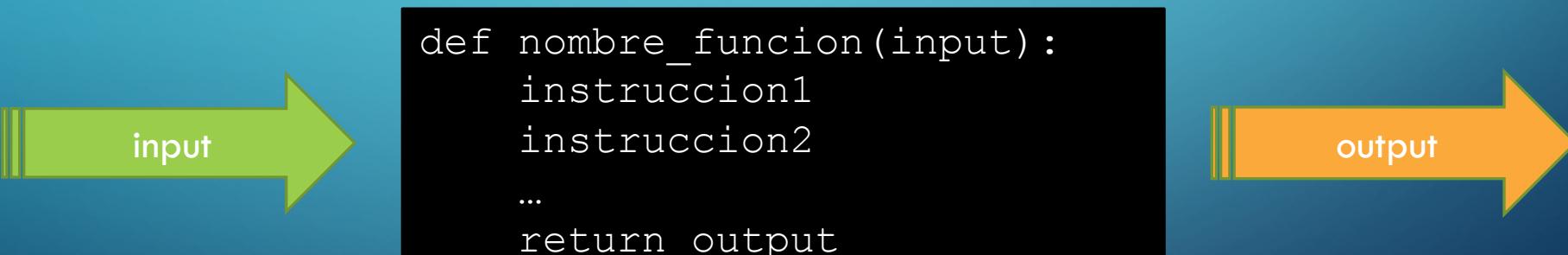
UNIVERSIDAD DE LA
DEFENSA NACIONAL

PARADIGMAS DE PROGRAMACIÓN

V

Funciones

Unidad de programación con orden de secuencia y alcance propio	Diseñadas para realizar <u>una</u> tarea al momento de ser invocada	Reciben parámetros y retornan resultados. Tienen una firma (signature)	Pueden ser recursivas (se invocan a si misma) o anónimas (sin nombre)	Permiten la reutilización de código. Generalmente, verbos
----------------------------------------------------------------	---------------------------------------------------------------------	------------------------------------------------------------------------	-----------------------------------------------------------------------	-----------------------------------------------------------



En Python, siempre retorna un resultado

FUNCIONES - SINTAXIS

Palabra reservada “def” y luego el nombre de la función

Parámetros formales que recibe

Instrucciones a ejecutar.

Retorna el flujo mediante la instrucción return (opcional)

IMPORTANTE: Documentación completa de la función. El que no lo hace TIENE PUNTOS NEGATIVOS EN LA NOTA

PARÁMETROS Y PUNTEROS

```
inicio = "Hola"  
def imprimir_juntos(str1, str2):  
    print(str1 + str2)  
  
...  
imprimir_juntos(inicio, "que tal")
```

str: "Hola "

str: "que tal"

function:
imprimir_juntos

NoneType: None

PARÁMETROS Y PUNTEROS

```
inicio = "Hola"  
def imprimir_juntos(str1, str2):  
    print(str1 + str2)  
  
...  
imprimir_juntos(inicio, "que tal")
```

Global:
inicio
imprimir_juntos

str: "Hola "

str: "que tal"

function:
imprimir_juntos

NoneType: None

PARÁMETROS Y PUNTEROS

```
inicio = "Hola"  
def imprimir_juntos(str1, str2):  
    print(str1 + str2)  
  
...  
imprimir_juntos(inicio, "que tal")
```

Global:
inicio
imprimir_juntos

imprimir_juntos:
str1 = inicio
str2
returnValue

str: "Hola "

str: "que tal"

function:
imprimir_juntos

NoneType: None

PARÁMETROS Y PUNTEROS

```
cantidad = 3
saludos = ["que", "tal"]

def cambiar_todo(x, greets):
    x = 56
    greets[1] = "change!"
    greets = "borre toda la lista!"

cambiar_todo(cantidad, saludos)
print(cantidad)
print(saludos)
```

