BASIC PYTHON

Tecnofor
by SNGULAR

CONTENTS

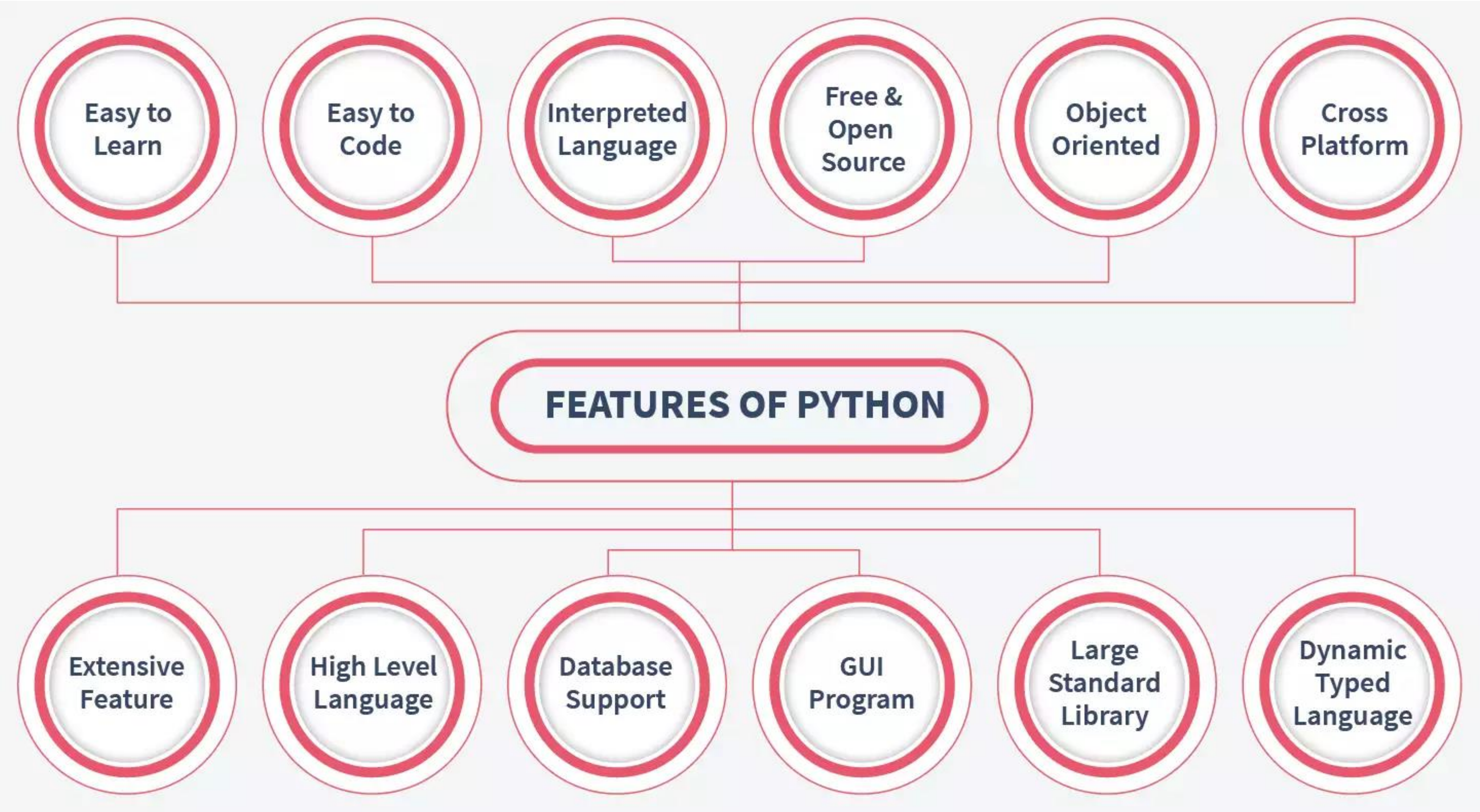
- Introduction to Python
  - What is Python?
  - Installing Python
  - Development environments (IDEs)
- Python Programming Fundamentals
  - Variables and data types
  - Operators
  - Control structures (if, else, loops)
- Functions and Modularity
  - Function definition and invocation
  - Parameters and arguments
  - Modularity and code reuse
- Data Structures in Python
  - Lists
  - Tuples
  - Dictionaries
  - Sets
- File Handling
  - Reading and writing text files
  - Using the with statement for file handling
- Introduction to Object-Oriented Programming (OOP)
  - Classes and objects
  - Attributes and methods
  - Encapsulation
- Exceptions
  - Error handling with try-except
  - The finally clause
  - Raising exceptions
- Introduction to Functional Programming in Python
  - Lambda functions
  - Map, filter, reduce
- Introduction to Modules and Packages
  - Creating and using modules
  - Importing modules and packages
- Introduction to Data Manipulation with Pandas
  - Pandas installation
  - Reading and writing data
  - Basic DataFrame operations

Tecnofor

Introduction to Python

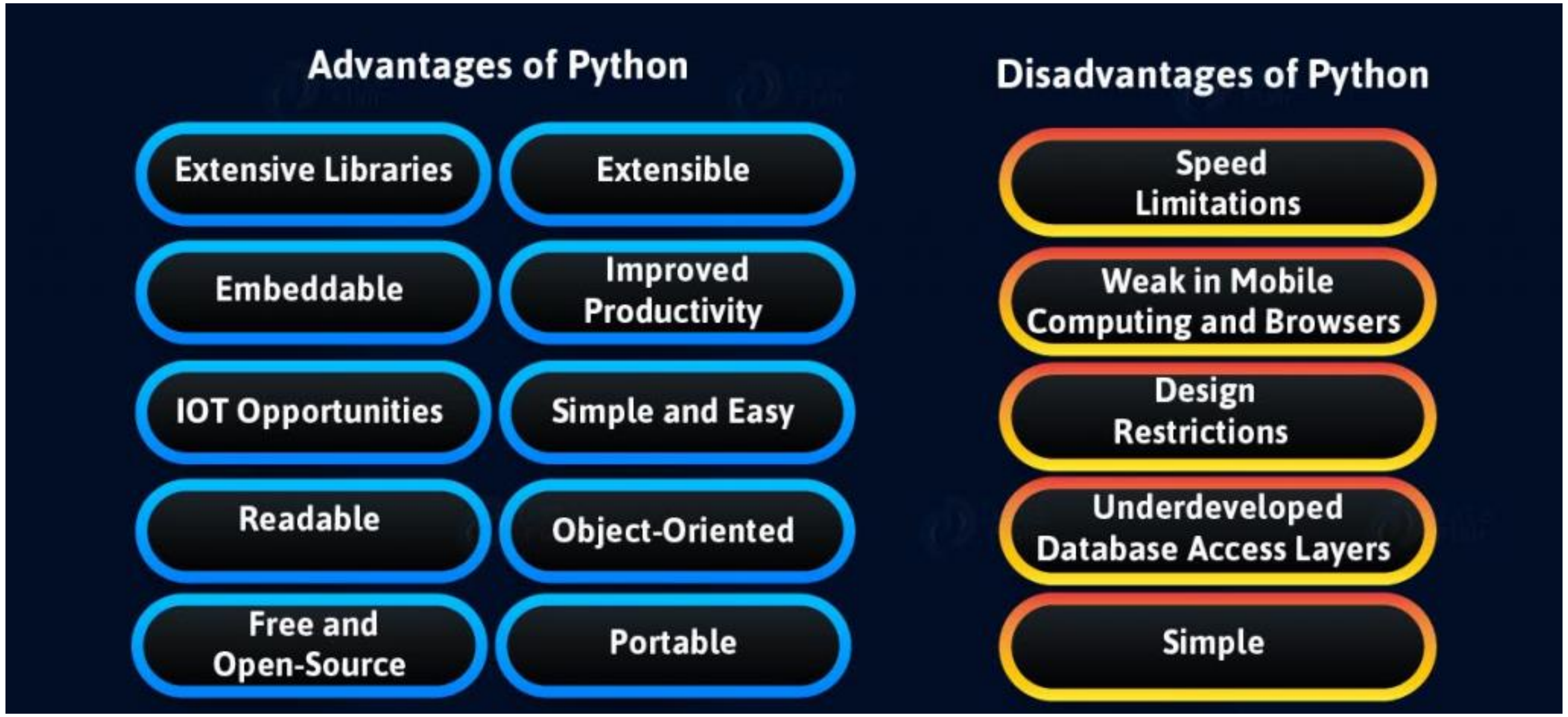
Introduction to Python

python™



FEATURES OF PYTHON

- Easy to Learn
- Easy to Code
- Interpreted Language
- Free & Open Source
- Object Oriented
- Cross Platform
- Extensive Feature
- High Level Language
- Database Support
- GUI Program
- Large Standard Library
- Dynamic Typed Language

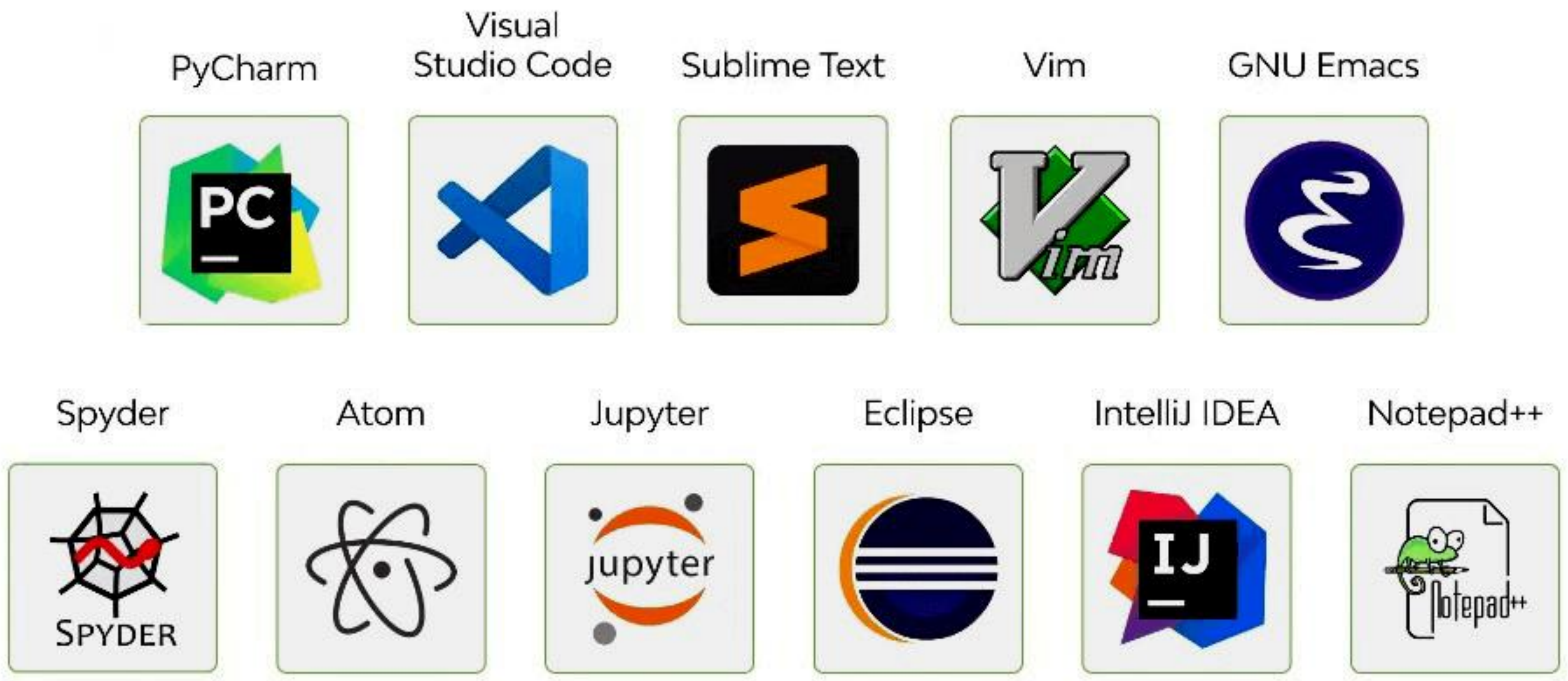


Advantages of Python

- Extensive Libraries
- Extensible
- Embeddable
- Improved Productivity
- IOT Opportunities
- Simple and Easy
- Readable
- Object-Oriented
- Free and Open-Source
- Portable

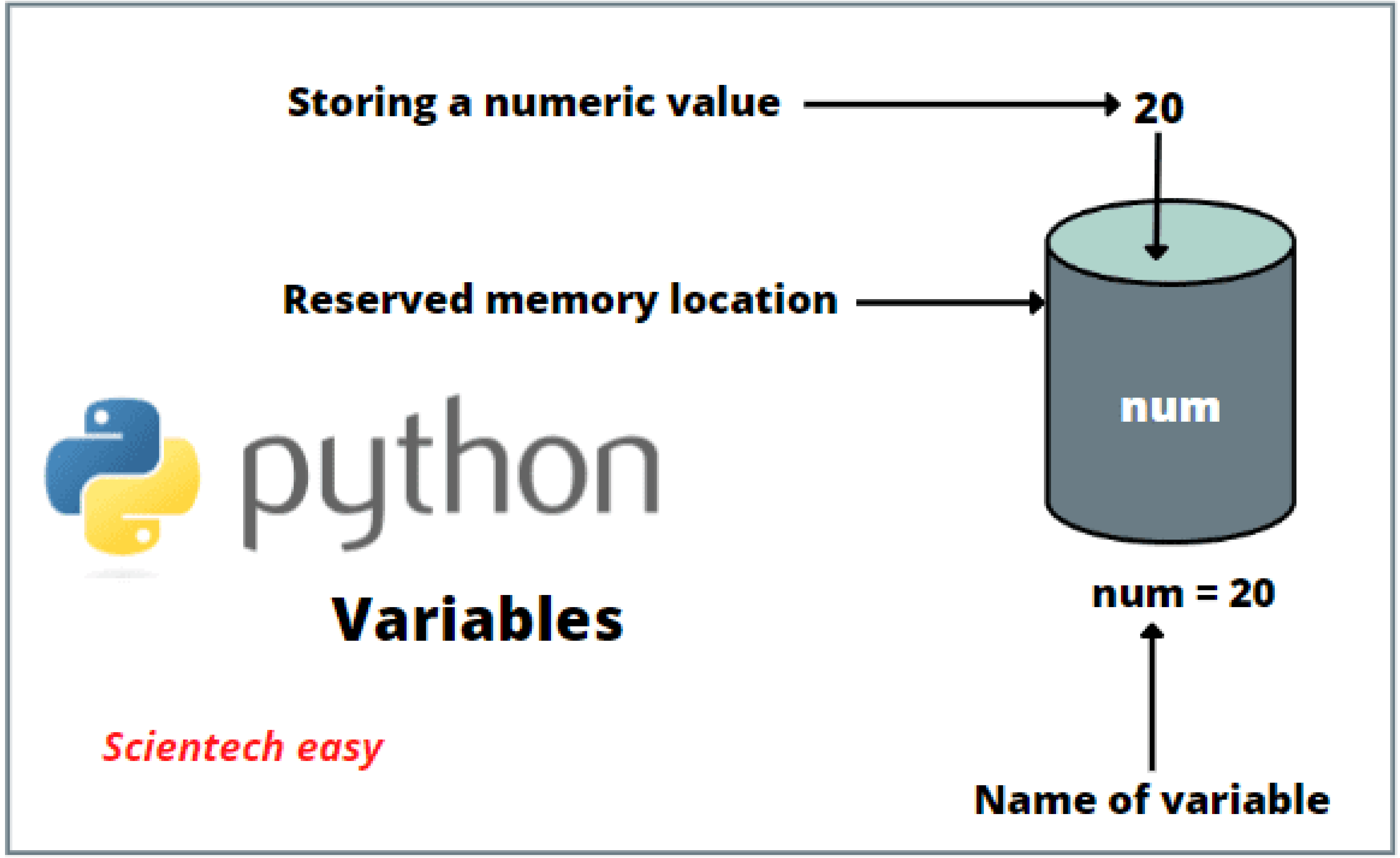
Disadvantages of Python

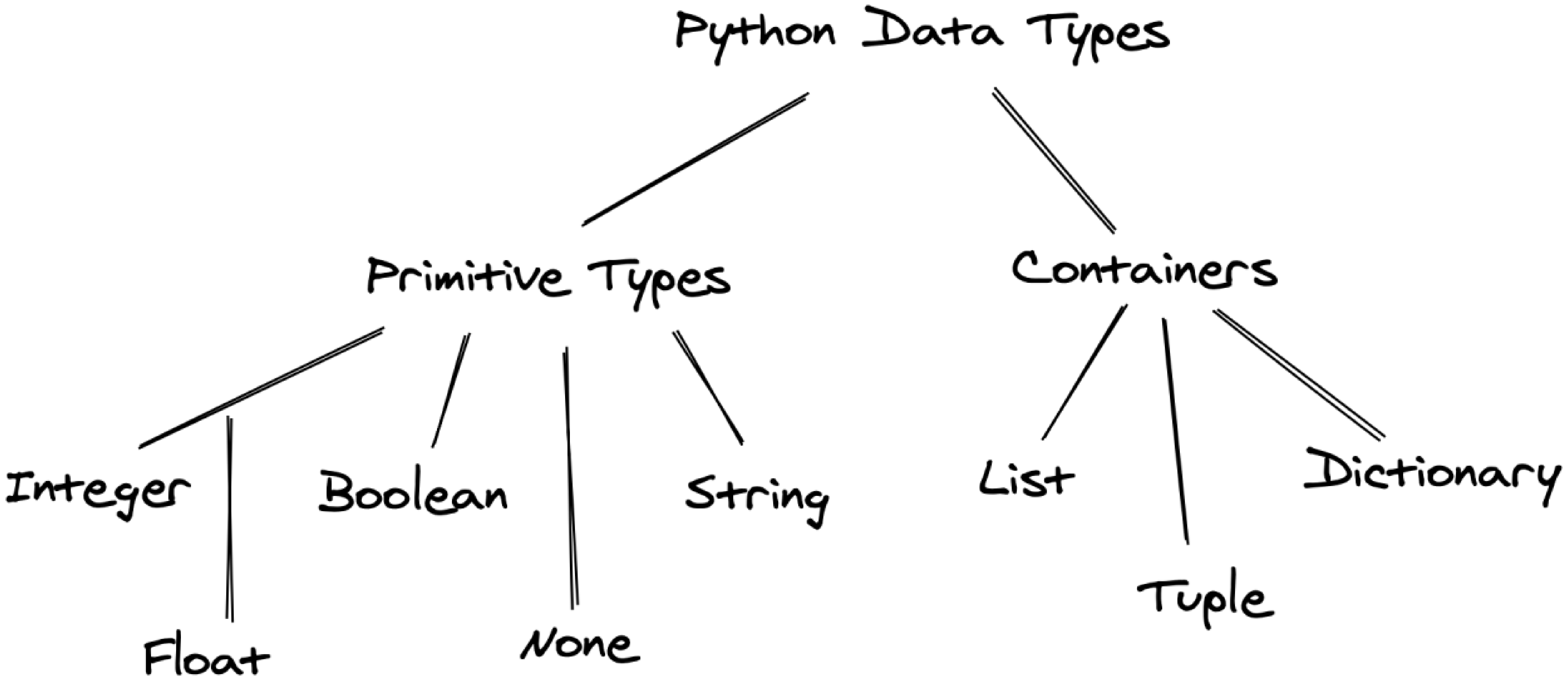
- Speed Limitations
- Weak in Mobile Computing and Browsers
- Design Restrictions
- Underdeveloped Database Access Layers
- Simple