CHALLENGE ACCEPTED



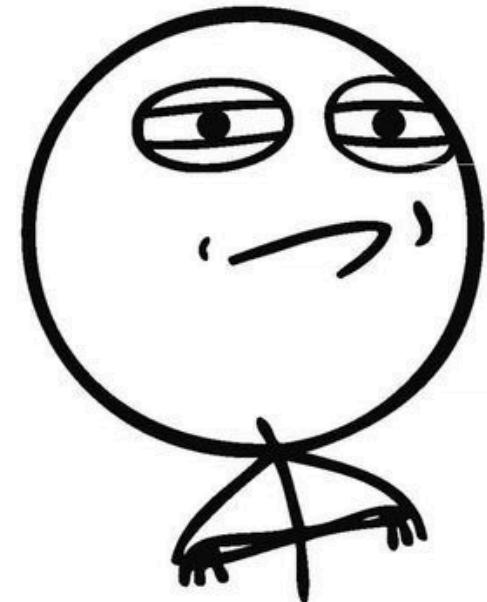
PARÁMETROS Y PUNTEROS

```
cantidad = 3
saludos = ["que", "tal"]

def cambiar_todo(x, greets):
    x = 56
    greets[1] = "change!"
    greets = "borre toda la lista!"

cambiar_todo(cantidad, saludos)
print(cantidad)
print(saludos)
```

CHALLENGE ACCEPTED



```
3
['que', 'change!']
```

PARÁMETROS Y PUNTEROS

```
cantidad = 3
saludos = ["que", "tal"]

def cambiar_todo(x, greets):
    x = 56
    greets[1] = "change!"
    greets = "borre toda la lista!"

cambiar_todo(cantidad, saludos)
print(cantidad)
print(saludos)
```

CHALLENGE ACCEPTED



3
['que', 'change!']

PARÁMETROS INVOCACIÓN



```
def posicion(a, b, c):  
    print(a, b, c)
```

```
>>> posicion("Uno", "Dos", "Tres")
```

```
Uno Dos Tres
```

```
>>> posicion("Tres", "Dos", "Uno")
```

```
Tres Dos Uno
```

```
>>> posicion(b="Pos B", c="Pos C", a="Pos A")
```

```
Pos A Pos B Pos C
```

```
# Parametros Default
```

```
def posicion(a, b="Pos 2", c="Pos 3"):  
    print(a, b, c)
```

```
>>> posicion(34)
```

```
34 Pos 2 Pos 3
```

```
>>> posicion(34, c="Pos C")
```

```
34 Pos 2 Pos C
```

```
>>> posicion(1, 2, 3)
```

```
1 2 3
```

PARÁMETROS INVOCACIÓN



```
def variable(*args):  
    print(args)  
  
>>> variable("a", "b", 3, 4)  
('a', 'b', 3, 4)  
>>> a = ("a", "b", 3, 4)  
>>> variable(*a)  
('a', 'b', 3, 4)  
  
# Con keywords  
def variable(**kargs):  
    print(kargs)  
  
>>> b = {'nombre': 'Nico', 'edad': 23}  
  
>>> variable(**b)  
{'nombre': 'Nico', 'edad': 23}  
  
>>> variable(nombre='Marias', edad=34)  
{'nombre': 'Marias', 'edad': 34}
```

Packing & Unpacking

FUNCIONES - OUTPUT

En Python, siempre retornan un valor

En caso de omisión de `return`, **devuelve None**, que es del tipo `NoneType`

Tambien pueden devolver más de un valor

Evitar efectos colaterales

```
def nada(*args):  
    pass  
  
>>> type(nada())  
<class 'NoneType'>  
  
def un_valor():  
    return 4 + 5  
  
def muchos_valores():  
    return (1, 2, 3, 4)  
  
>>> a, b, c, d, = muchos_valores()  
# a = 1, b = 2, ...
```

FUNCIONES – BUILT IN



Python built-in functions

`list`

`len`

`set`

`tuple`

`float`

`int`

`str`

`bool`

`type`

`print`

`input`

`bin`

`map`

`filter`

`...`

DOCUMENTACIÓN

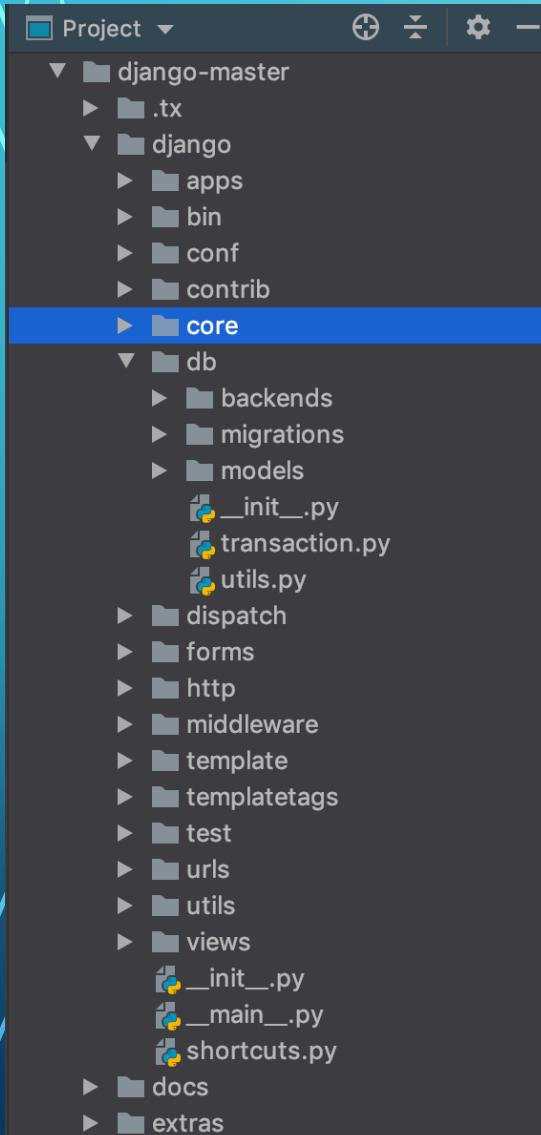
```
def complex(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments:
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)
    """
    if imag == 0.0 and real == 0.0:
        return complex_zero
    ...

```

Docstrings

MÓDULOS - PACKAGES



Utilizamos la forma mas tradicional (hay otras)

Cada modulo contiene un archivo `__init__` vacío

Dos comandos para importar módulos o funciones

```
import module_name [as  
alias_name]
```

- Importa el **namespace** para ser utilizado
- Opcional, se puede dar un alias

```
from module_name import c1,  
c2... [as c_name]
```

- Importa componentes del modulo indicado
- Opcional, se puede dar un alias

MODULES - INVOCACIÓN

```
>>> import datetime  
>>> datetime.datetime.now()  
datetime.datetime(2019, 8, 2, 16, 42, 5, 466515)  
  
>>> import datetime as dt  
>>> dt.datetime.now()  
datetime.datetime(2019, 8, 2, 16, 42, 23, 364792)  
  
>>> from datetime import datetime  
>>> datetime.now()  
datetime.datetime(2019, 8, 2, 16, 42, 45, 457061)  
  
>>> from datetime import datetime as dt  
>>> dt.now()  
datetime.datetime(2019, 8, 2, 16, 43, 3, 2035)
```

MODULES - INVOCACIÓN

```
>>> import datetime  
>>> datetime.datetime.now()  
datetime.datetime(2019, 8, 2, 16, 42, 5, 466515)
```

Namespace datetime

```
>>> import datetime as dt  
>>> dt.datetime.now()  
datetime.datetime(2019, 8, 2, 16, 42, 23, 364792)
```

Namespace datetime con alias dt

```
>>> from datetime import datetime  
>>> datetime.now()  
datetime.datetime(2019, 8, 2, 16, 42, 45, 457061)
```

Importa el tipo datetime del modulo datetime

```
>>> from datetime import datetime as dt  
>>> dt.now()  
datetime.datetime(2019, 8, 2, 16, 43, 3, 2035)
```

Importa el tipo datetime del modulo datetime con otro nombre, dt

Requisitos:

- Tener instalado Python 3.8+
- Terminal (cmd) o IDE

FUNCIONES DE STRINGS

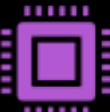
CONDICIONES INESPERADAS



Necesidad de software seguro



Paradigma DevOps



Ingeniería de Software: TDD



Testing

unitario
funcional
integral
otros

EXCEPCIONES EN PYTHON

Python implementa el mecanismo de Exceptions

Manejo de errores

Utilizadas para cuando se le solicita al programar realizar acciones que no puede.

- Abrir un archivo inexistente
- Dividir un numero por cero

Python muestra el *stack trace* de la excepción ocurrida para favorecer la corrección del problema o error.

- Nombre de archivo y la línea donde ocurrió

EJEMPLO: CALCULAR ALTURA

- Se define la función `calcular_altura` que recibe el área y la base de un rectángulo y debe calcular la altura.
- El programa utiliza `input` para la obtención de datos del usuario y luego la función para obtener el resultado

```
def calcular_altura(area, base):  
    return area / base  
  
while True:  
    area = int(input("Ingrese el área:"))  
    base = int(input("Ingrese la base:"))  
  
    print(calcular_altura(area, base))
```



¿Que cosas pueden fallar?