PyCharm
Visual Studio Code
Sublime Text
Vim
GNU Emacs
Spyder
Atom
Jupyter
Eclipse
IntelliJ IDEA
Notepad++

Tecnofor

Python Programming Fundamentals


Storing a numeric value
20
Reserved memory location
num
python
Variables
num = 20
Scientech easy
Name of variable



Python Data Types

Primitive Types

Containers

Integer

Float

Boolean

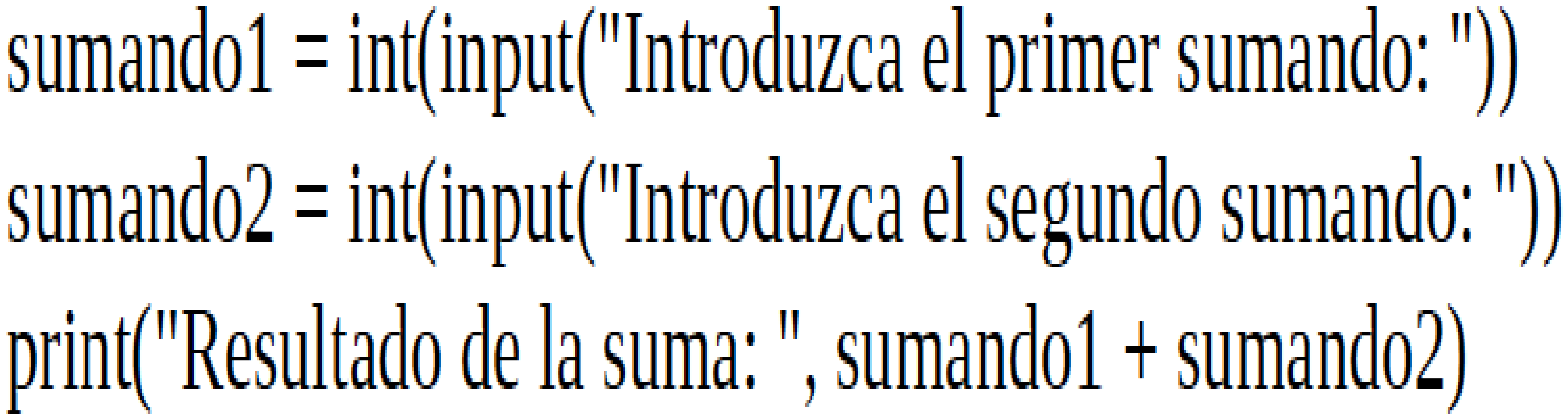
None

String

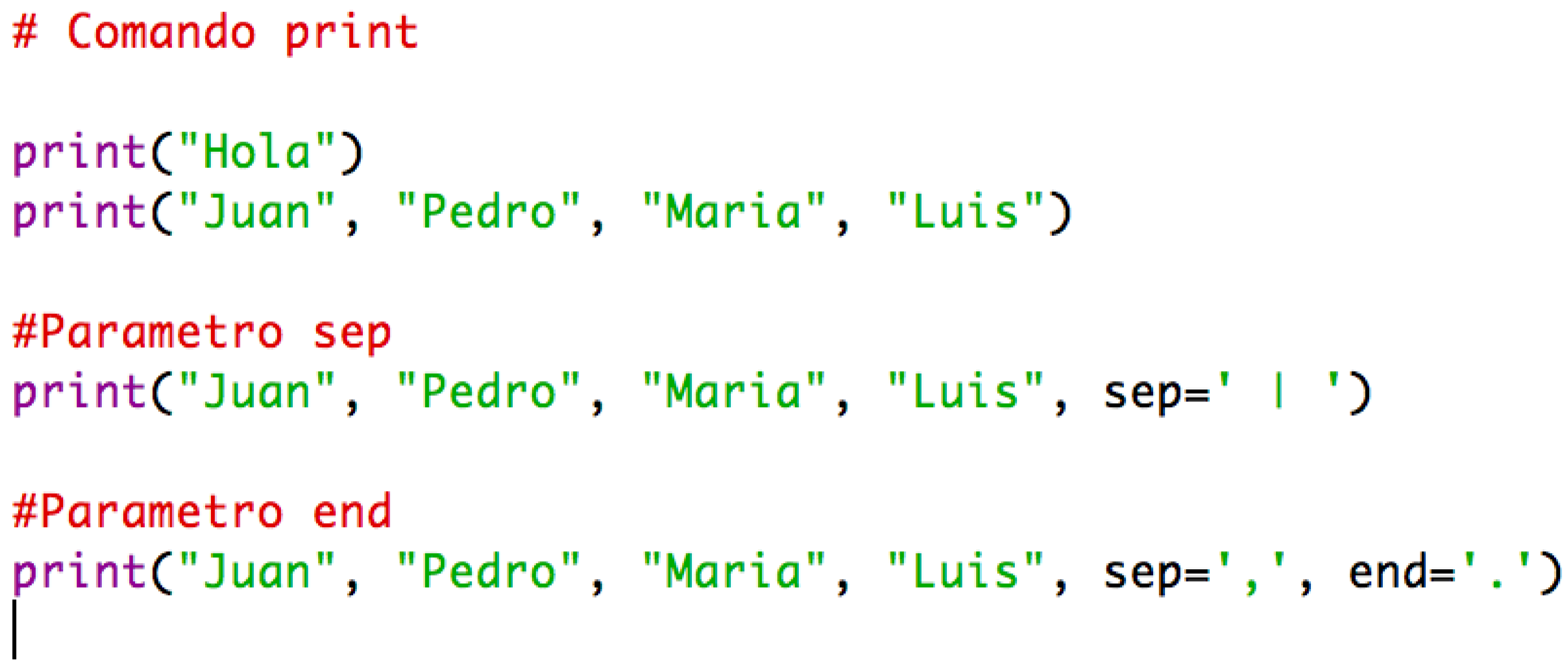
List

Tuple

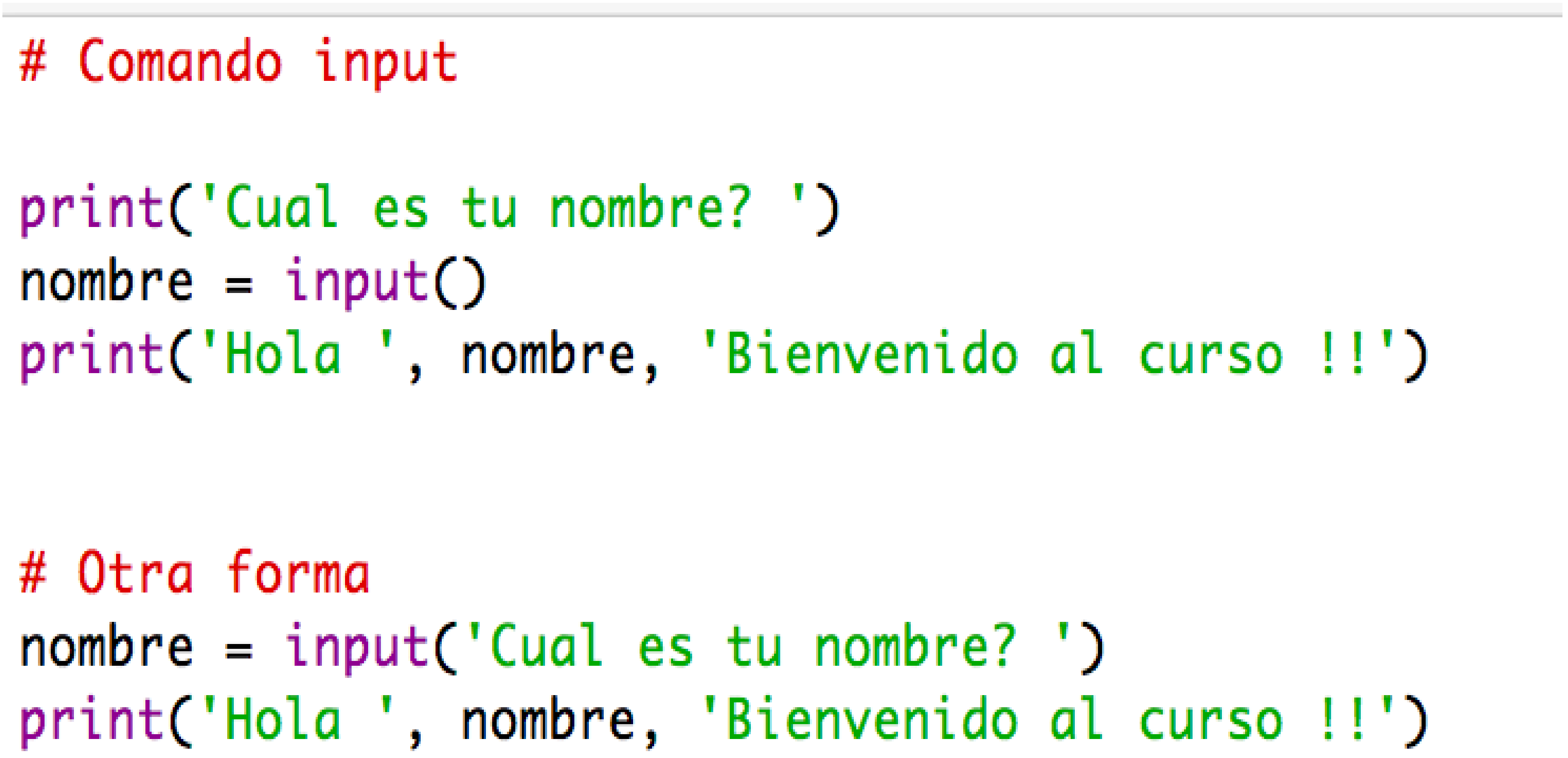
Dictionary



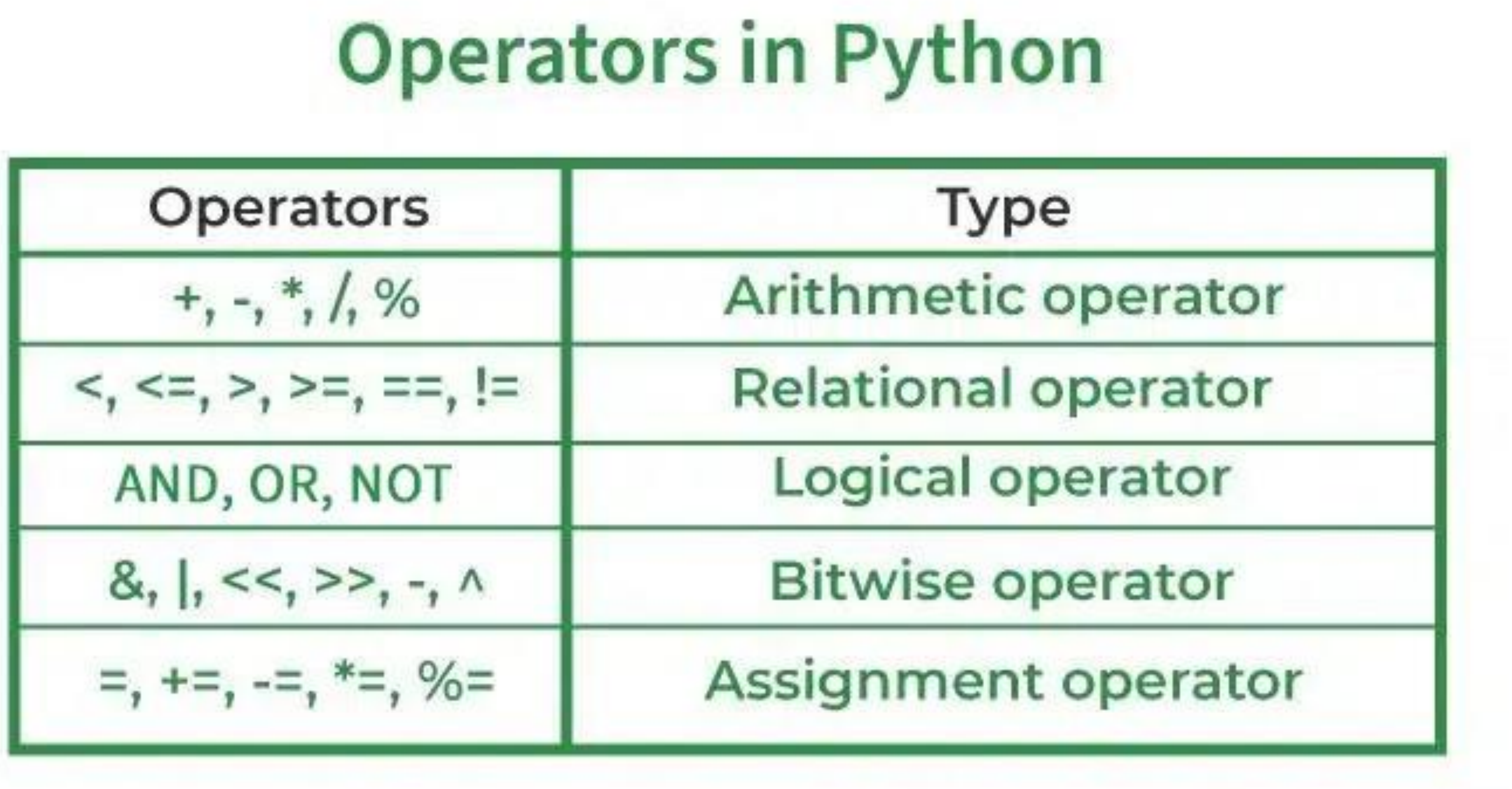
```
sumando1 = int(input("Introduzca el primer sumando: "))
sumando2 = int(input("Introduzca el segundo sumando: "))
print("Resultado de la suma: ", sumando1 + sumando2)
```



```
# Comando print

print("Hola")
print("Juan", "Pedro", "Maria", "Luis")

#Parametro sep
print("Juan", "Pedro", "Maria", "Luis", sep=' | ')

#Parametro end
print("Juan", "Pedro", "Maria", "Luis", sep=',', end='.')
```

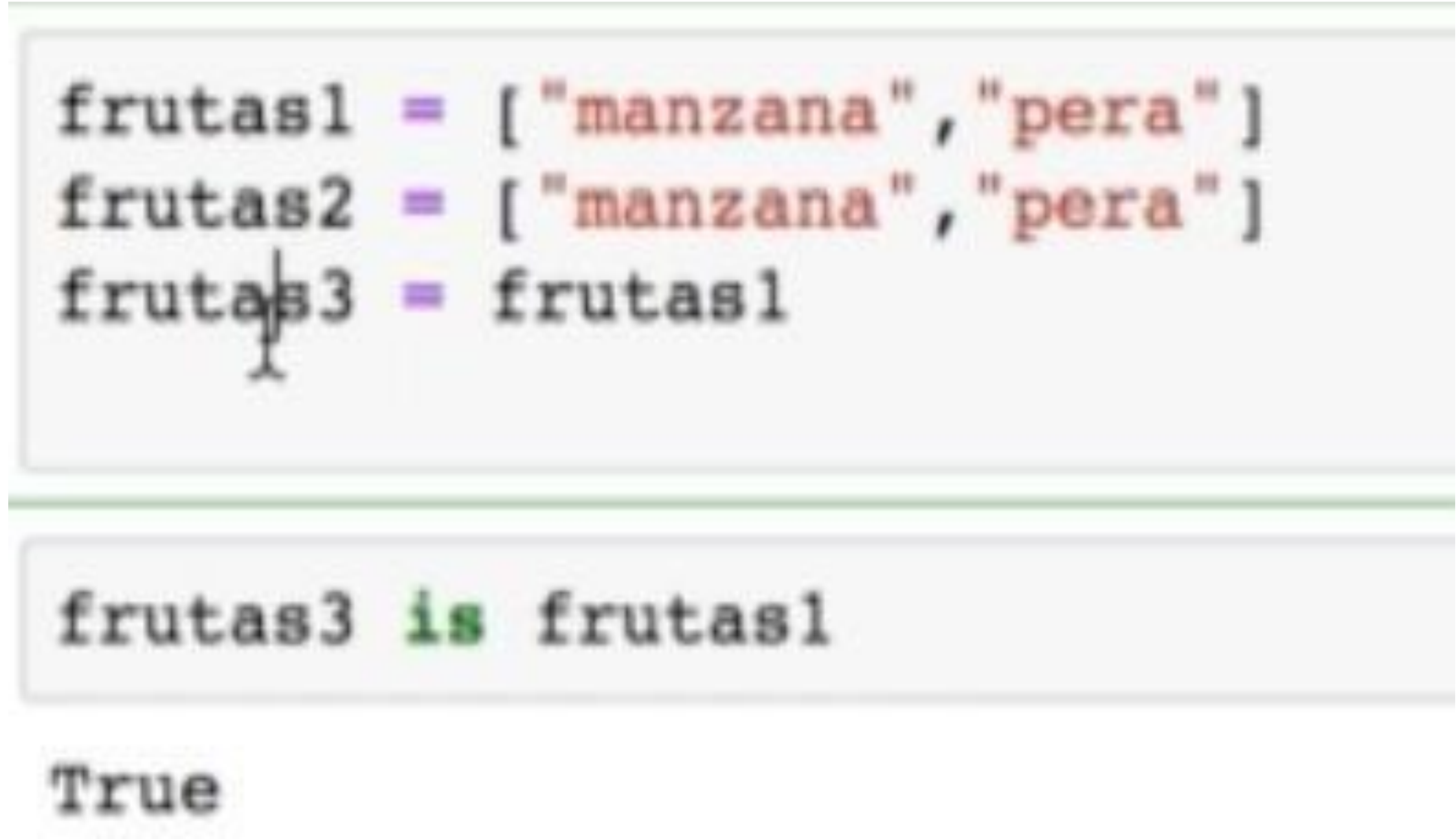


```
# Comando input

print('Cual es tu nombre? ')
nombre = input()
print('Hola ', nombre, 'Bienvenido al curso !!')


# Otra forma
nombre = input('Cual es tu nombre? ')
print('Hola ', nombre, 'Bienvenido al curso !!')
```



Operators in Python

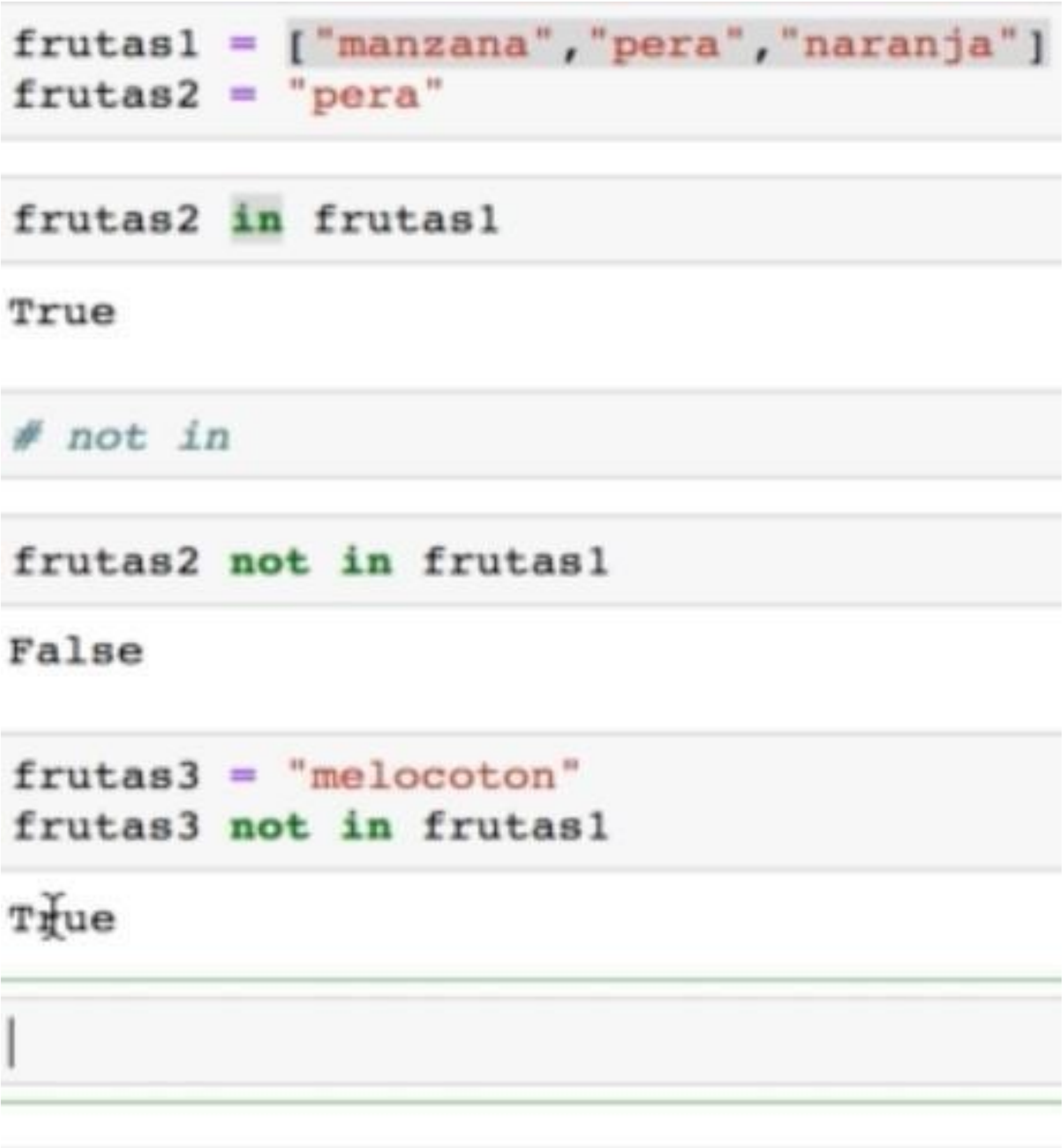
Operators	Type
+, -, *, /, %	Arithmetic operator
<, <=, >, >=, ==, !=	Relational operator
AND, OR, NOT	Logical operator
&, |, <<, >>, -, ^	Bitwise operator
=, +=, -=, *=, %=	Assignment operator



```
frutas1 = ["manzana","pera"]
frutas2 = ["manzana","pera"]
frutas3 = frutas1
```

```
frutas3 is frutas1
```

```
True
```



```
frutas1 = ["manzana","pera","naranja"]
frutas2 = "pera"
```

```
frutas2 in frutas1
```

```
True
```

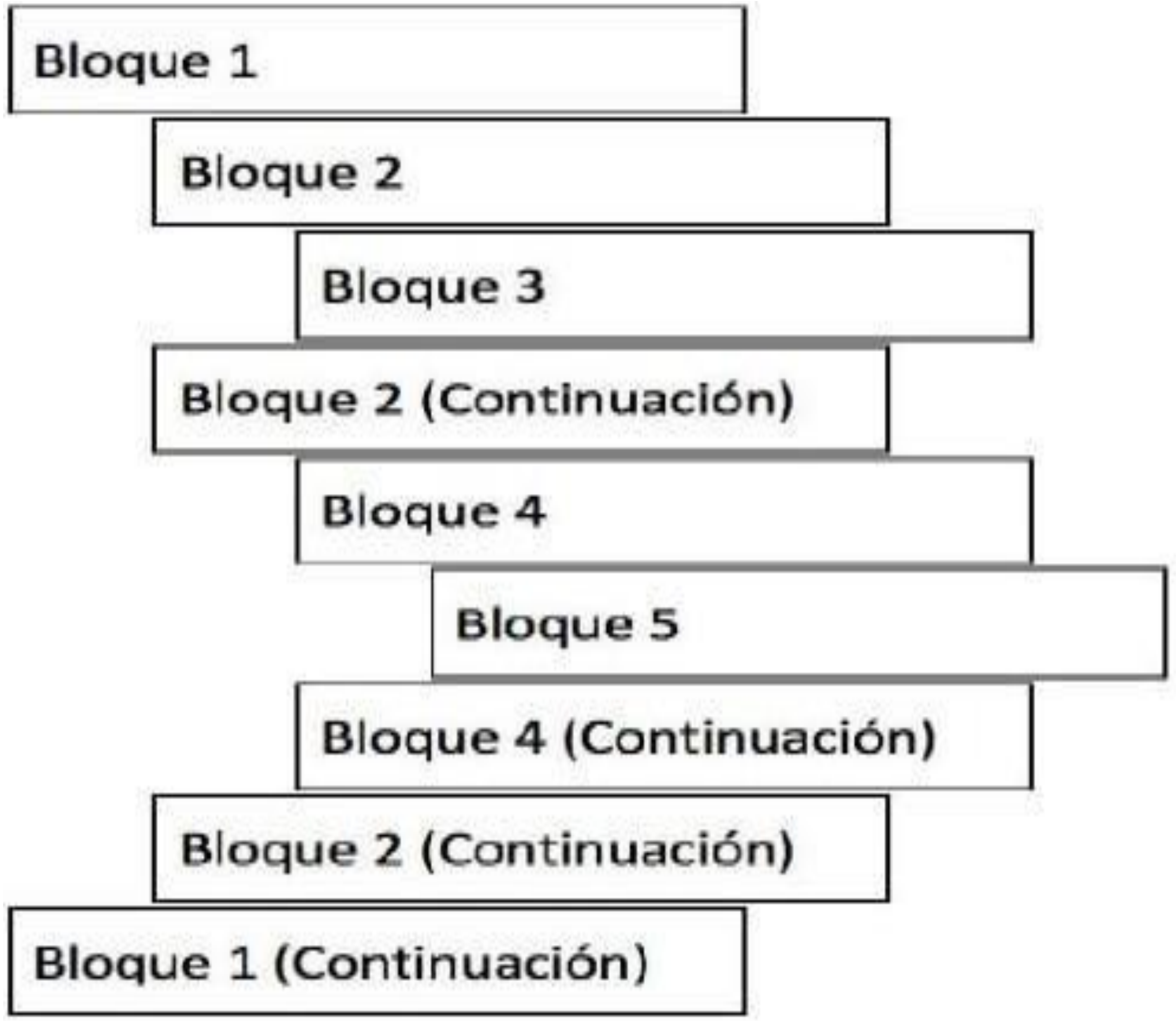
```
# not in
```

```
frutas2 not in frutas1
```

```
False
```

```
frutas3 = "melocoton"
frutas3 not in frutas1
```

```
True
```



Bloque 1

Bloque 2

Bloque 3

Bloque 2 (Continuación)

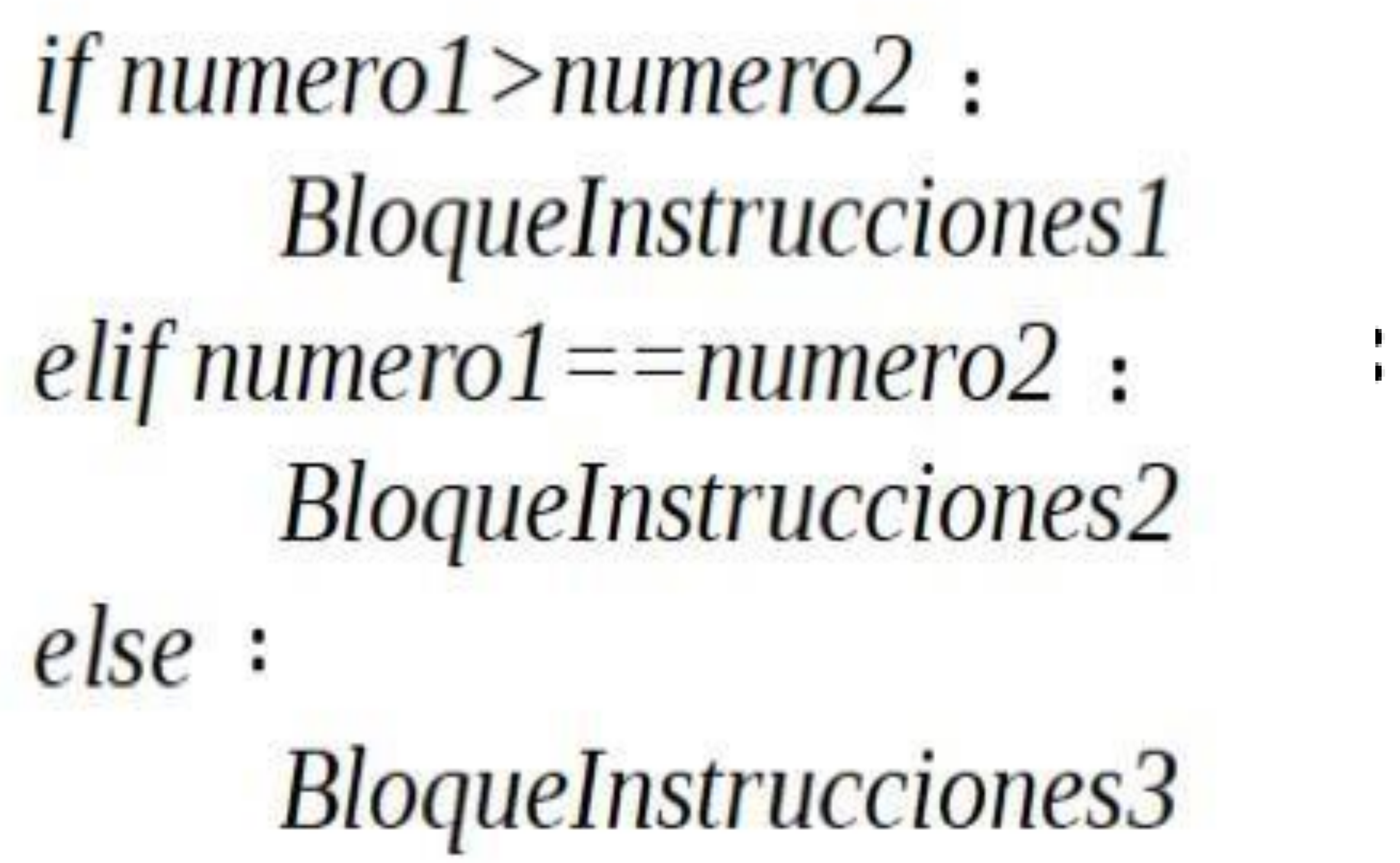
Bloque 4

Bloque 5

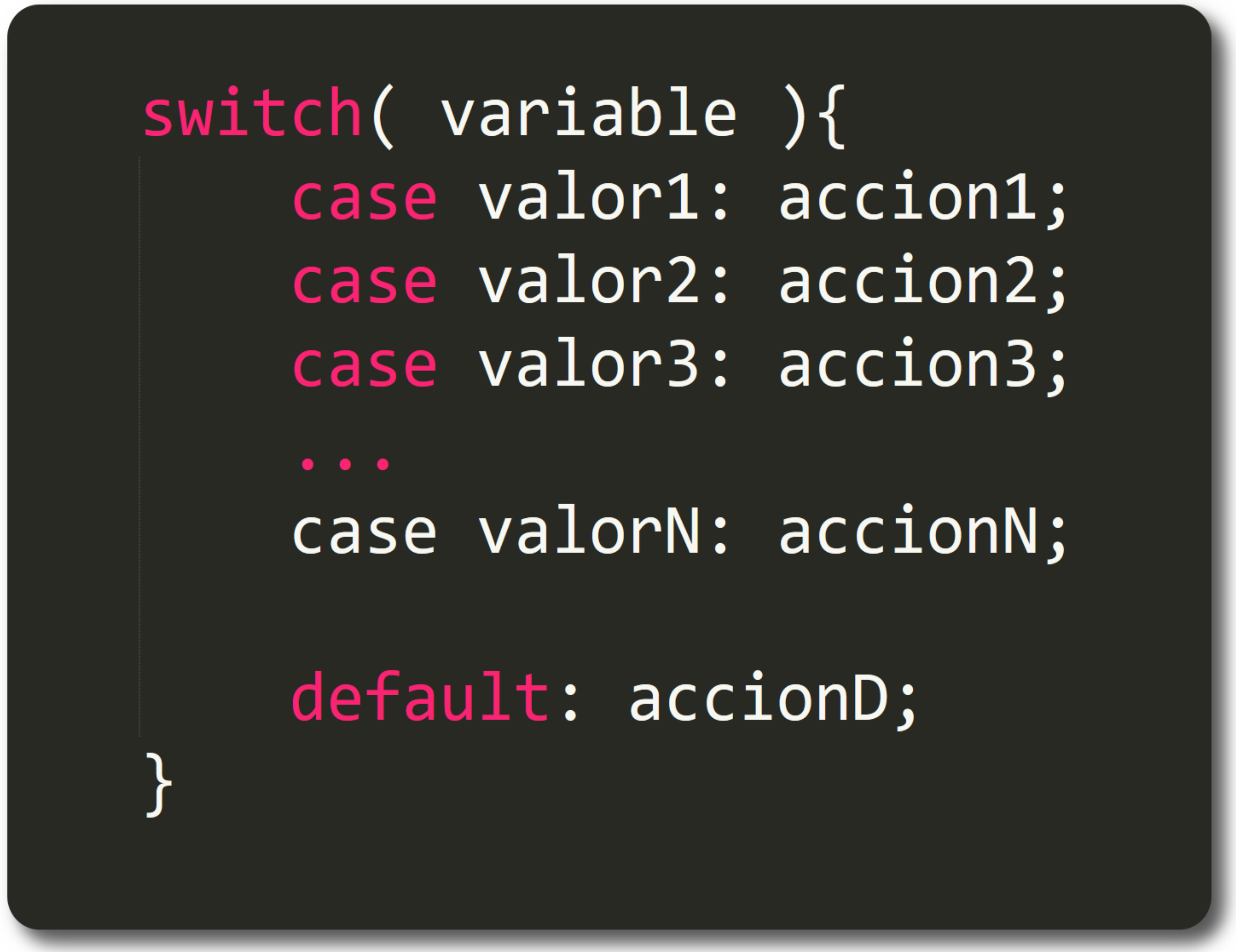
Bloque 4 (Continuación)

Bloque 2 (Continuación)

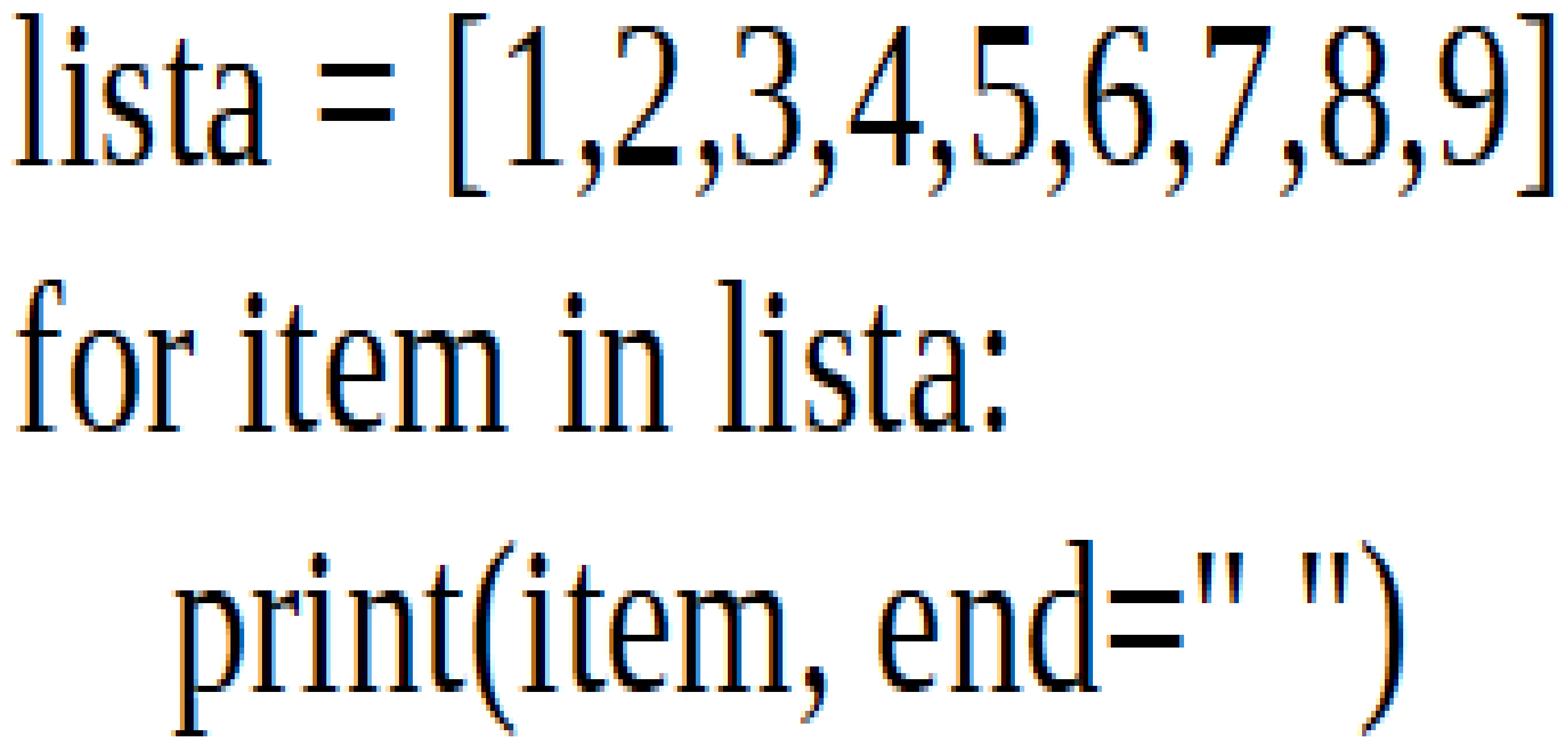
Bloque 1 (Continuación)