MECANISMO DE TRY/EXCEPT/ELSE

try:

[Codigo a ejecutar]

except:

[En caso de haber ocurrido la exception]

else:

[En caso de no haber ocurrido la exception]

finally:

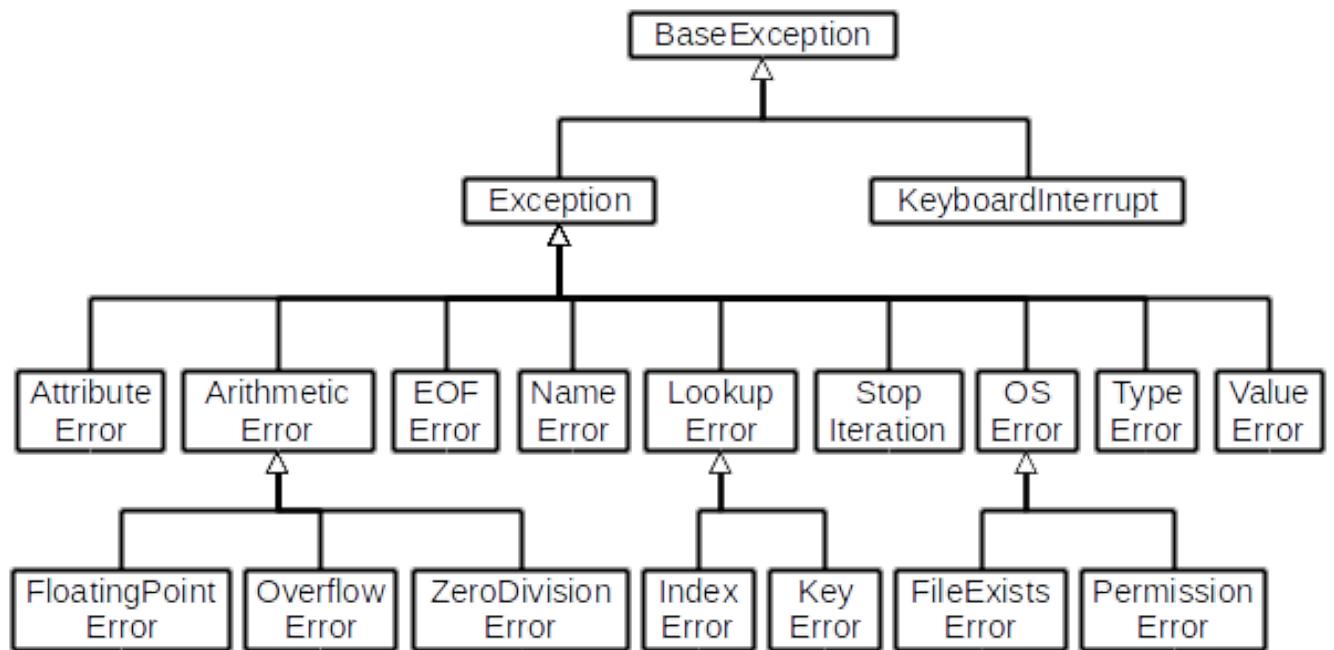
[Siempre se ejecuta]

DISPARAR EXCEPCIONES: RAISE

- Muy útiles cuando queremos realizar algún registro pero interrumpir el lanzamiento de la exception
- Se realiza simplemente utilizando `raise NombreExcepcion`

```
except (TypeError, AttributeError) as e:  
    log('Hit an exception')  
    raise e
```

EXCEPTIONS PYTHON 3



ASSERTS



En Python están pensadas para informar errores “irrecuperables” al desarrollador o situación que no se esperan en una ejecución normal.



Principalmente, difieren de las excepciones en informar estados o situación que no deberían ocurrir, y si ocurren, es por algún estado que no estuvo pensado previamente.

```
def calcular_altura(area, base):
    """Dada el area de un rectangulo y su base, calcula la altura"""
    resultado = area / base
    assert 0 <= resultado
    return resultado
```

ALGUNOS CUIDADOS CON ASSERT

- Se tiene que utilizar para estados o condiciones que no están previstas
- NO para validaciones de reglas de negocio o ingreso de datos
 - Variable global `__debug__` desactivada en ambiente productivos

```
def delete_product(prod_id, user):
    assert user.is_admin()
    assert store.has_product(prod_id)

    store.get_product(prod_id).delete()
```



Utilizarlos como elementos para DEBUG y no validación o manejo de errores