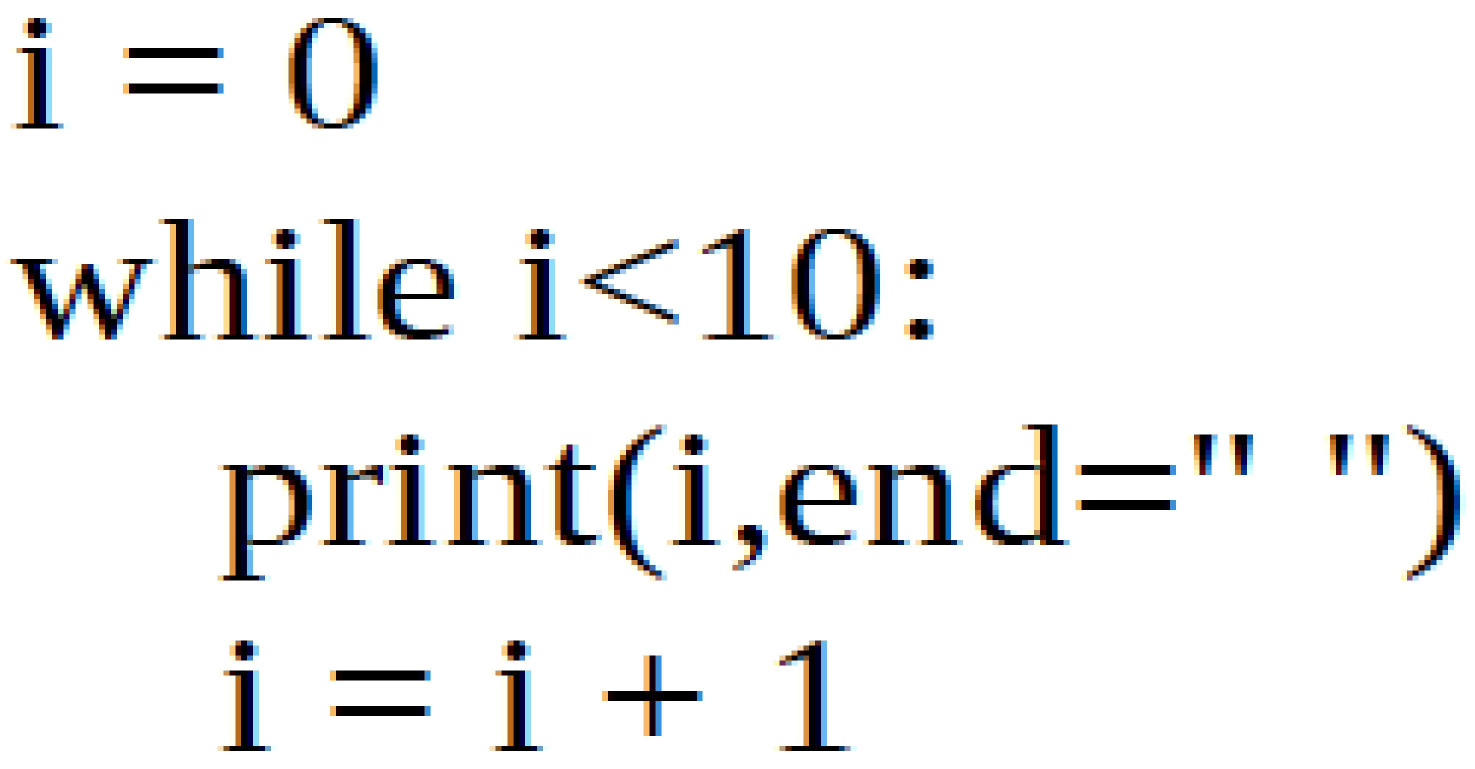
```
if numero1>numero2 :
        BloqueInstrucciones1
elif numero1==numero2 :
        BloqueInstrucciones2
else :
        BloqueInstrucciones3
```



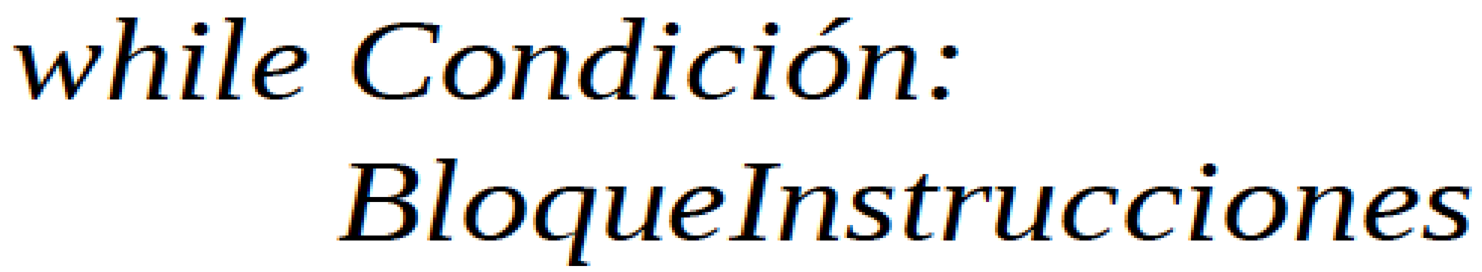
```
switch( variable ){
    case valor1: accion1;
    case valor2: accion2;
    case valor3: accion3;
    ...
    case valorN: accionN;

    default: accionD;
}
```



```
lista = [1,2,3,4,5,6,7,8,9]
for item in lista:
    print(item, end=" ")
```



```
i = 0
while i<10:
    print(i,end=" ")
    i = i + 1
```



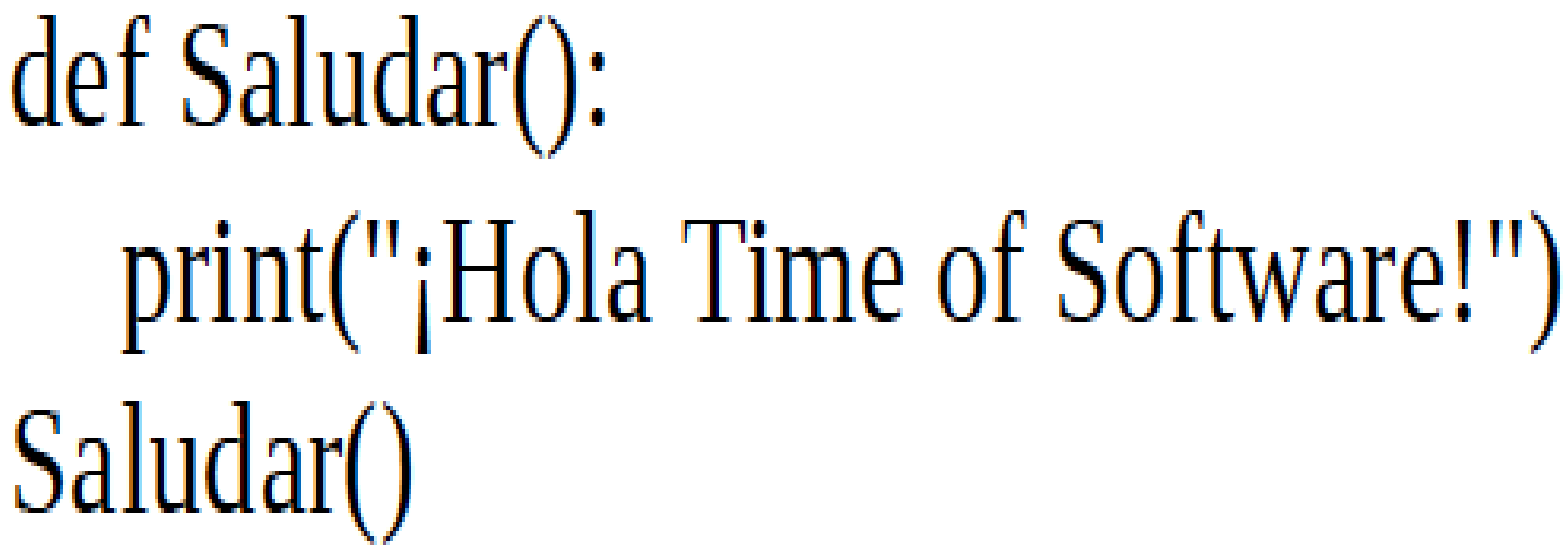
```
while Condición:
        BloqueInstrucciones
```

```
i = 0
while i<10:
    print(i,end=" ")
    i = i + 1
```

```
while Condición:
        BloqueInstrucciones
```

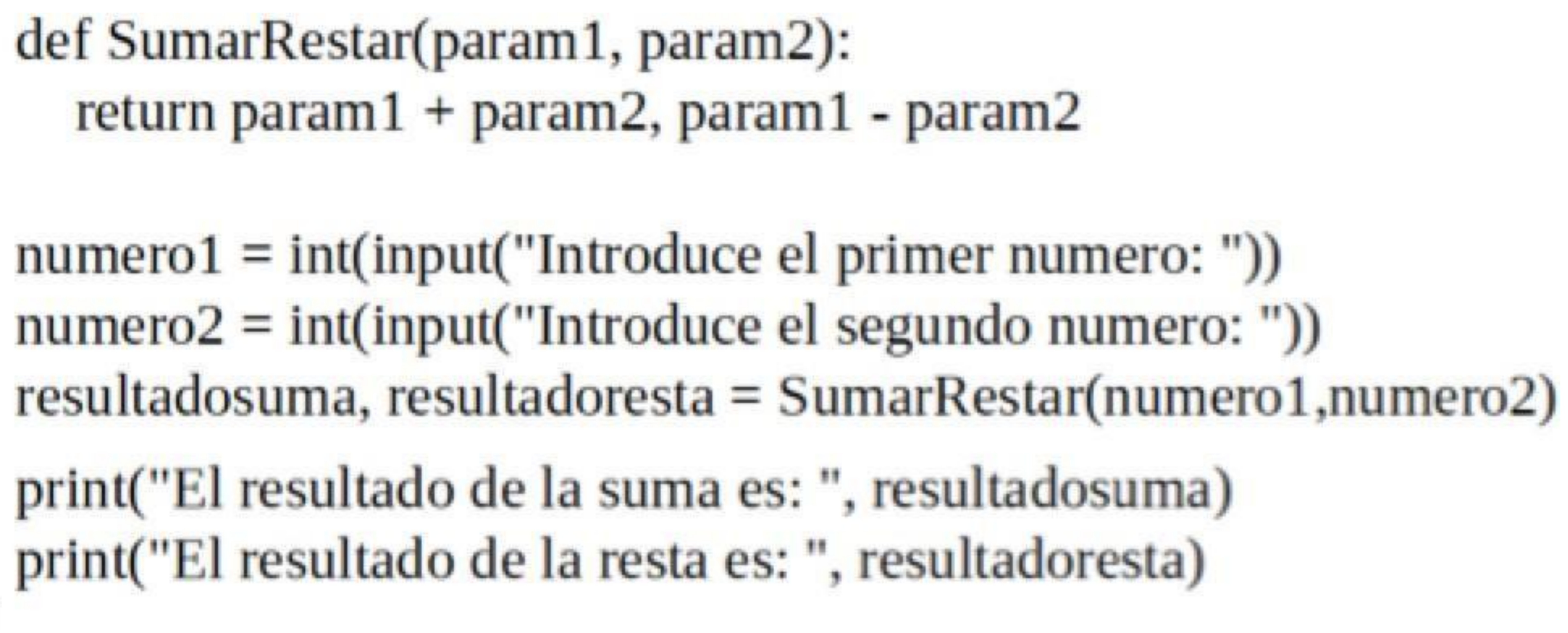
Functions and Modularity

FUNCTION DEFINITION



```
def Saludar():
    print("¡Hola Time of Software!")
Saludar()
```

MORE THAN ONE RETURN

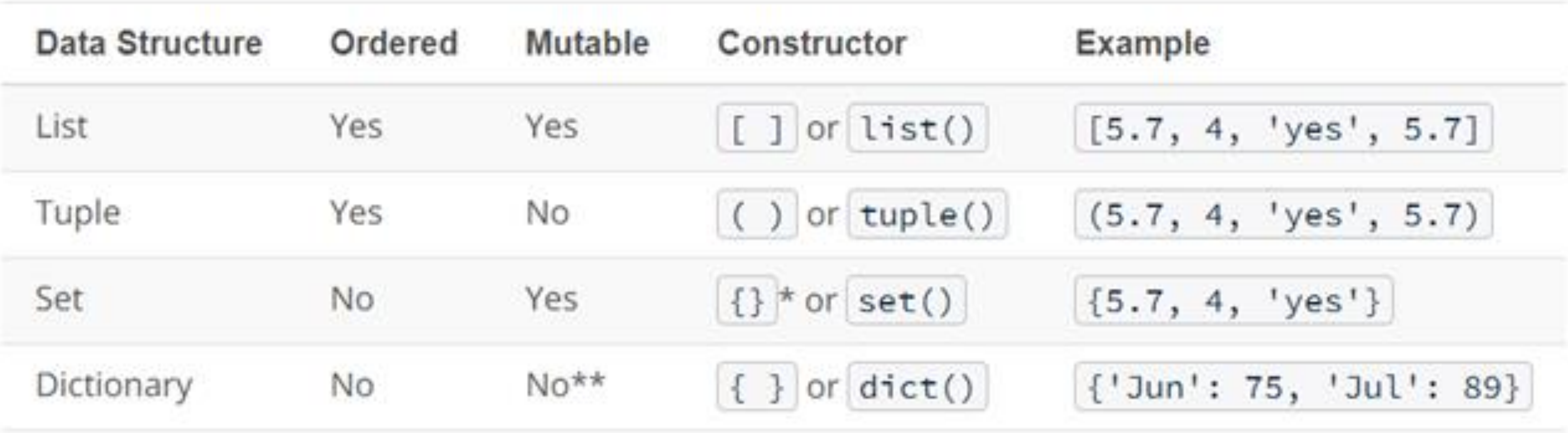


```
def SumarRestar(param1, param2):
    return param1 + param2, param1 - param2

numero1 = int(input("Introduce el primer numero: "))
numero2 = int(input("Introduce el segundo numero: "))
resultadosuma, resultadoresta = SumarRestar(numero1,numero2)
print("El resultado de la suma es: ", resultadosuma)
print("El resultado de la resta es: ", resultadoresta)
```

*ARGS

```
def Sumar(*valores):
    resultado = 0
    for item in valores:
        resultado = resultado + item
    return resultado

resultado = Sumar(23,56,3,89,78,455)
print("El resultado de la suma es: ", resultado)
```

Data Structures in Python



Data Structure	Ordered	Mutable	Constructor	Example
List	Yes	Yes	`[ ]` or `list()`	`[5.7, 4, 'yes', 5.7]`
Tuple	Yes	No	`( )` or `tuple()`	`(5.7, 4, 'yes', 5.7)`
Set	No	Yes	`{}`* or `set()`	`{5.7, 4, 'yes'}`
Dictionary	No	No**	`{ }` or `dict()`	`{'Jun': 75, 'Jul': 89}`

Tecnofor

File Handling

FUNCTIONS

The four primary functions used for file handling in Python are:

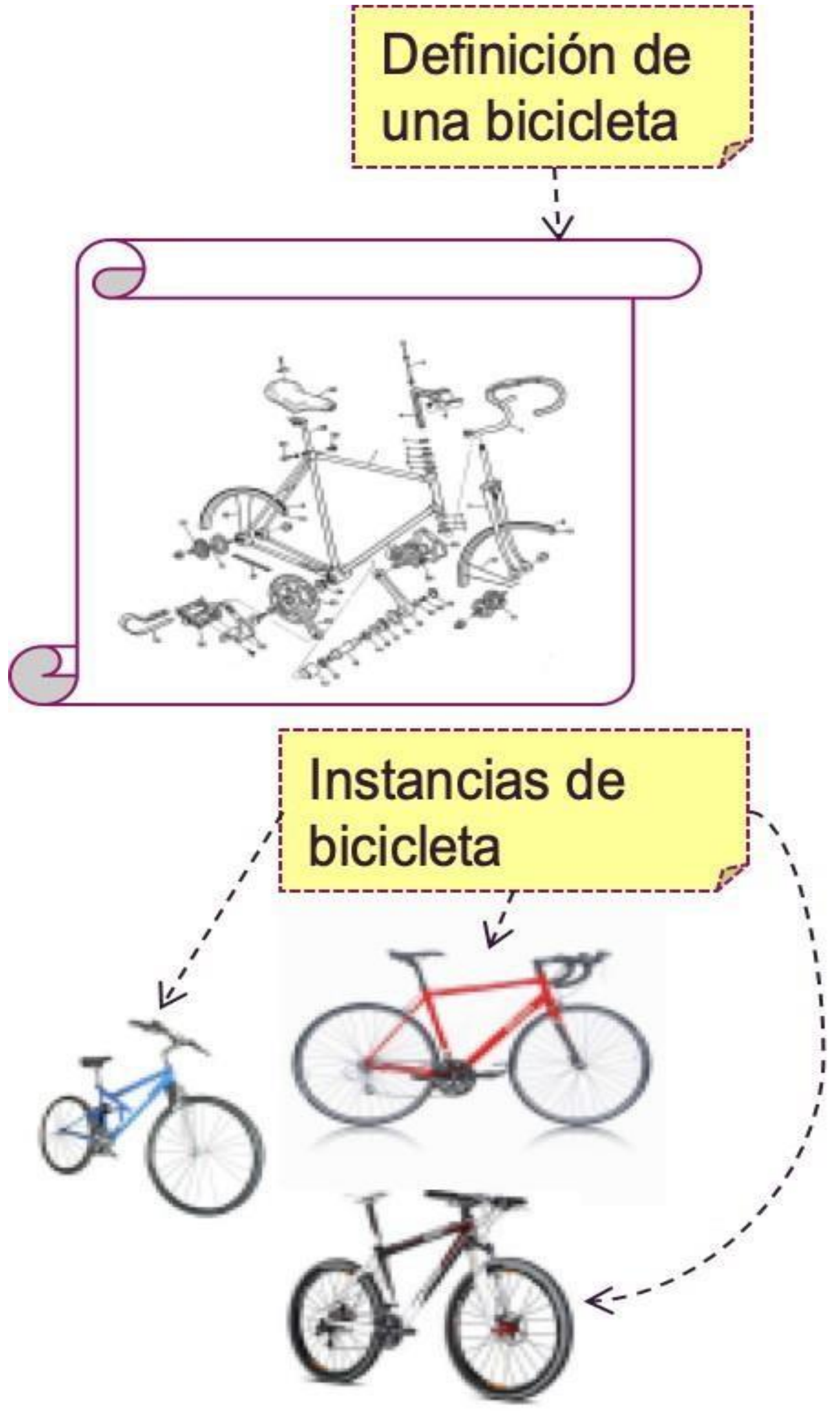
- open() : Opens a file and returns a file object.
- read() : Reads data from a file.
- write() : Writes data to a file.
- close() : Closes the file, releasing its resources.

MODES

1.r: open an existing file for a read operation.
2.w: open an existing file for a write operation. If the file already contains some data, then it will be overridden but if the file is not present then it creates the file as well.
3.a: open an existing file for append operation. It won't override existing data.
4.r+: To read and write data into the file. This mode does not override the existing data, but you can modify the data starting from the beginning of the file.
5.w+: To write and read data. It overwrites the previous file if one exists, it will truncate the file to zero length or create a file if it does not exist.
6.a+: To append and read data from the file. It won't override existing data.

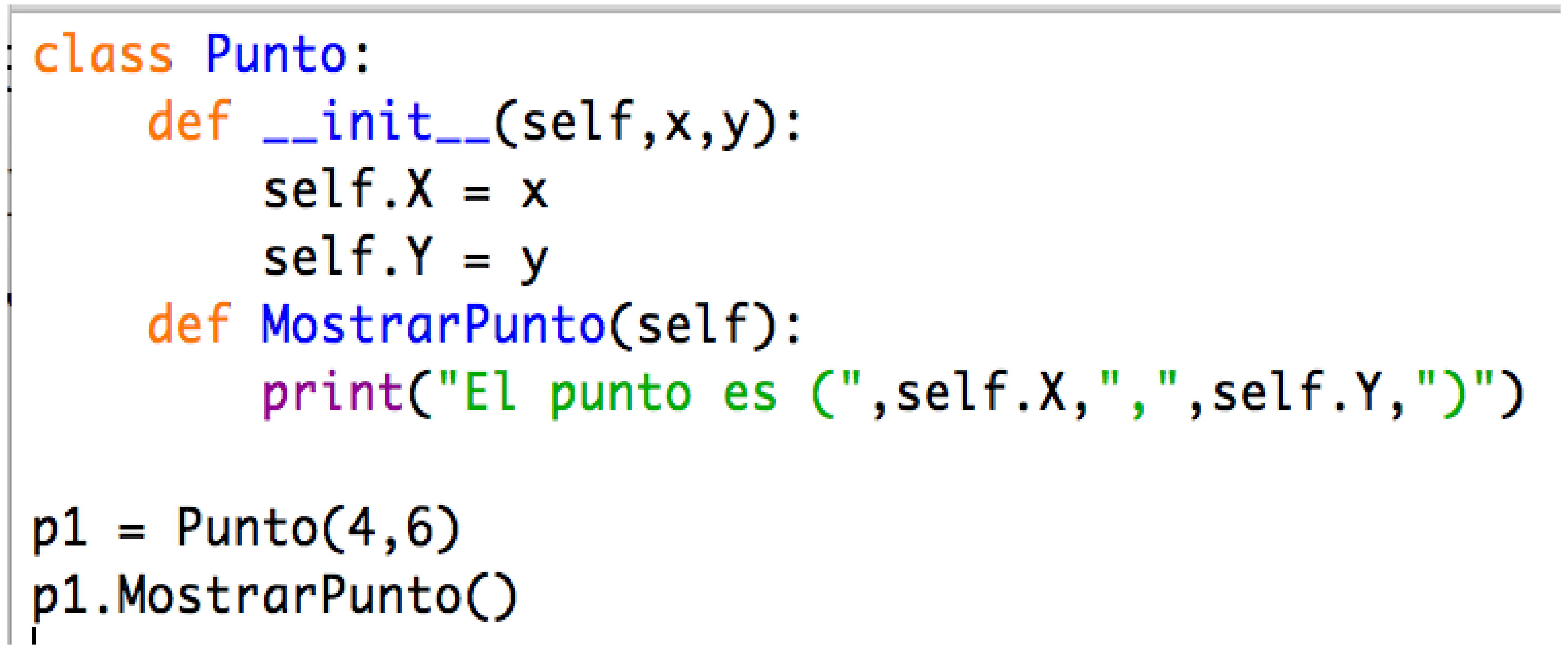
Tecnofor

Introduction to Object-Oriented Programming (OOP)

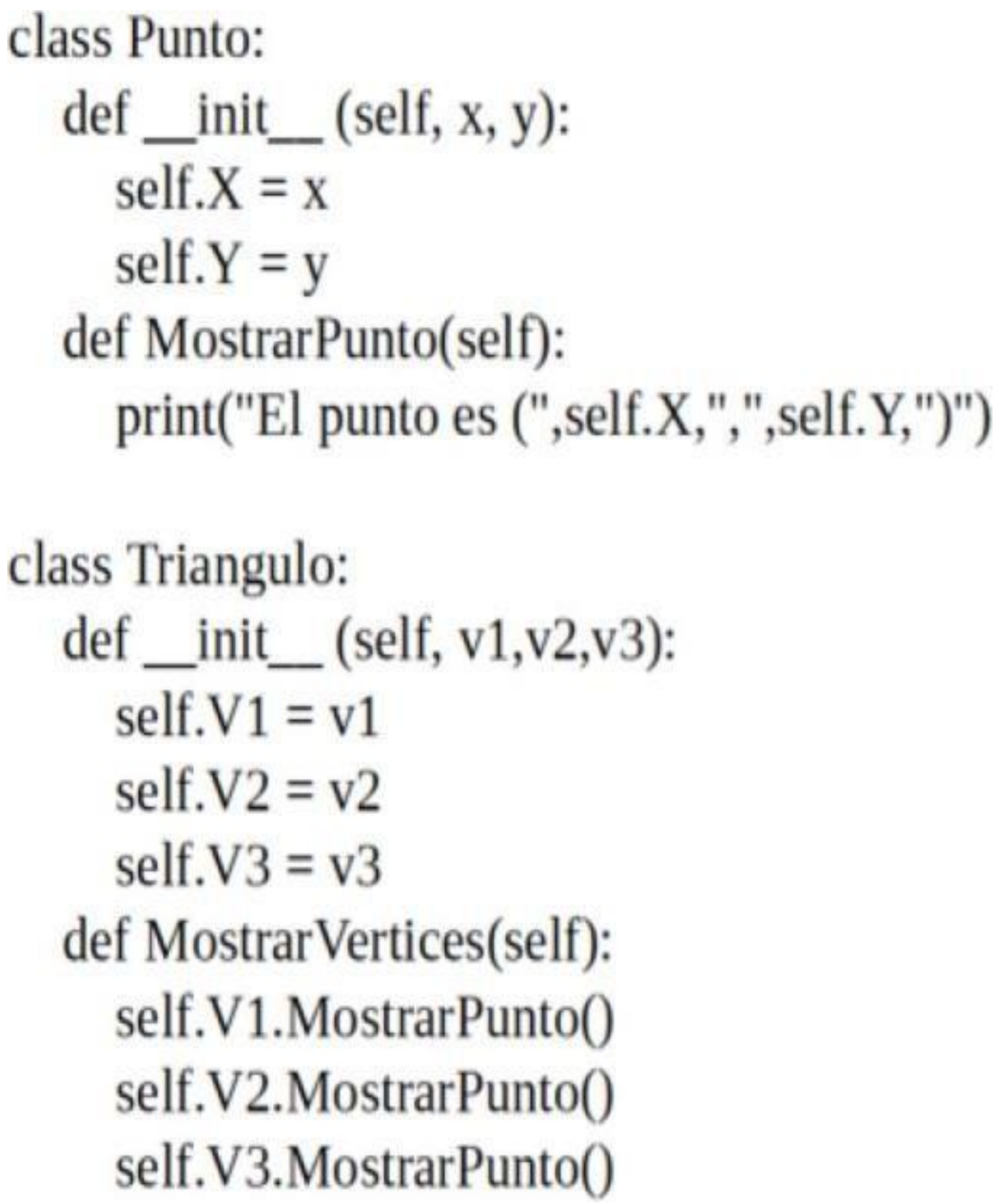


Definición de una bicicleta

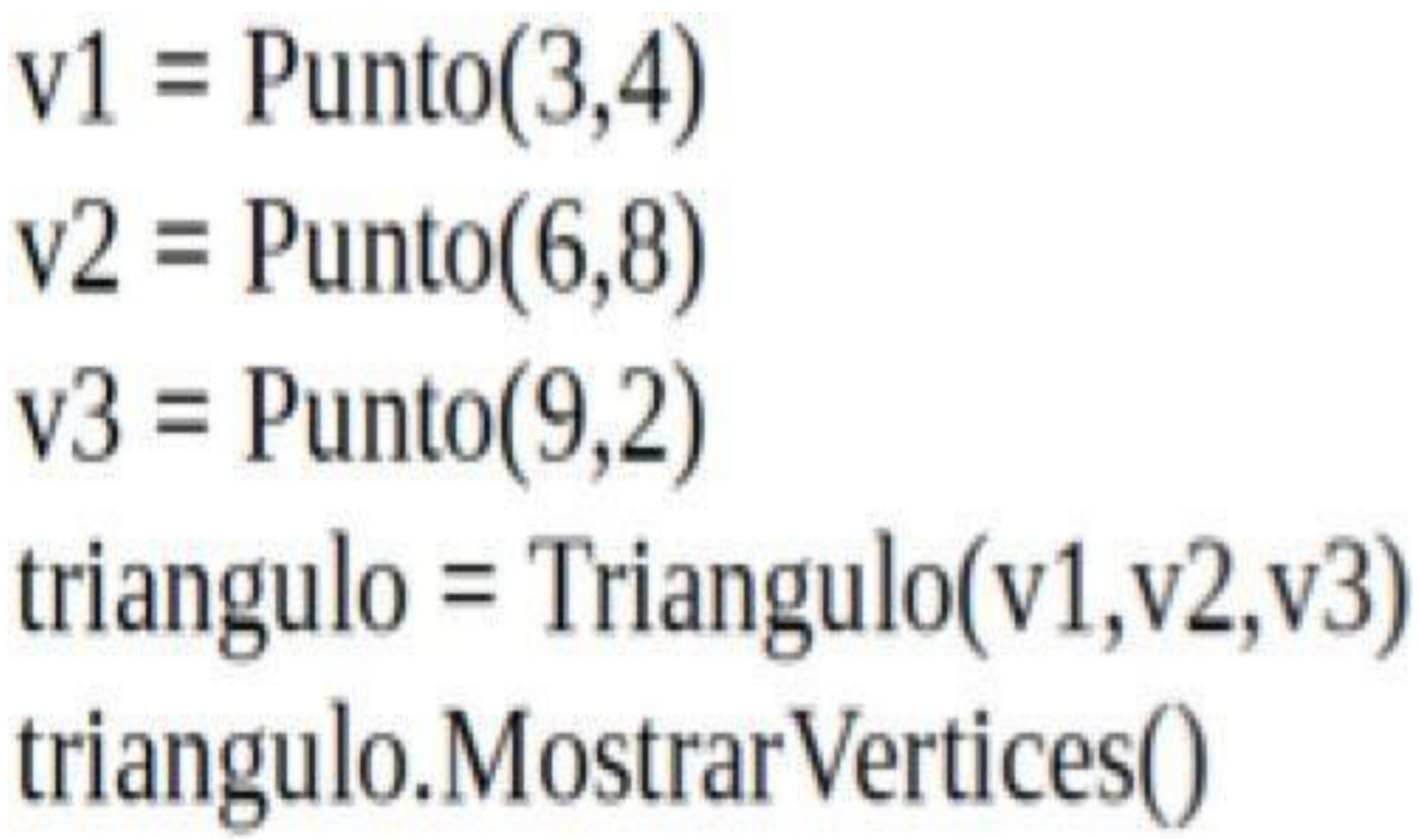
Instancias de bicicleta



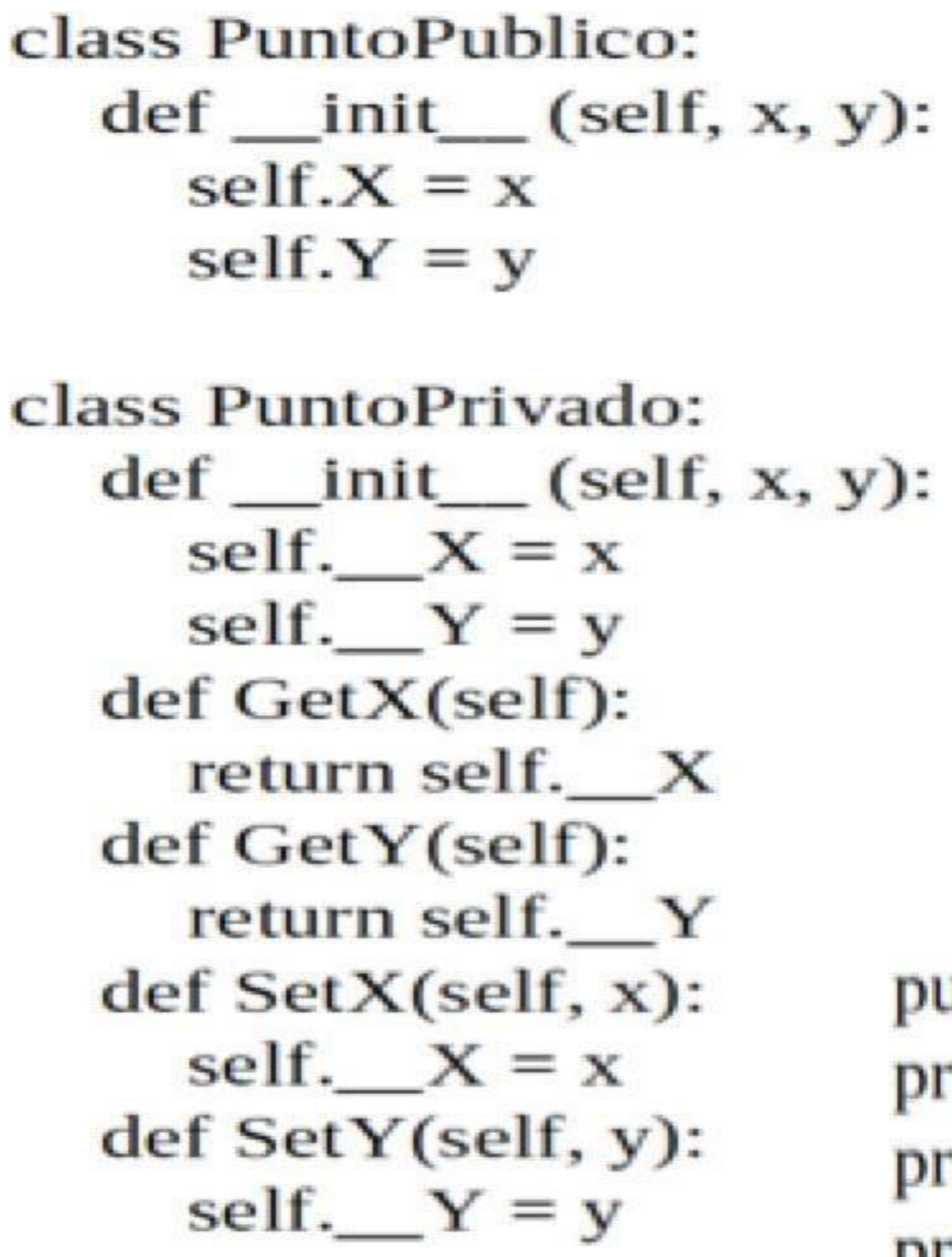
```
class Punto:
    def __init__(self,x,y):
        self.X = x
        self.Y = y
    def MostrarPunto(self):
        print("El punto es (",self.X,",",self.Y,")")

p1 = Punto(4,6)
p1.MostrarPunto()
```



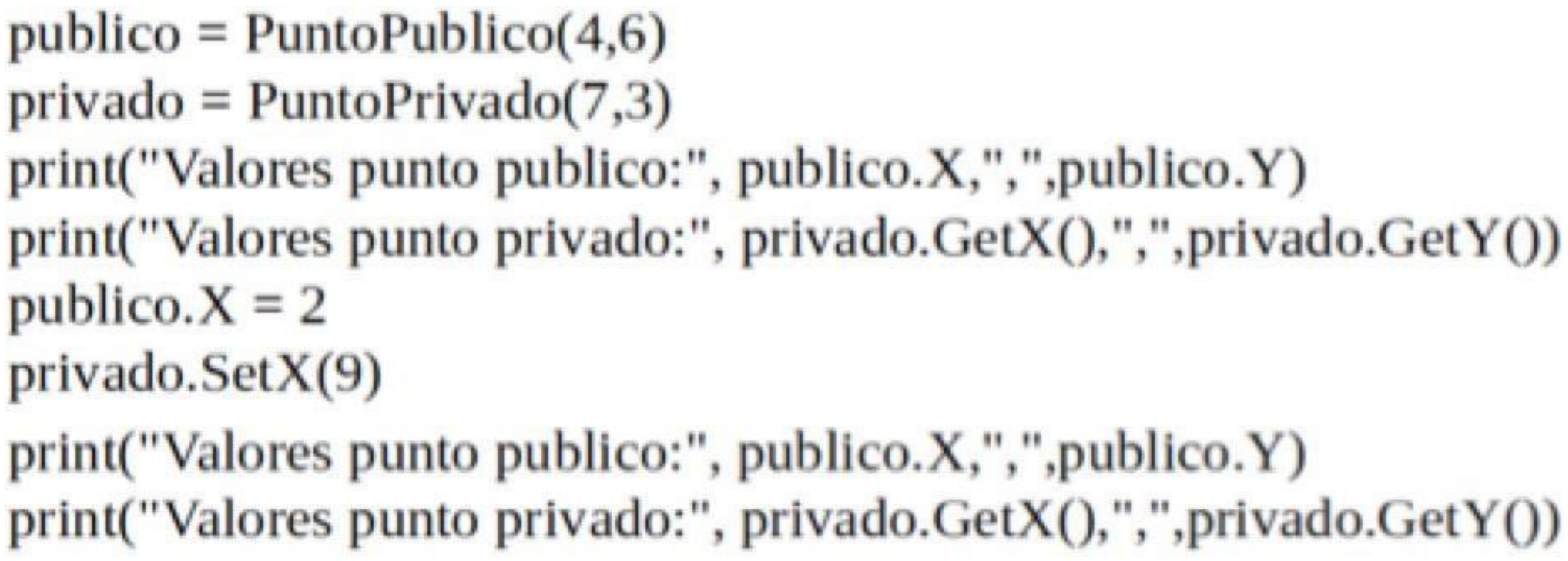
```
class Punto:
   def __init__(self, x, y):
      self.X = x
      self.Y = y
   def MostrarPunto(self):
      print("El punto es (",self.X,",",self.Y,")")

class Triangulo:
   def __init__(self, v1,v2,v3):
      self.V1 = v1
      self.V2 = v2
      self.V3 = v3
   def MostrarVertices(self):
      self.V1.MostrarPunto()
      self.V2.MostrarPunto()
      self.V3.MostrarPunto()
```



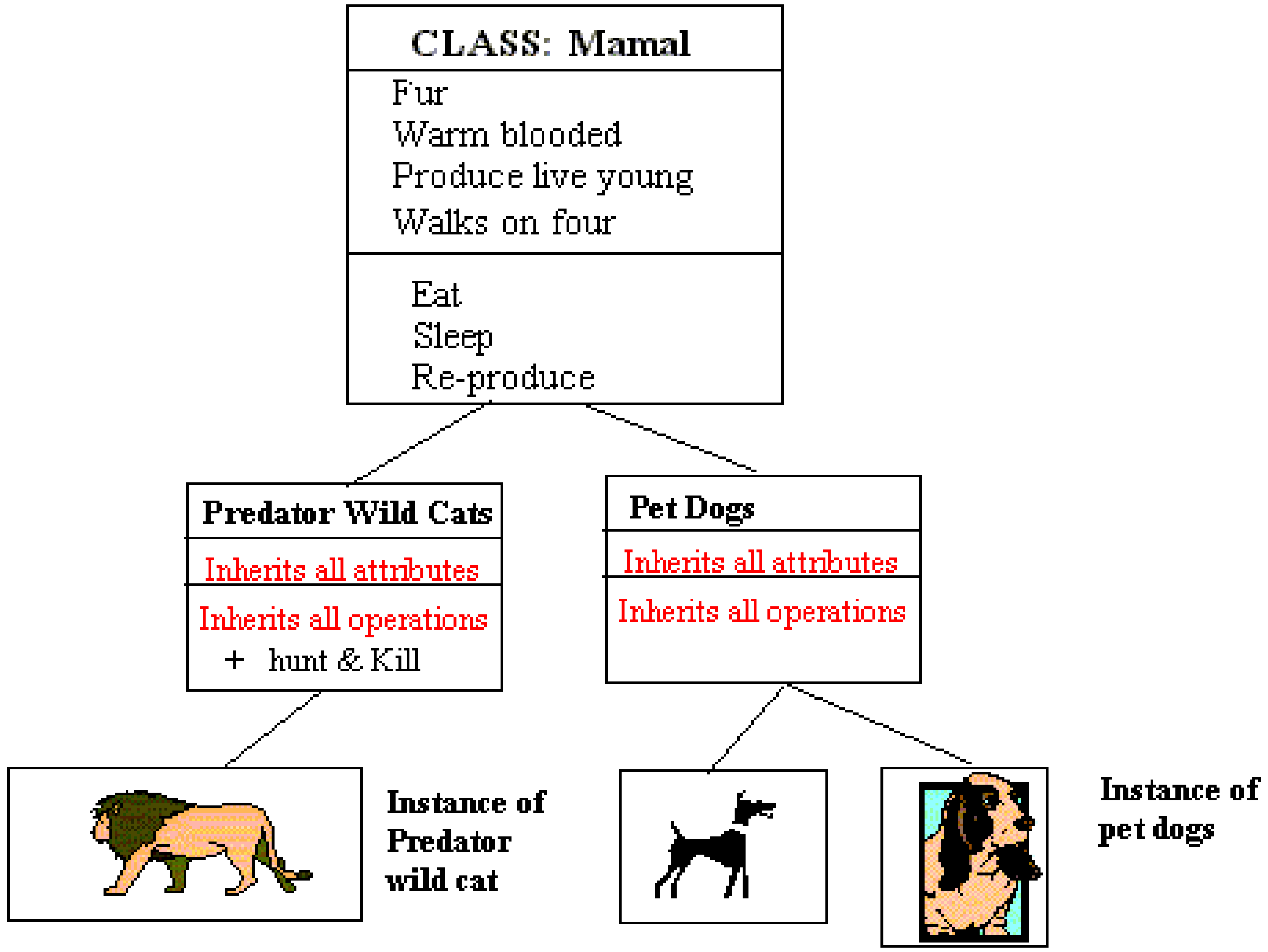
```
v1 = Punto(3,4)
v2 = Punto(6,8)
v3 = Punto(9,2)
triangulo = Triangulo(v1,v2,v3)
triangulo.MostrarVertices()
```



```
class PuntoPublico:
   def __init__ (self, x, y):
      self.X = x
      self.Y = y

class PuntoPrivado:
   def __init__ (self, x, y):
      self.__X = x
      self.__Y = y
   def GetX(self):
      return self.__X
   def GetY(self):
      return self.__Y
   def SetX(self, x):
      self.__X = x
   def SetY(self, y):
      self.__Y = y
```



```
publico = PuntoPublico(4,6)
privado = PuntoPrivado(7,3)
print("Valores punto publico:", publico.X,",",publico.Y)
print("Valores punto privado:", privado.GetX(),",",privado.GetY())
publico.X = 2
privado.SetX(9)
print("Valores punto publico:", publico.X,",",publico.Y)
print("Valores punto privado:", privado.GetX(),",",privado.GetY())
```



CLASS: Mamal

Fur
Warm blooded
Produce live young
Walks on four

Eat
Sleep
Re-produce

Predator Wild Cats

Inherits all attributes

Inherits all operations
+ hunt & Kill

Pet Dogs

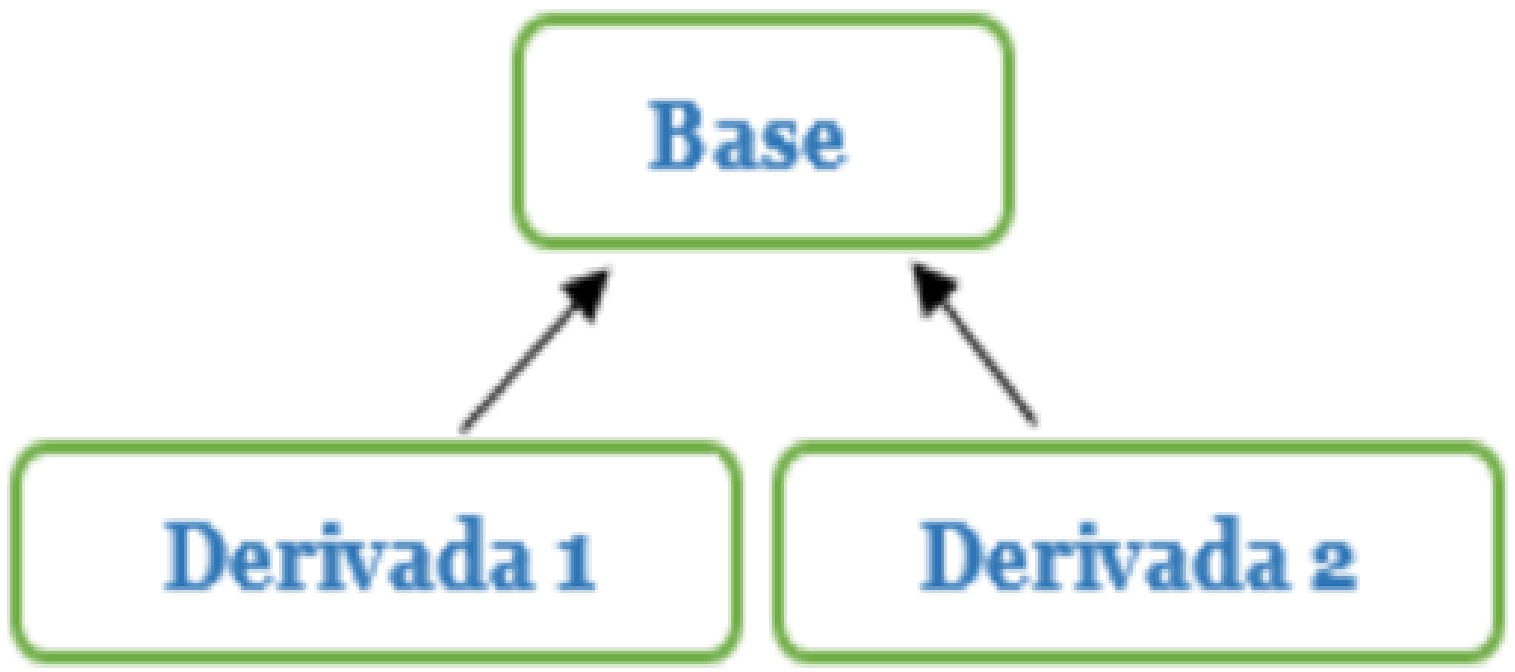
Inherits all attributes

Inherits all operations

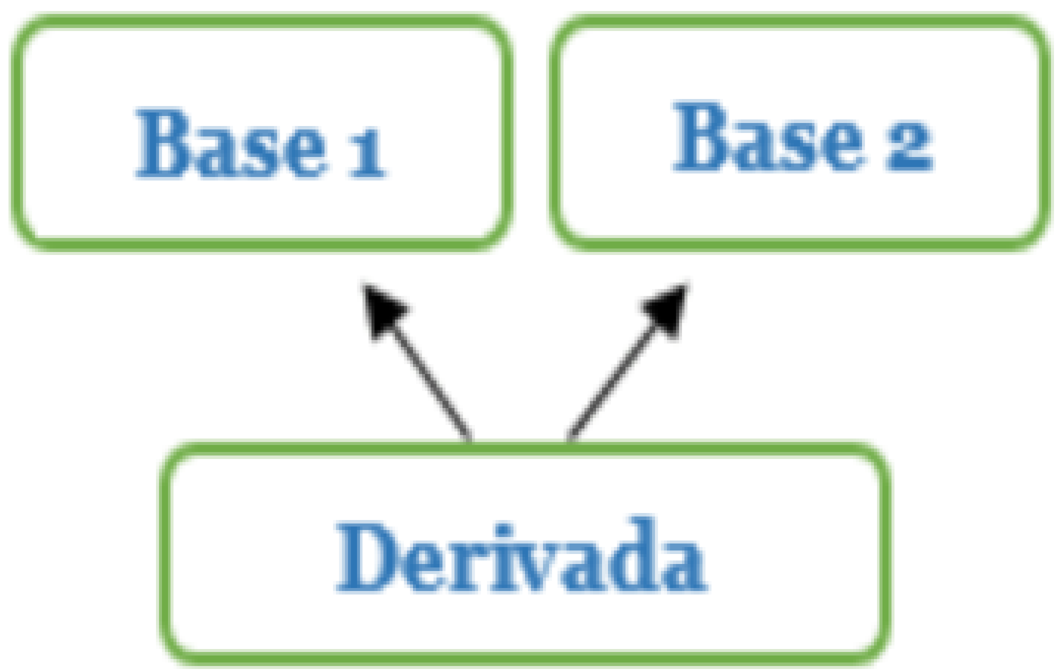
Instance of Predator wild cat

Instance of pet dogs

Herencia simple

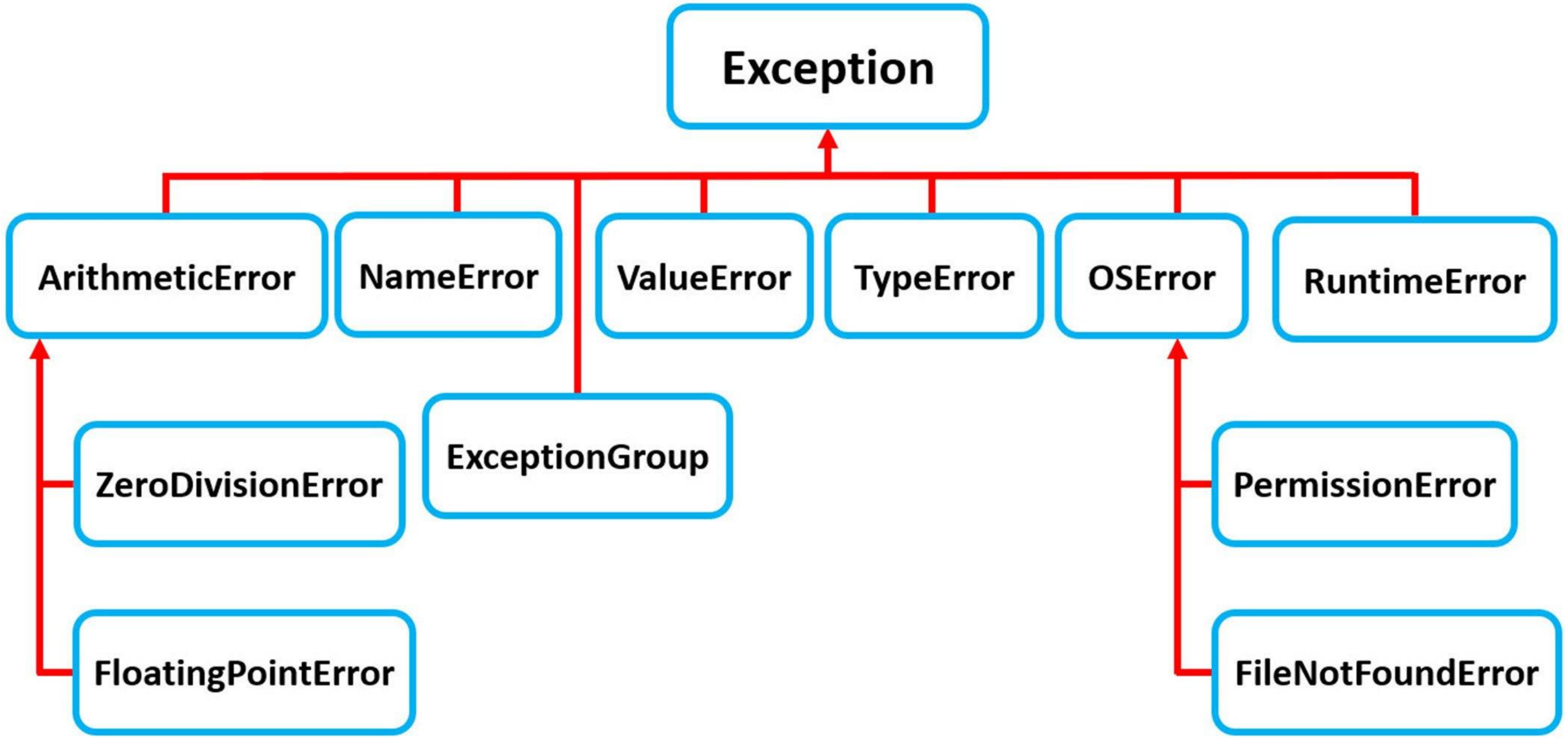


Herencia múltiple



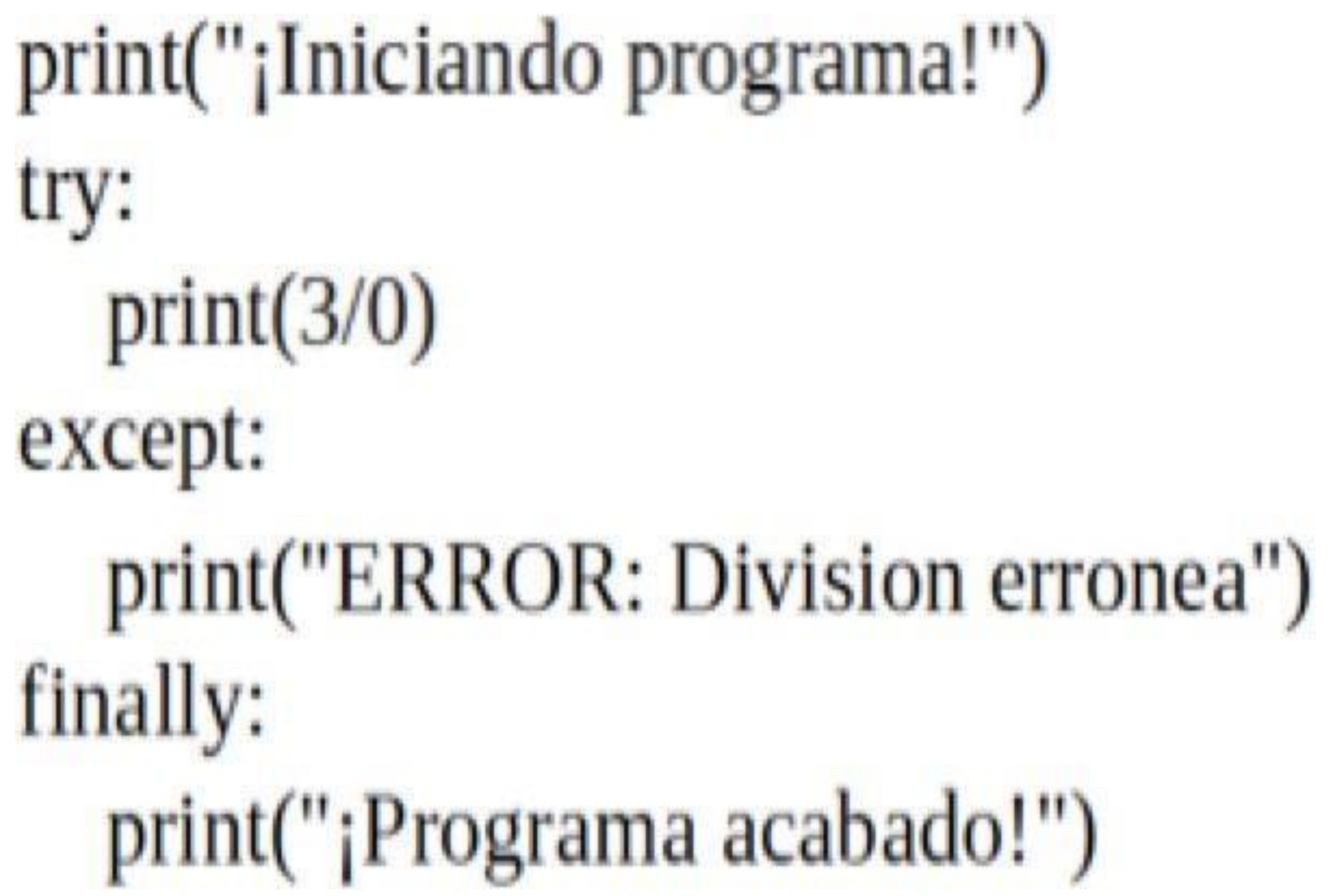
```
class Negocio(Hotel, Restaurante):
    def __init__(self):
        self.__Nombre = ""
        self.__Direccion = ""
        self.__Telefono = 0
    def SetNombre(self, nombre):
        self.__Nombre = nombre
    def SetDireccion(self, direccion):
        self.__Direccion = direccion
    def SetTelefono(self, telefono):
        self.__Telefono = telefono
    def MostrarNegocio(self):
        print("#########")
        print("Negocio:")
        print("\tNombre:", self.__Nombre)
        print("\tDireccion:", self.__Direccion)
        print("\tTelefono:", self.__Telefono)
        self.MostrarHotel()
        self.MostrarRestaurante()
        print("#########")
```

Exceptions

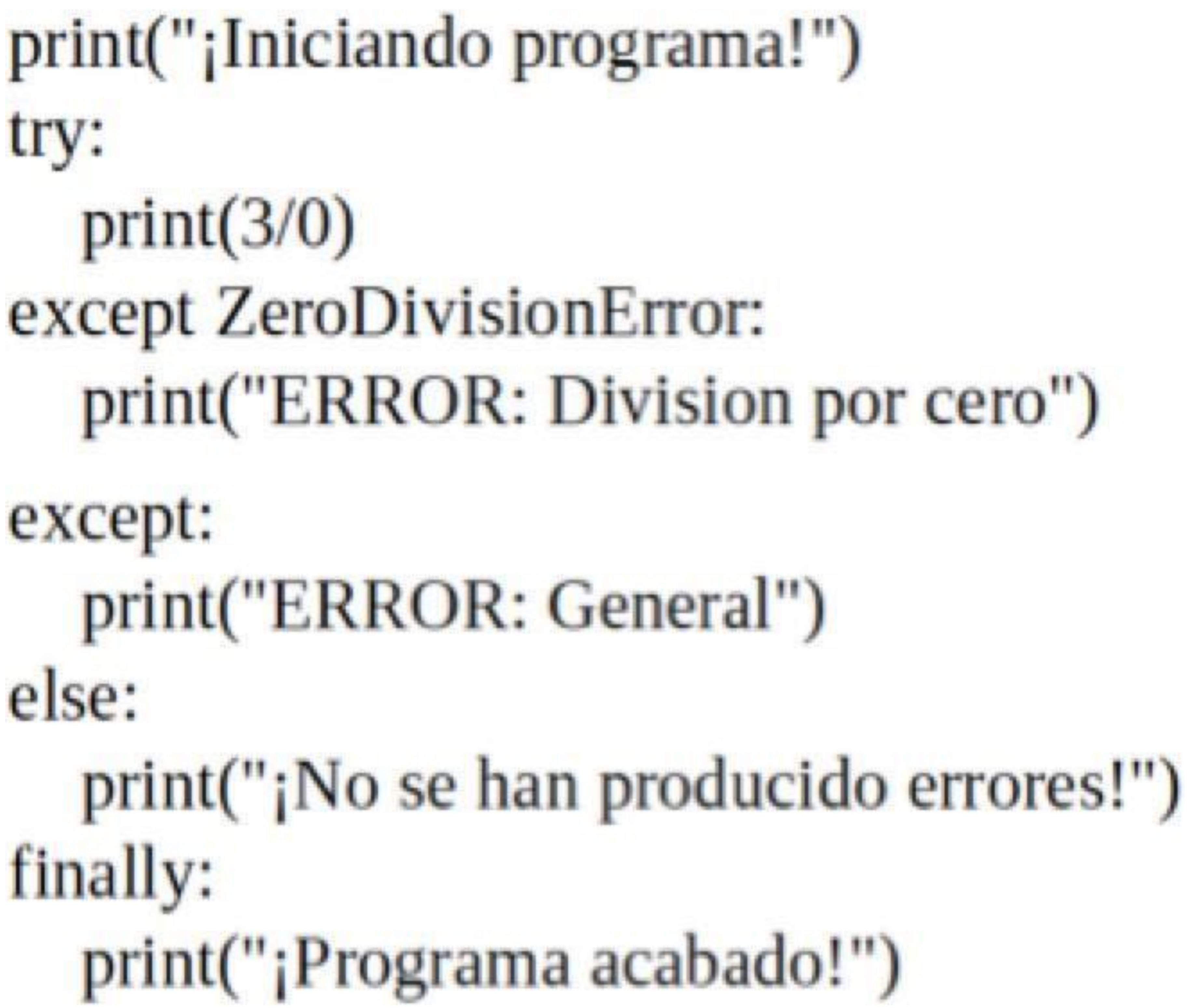


Tecnofor
by SNGULAR

```
try:
    print(3/0)
except:
    print("ERROR: Division por cero")
```



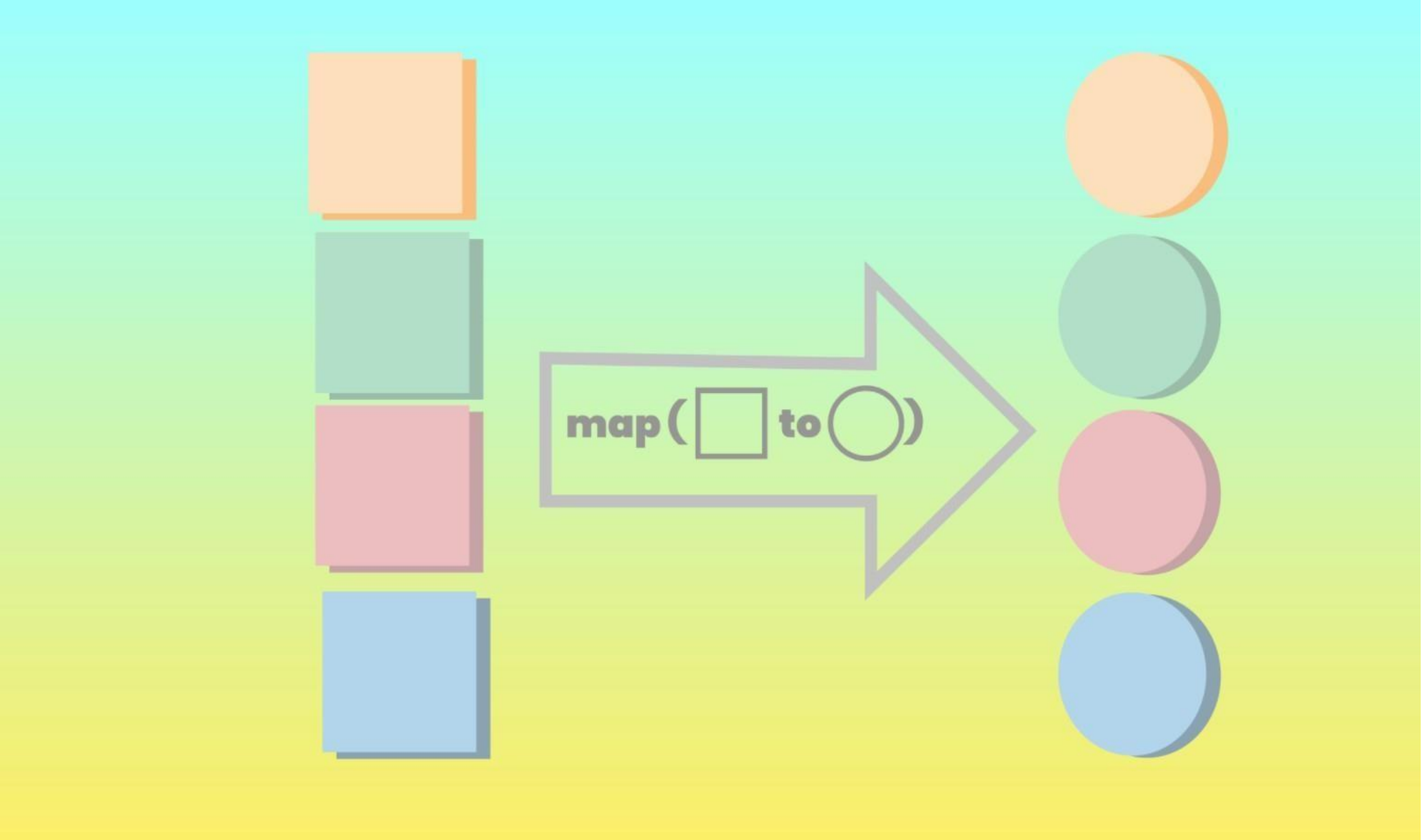
```
print("¡Iniciando programa!")
try:
    print(3/0)
except:
    print("ERROR: Division erronea")
finally:
    print("¡Programa acabado!")
print("¡Iniciando programa!")
try:
    print(3/1)
except:
    print("ERROR: Division erronea")
else:
    print("¡No se han producido errores!")
finally:
    print("¡Programa acabado!")
```

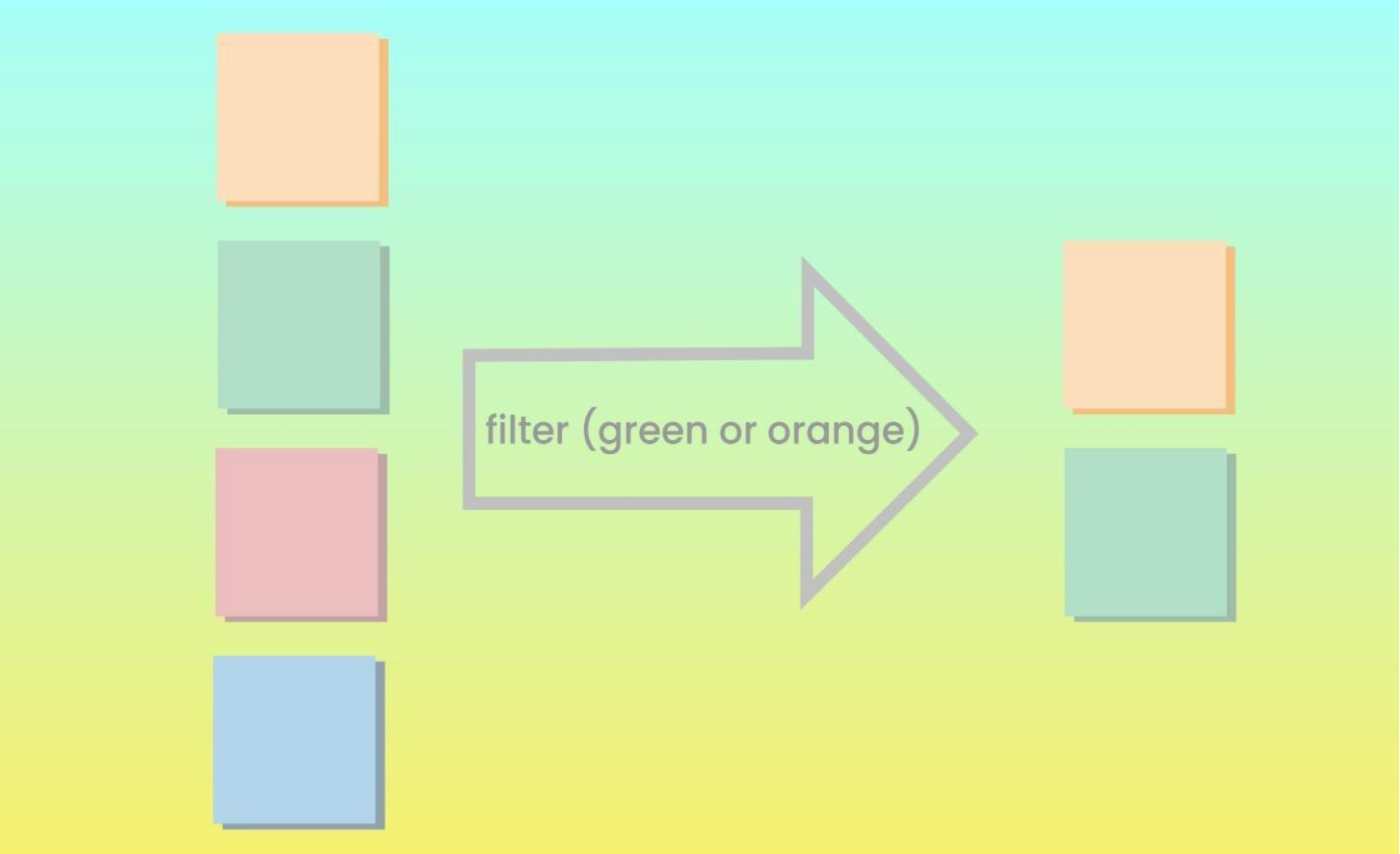


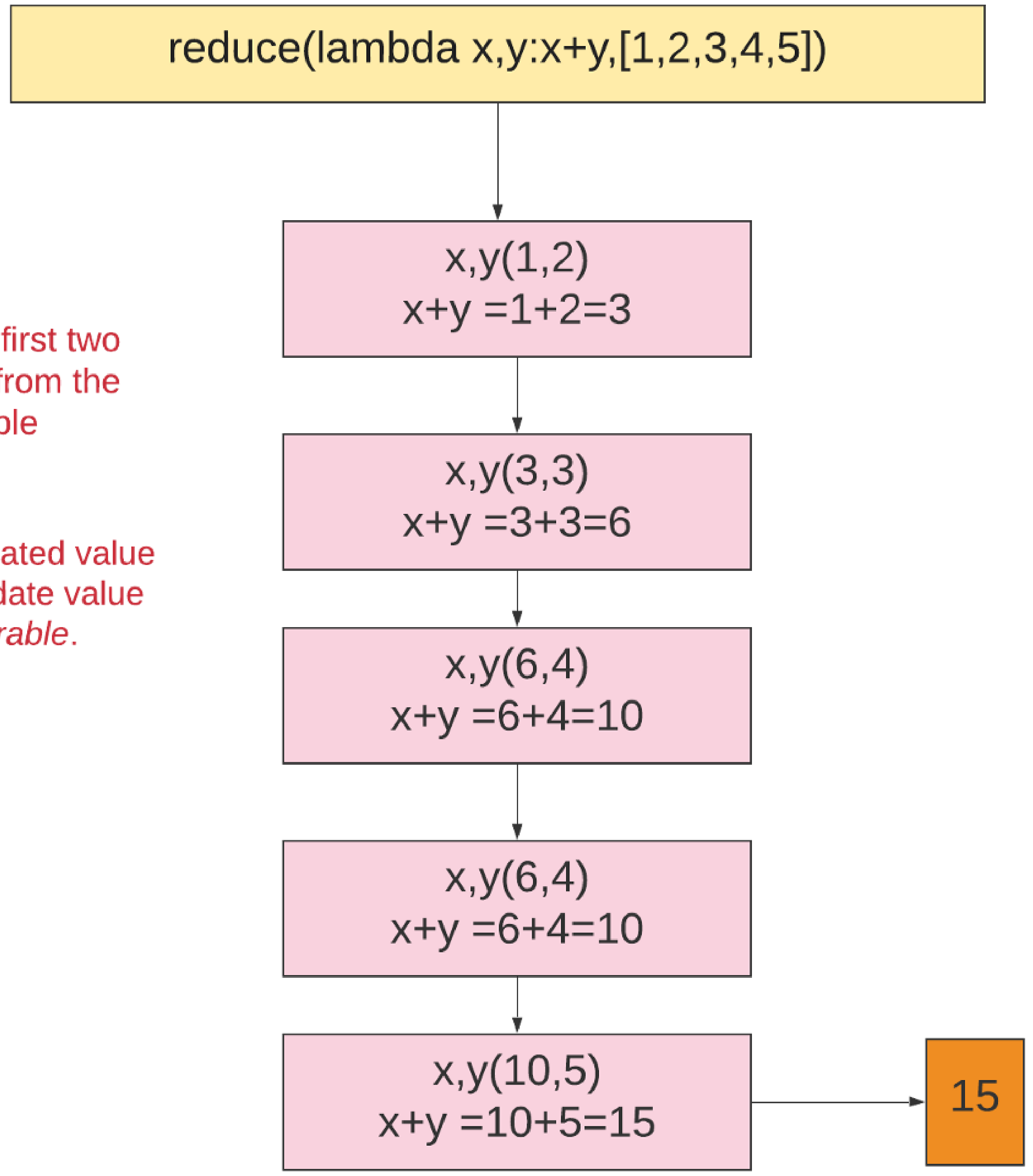
```
print("¡Iniciando programa!")
try:
    print(3/0)
except ZeroDivisionError:
    print("ERROR: Division por cero")
except:
    print("ERROR: General")
else:
    print("¡No se han producido errores!")
finally:
    print("¡Programa acabado!")
```

Tecnofor

Introduction to Functional Programming in Python


map (□ to ○)


filter (green or orange)



reduce(lambda x,y:x+y,[1,2,3,4,5])

x,y(1,2)
x+y =1+2=3

x,y(3,3)
x+y =3+3=6

x,y(6,4)
x+y =6+4=10

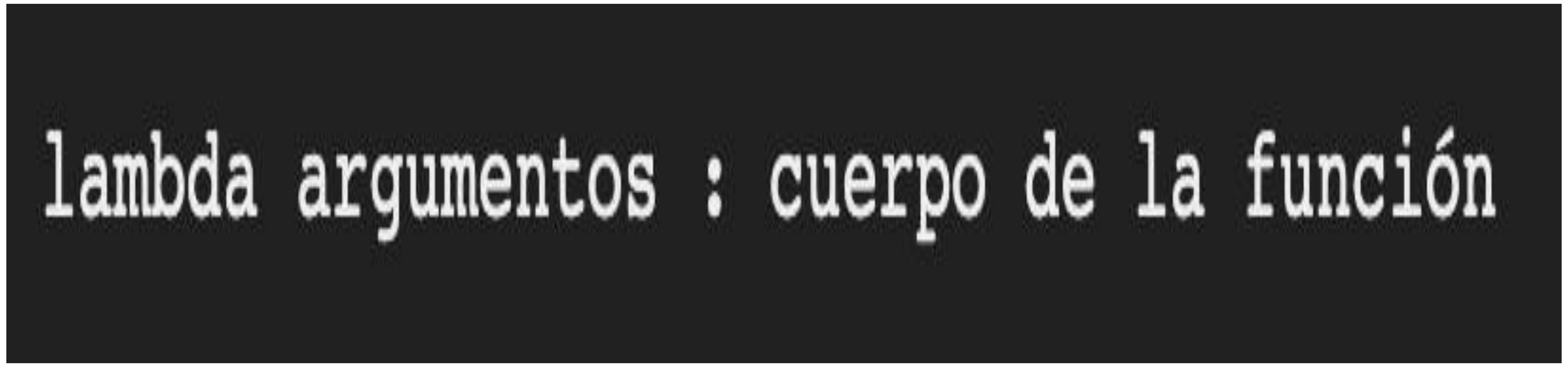
x,y(6,4)
x+y =6+4=10

x,y(10,5)
x+y =10+5=15

15

x,y -takes first two elements from the iterable

x, is the accumulated value and *y*, is the update value from the *iterable*.

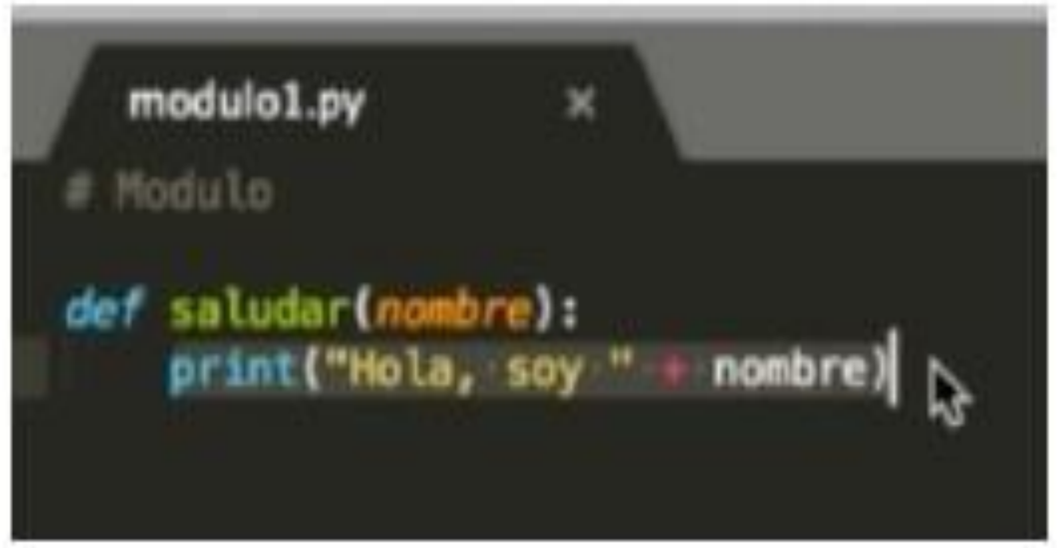


```
lambda argumentos : cuerpo de la función
```

Tecnofor

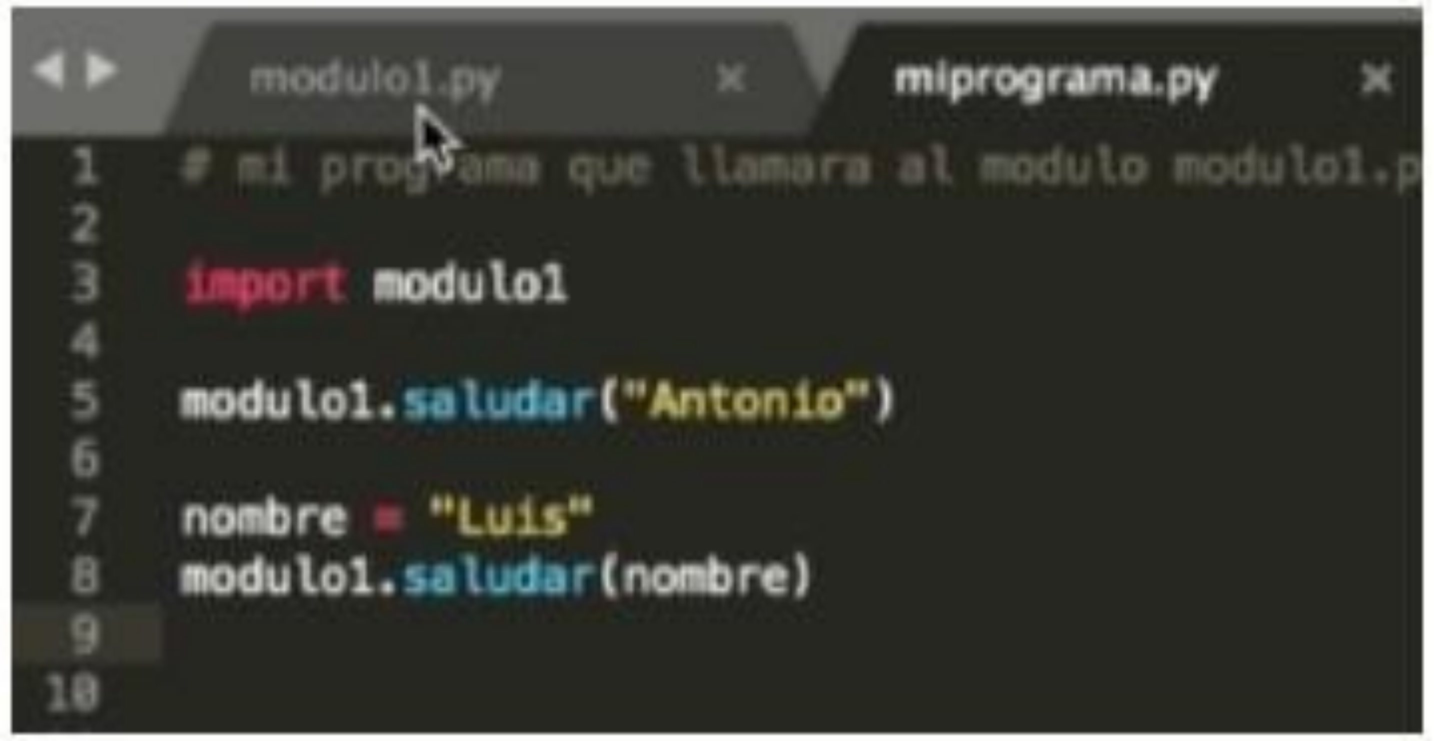
Introduction to Modules and Packages

Declaración del módulo



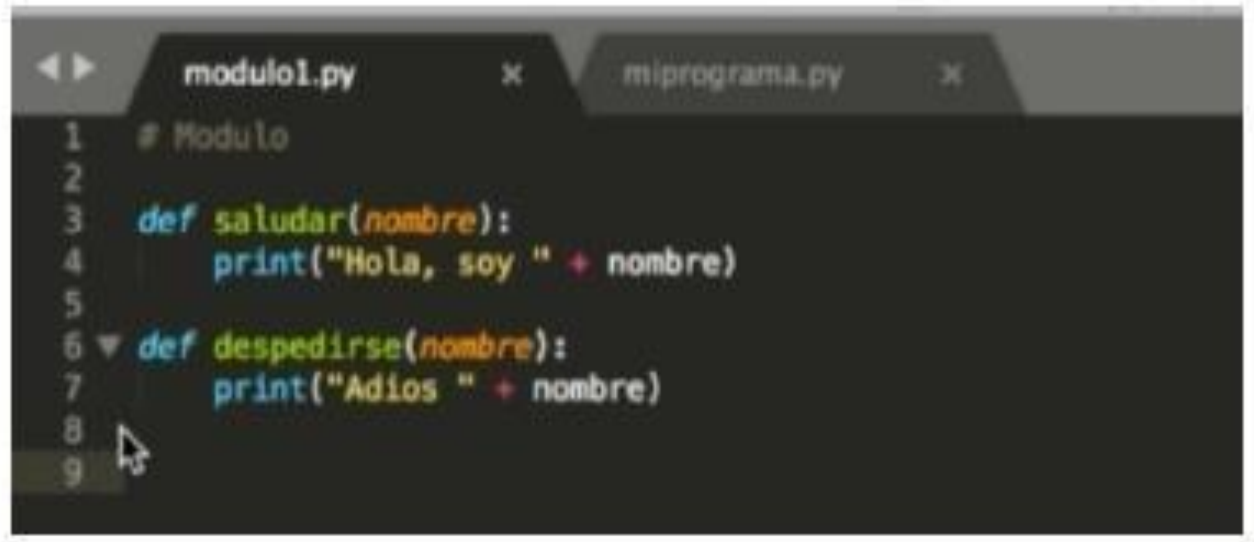
```python
# Modulo

def saludar(nombre):
    print("Hola, soy " + nombre)
```

Uso del módulo



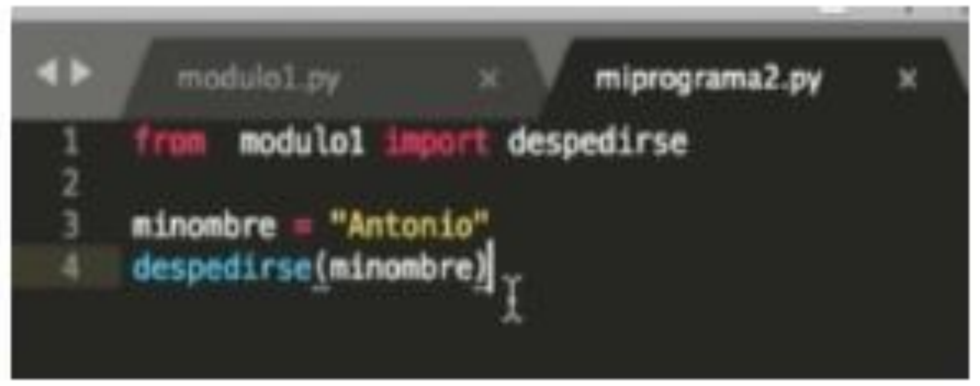
```python
# mi programa que llamara al modulo modulo1.p

import modulo1

modulo1.saludar("Antonio")

nombre = "Luis"
modulo1.saludar(nombre)
```

Declaración de otra función



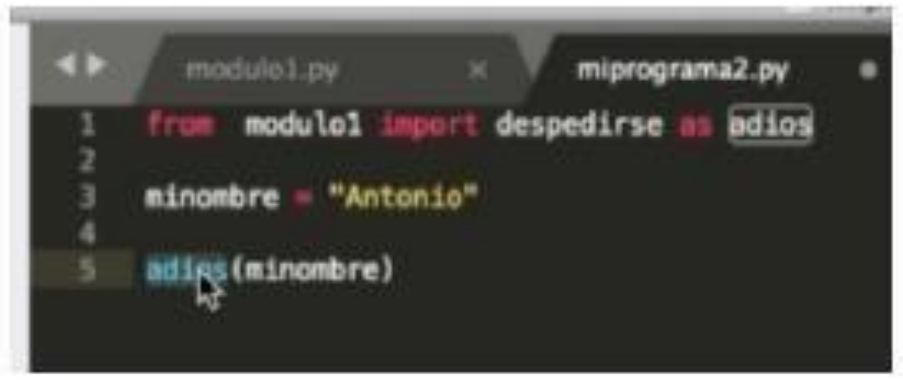
```
# Modulo

def saludar(nombre):
    print("Hola, soy " + nombre)

def despedirse(nombre):
    print("Adios " + nombre)
```

Uso del módulo



```
from  modulo1 import despedirse

minombre = "Antonio"
despedirse(minombre)
```

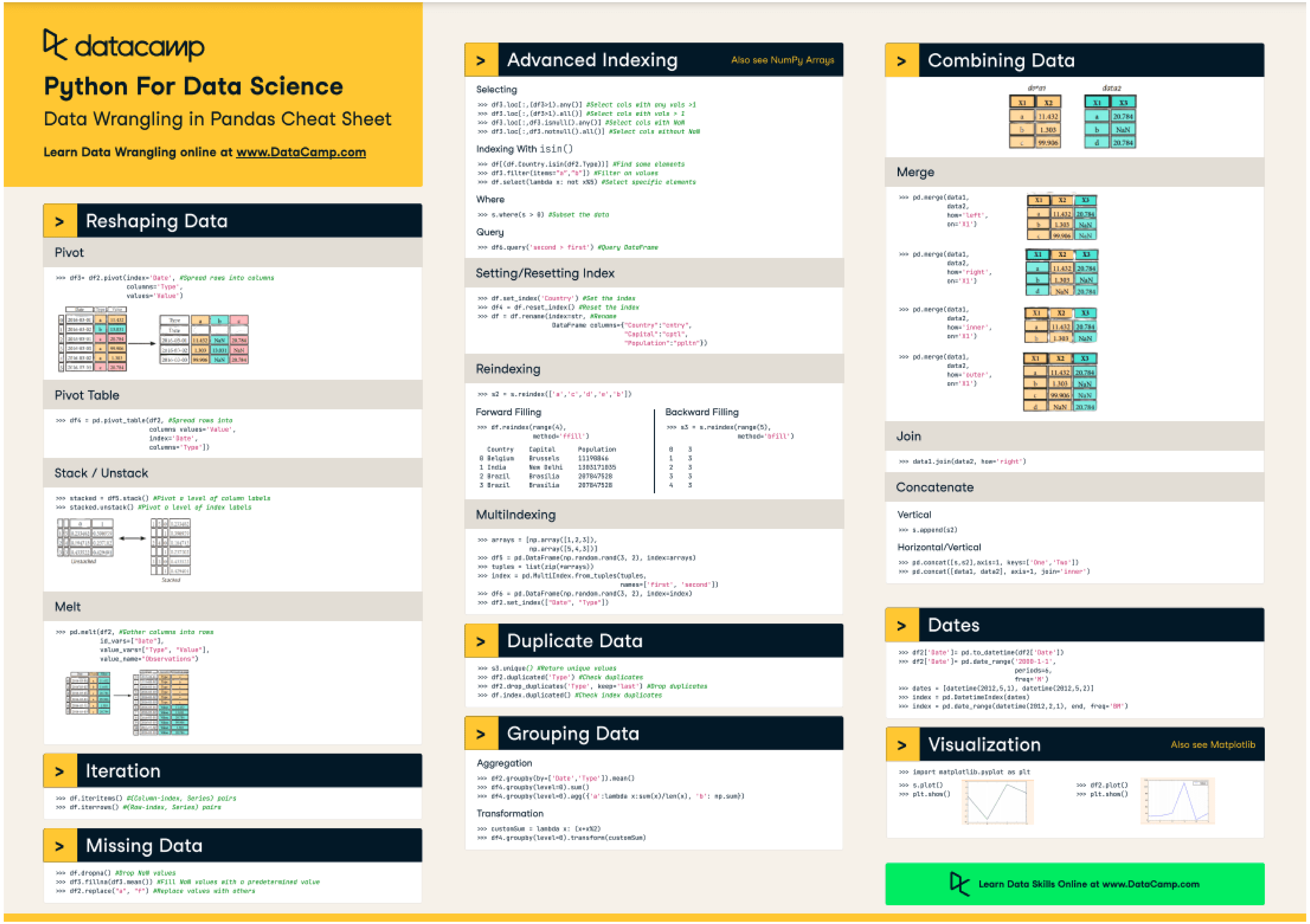
Con alias



```
from  modulo1 import despedirse as adios

minombre = "Antonio"

adios(minombre)
```

Tecnofor

Introduction to Data Manipulation with Pandas



datacamp

Python For Data Science

Data Wrangling in Pandas Cheat Sheet

Learn Data Wrangling online at www.DataCamp.com

> Reshaping Data

Pivot

```
>>> df3= df2.pivot(index='Date', #Spread rows into columns
                   columns='Type',
                   values='Value')
```

Pivot Table

```
>>> df4 = pd.pivot_table(df2, #Spread rows into
                         columns values='Value',
                         index='Date',
                         columns='Type'])
```

Stack / Unstack

```
>>> stacked = df5.stack() #Pivot a level of column labels
>>> stacked.unstack() #Pivot a level of index labels
```

Unstacked

Stacked

Melt

```
>>> pd.melt(df2, #Gather columns into rows
          id_vars=["Date"],
          value_vars=["Type", "Value"],
          value_name="Observations")
```

> Iteration

```
>>> df.iteritems() #(Column-index, Series) pairs
>>> df.iterrows() #(Row-index, Series) pairs
```

> Missing Data

```
>>> df.dropna() #Drop NaN values
>>> df3.fillna(df3.mean()) #Fill NaN values with a predetermined value
>>> df2.replace("a", "f") #Replace values with others
```

> Advanced Indexing

Also see NumPy Arrays

Selecting

```
>>> df3.loc[:,(df3>1).any()] #Select cols with any vals >1
>>> df3.loc[:,(df3>1).all()] #Select cols with vals > 1
>>> df3.loc[:,df3.isnull().any()] #Select cols with NaN
>>> df3.loc[:,df3.notnull().all()] #Select cols without NaN
```

Indexing With isin()

```
>>> df[(df.Country.isin(df2.Type))] #Find same elements
>>> df3.filter(items="a","b"]) #Filter on values
>>> df.select(lambda x: not x%5) #Select specific elements
```

Where

```
>>> s.where(s > 0) #Subset the data
```

Query

```
>>> df6.query('second > first') #Query DataFrame
```

Setting/Resetting Index

```
>>> df.set_index('Country') #Set the index
>>> df4 = df.reset_index() #Reset the index
>>> df = df.rename(index=str, #Rename
                   DataFrame columns={"Country":"cntry",
                                      "Capital":"cptl",
                                      "Population":"ppltn"})
```

Reindexing

```
>>> s2 = s.reindex(['a','c','d','e','b'])
```

Forward Filling

```
>>> df.reindex(range(4),
               method='ffill')
```

```
  Country    Capital     Population
0 Belgium    Brussels    11190846
1 India      New Delhi   1303171035
2 Brazil     Brasília    207847528
3 Brazil     Brasília    207847528
```

Backward Filling

```
>>> s3 = s.reindex(range(5),
                   method='bfill')
```

```
0    3
1    3
2    3
3    3
4    3
```

MultiIndexing

```
>>> arrays = [np.array([1,2,3]),
              np.array([5,4,3])]
>>> df5 = pd.DataFrame(np.random.rand(3, 2), index=arrays)
>>> tuples = list(zip(*arrays))
>>> index = pd.MultiIndex.from_tuples(tuples,
                                      names=['first', 'second'])
>>> df6 = pd.DataFrame(np.random.rand(3, 2), index=index)
>>> df2.set_index(["Date", "Type"])
```

> Duplicate Data

```
>>> s3.unique() #Return unique values
>>> df2.duplicated('Type') #Check duplicates
>>> df2.drop_duplicates('Type', keep='last') #Drop duplicates
>>> df.index.duplicated() #Check index duplicates
```

> Grouping Data

Aggregation

```
>>> df2.groupby(by=['Date','Type']).mean()
>>> df4.groupby(level=0).sum()
>>> df4.groupby(level=0).agg({'a':lambda x:sum(x)/len(x), 'b': np.sum})
```

Transformation

```
>>> customSum = lambda x: (x+x%2)
>>> df4.groupby(level=0).transform(customSum)
```

> Combining Data

data1

X1	X2
a	11.432
b	1.303
c	99.906

data2

X1	X3
a	20.784
b	NaN
d	20.784

Merge

```
>>> pd.merge(data1,
             data2,
             how='left',
             on='X1')
```

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN

```
>>> pd.merge(data1,
             data2,
             how='right',
             on='X1')
```

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
d	NaN	20.784

```
>>> pd.merge(data1,
             data2,
             how='inner',
             on='X1')
```

X1	X2	X3
a	11.432	20.784
b	1.303	NaN

```
>>> pd.merge(data1,
             data2,
             how='outer',
             on='X1')
```

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN
d	NaN	20.784

Join

```
>>> data1.join(data2, how='right')
```

Concatenate

Vertical

```
>>> s.append(s2)
```

Horizontal/Vertical

```
>>> pd.concat([s,s2],axis=1, keys=['One','Two'])
>>> pd.concat([data1, data2], axis=1, join='inner')
```

> Dates

```
>>> df2['Date']= pd.to_datetime(df2['Date'])
>>> df2['Date']= pd.date_range('2000-1-1',
                               periods=6,
                               freq='M')
>>> dates = [datetime(2012,5,1), datetime(2012,5,2)]
>>> index = pd.DatetimeIndex(dates)
>>> index = pd.date_range(datetime(2012,2,1), end, freq='BM')
```

> Visualization

Also see Matplotlib

```
>>> import matplotlib.pyplot as plt
>>> s.plot()
>>> plt.show()
```

```
>>> df2.plot()
>>> plt.show()
```

Learn Data Skills Online at www.DataCamp.com

Tecnofor

by SNGULAR

¡Gracias!

Plaza de la Independencia, 8
28001-Madrid

www.tecnofor.es