Introducción a las cadenas de caracteres

```
>>> cad1 = "Hola"
>>> cad2 = '¿Qué tal?'
>>> cad3 = '''Hola,
que tal?'''
```

Comparación

```
"a">"A"
True

"informatica">"informacion
"
True

"abcde">"abcdef"
False
```

Operadores

+: Concatenación

```
>>> "hola " + "que tal"
'hola que tal'
```

***: Repetición**

```
>>> "abc" * 3
'abcabcabc'
```

Indexación

```
>>> cadena = "josé"
>>> cadena[0]
'j'
>>> cadena[3]
'é'
```

Longitud

```
>>> cadena = "josé"
>>> len(cadena)
4
```

Pasando de pseudocódigo a python3

Pseudocódigo

```
Definir nombre Como Cadena;
Escribir "Dime tu nombre:";
Leer nombre;
```

Python3

```
# nombre = input("Dime tu nombre:")
```

Pseudocódigo

```
Definir numero Como Entero;
Escribir "Dime un número entero:";
Leer numero;
```

Python3

```
numero = int(input("Dime un número entero:"))
```

Pseudocódigo

```
Definir numero Como Real;
Escribir "Dime un número real:";
Leer numero;
```

Python3

```
numero = float(input("Dime un número real:"))
```

Pasando de pseudocódigo a python3

Pseudocódigo

```
Escribir "Hola ",nombre;
```

Python3

```
print("Hola",nombre)
```

Pseudocódigo

```
Escribir Sin Saltar var," ";
```

Python3

```
print(var," ",end="")
```

Pseudocódigo

```
numero <- 7;
```

Python3

```
numero = 7
```

Pseudocódigo

```
trunc(7/2)
```

Python3

```
7 // 2
```

Pasando de pseudocódigo a python3

Pseudocódigo

```
raiz(9)
```

Python3

```
import math
math.sqrt(9)
```

Pseudocódigo

```
subcadena(cadena,0,0)
```

Python3

```
cadena[0]
```

Pseudocódigo

```
cadena3 <- concatenar(cadena1,cadena2)
```

Python3

```
cadena3 = cadena1 + cadena2
```

Pseudocódigo

```
cadena <- Mayusculas(cadena)
```

Python3

```
cadena = cadena.upper()
```

Curso Introducción a la programación con Python3

Estructuras de control

OpenWebinars

Estructura de control: Alternativas

Alternativa simple

```
edad = int(input("Dime tu
edad:"))
if edad>=18:
    print("Eres mayor de edad")
print("Programa terminado")
```

Alternativa doble

```
edad = int(input("Dime tu
edad:"))
if edad>=18:
    print("Eres mayor de edad")
else:
    print("Eres menor de edad")
print("Programa terminado")
```

Estructura de control: Alternativas

Alternativa múliple (pseudocódigo)

```
Proceso notas
    Definir nota como entero;
    Escribir "Dime tu nota:";
    Leer nota;
    Segun nota Hacer
       1,2,3,4: Escribir "Suspenso";
       5: Escribir "Suficiente";
       6,7: Escribir "Bien";
       8: Escribir "Notable";
       9,10: Escribir "Sobresaliente";
       De Otro Modo:
            Escribir "Nota incorrecta";
    FinSegun
    Escribir "Programa terminado";
FinProceso
```

Alternativa múltiple en python3

```
nota = int(input("Dime tu nota:"))
if nota >=1 and nota <= 4:
     print("Suspenso")
elif nota == 5:
     print("Suficiente")
elif nota == 6 or nota == 7:
    print("Bien")
elif nota == 8:
    print("Notable")
elif nota ==9 or nota == 10:
    print("Sobresaliente")
else:
    print("Nota incorrecta")
print("Programa terminado")
```

Estructura de control repetitivas: while

while

```
secreto = "asdasd"
clave = input("Dime la clave:")
while clave != secreto:
    print("Clave incorrecta!!!")
    clave = input("Dime la clave:")
print("Bienvenido!!!")
print("Programa terminado")
```

Instrucciones: continue

```
cont = 0
while cont<10:
     cont = cont + 1
     if cont % 2 != 0:
          continue
     print(cont)
```

Instrucciones: break

```
secreto = "asdasd"
clave = input("Dime la clave:")
while clave != secreto:
    print("Clave incorrecta!!!")
     otra = input("¿Quieres introducir otra clave (S/N)?:")
     if otra.upper()=="N":
     break;
    clave = input("Dime la clave:")
if clave == secreto:
     print("Bienvenido!!!")
print("Programa terminado")
```

¿Y la estructura “Repetir” de pseudocódigo?

Pseudocódigo

```
Proceso login
    Definir secreto, clave como cadena;
    secreto <- "asdasd";
    Repetir
      Escribir "Dime la clave:";
      Leer clave;
      Si clave<>secreto Entonces
           Escribir "Clave incorrecta!!!";
      FinSi
    Hasta Que clave=secreto
    Escribir "Bienvenido!!!";
    Escribir "Programa terminado";
FinProceso
```

Python3

```
secreto = "asdasd"
while True:
     clave = input("Dime la clave:")
     if clave != secreto:
          print("Clave incorrecta!!!")
     if clave == secreto:
          break;
print("Bienvenido!!!")
print("Programa terminado")
```

Estructura de control repetitivas: for

Pseudocódigo

```
Proceso Contar
    Definir var como Entero;
    Para var<-1 Hasta 10 Hacer
       Escribir Sin Saltar var," ";
    FinPara
FinProceso
```

Python3

```
for var in range(1,11):
    print(var," ",end="")
```

Pseudocódigo

```
Proceso ContarDescesdente
    Definir var como Entero;
    Para var<-10 Hasta 1 Con Paso -1 Hacer
       Escribir Sin Saltar var," ";
    FinPara
FinProceso
```

Python3

```
for var in range(10,0,-1):
    print(var," ",end="")
```

Pseudocódigo

```
Proceso ContarPares
    Definir var como Entero;
    Para var<-2 Hasta 10 Con Paso 2 Hacer
       Escribir Sin Saltar var," ";
    FinPara
FinProceso
```

Python3

```
for var in range(2,11,2):
    print(var," ",end="")
```

Usos específico de variables: contadores

```
cont = 0;
for var in range(1,6):
    num = int(input("Dime un número:"))
    if num % 2 == 0:
        cont = cont + 1
print("Has introducido ",cont," números pares.")
```

El contador se **inicializa** a un valor inicial

El contador se **incrementa**
Un contador también se puede **decrementar**

Un contador es una variable entera que la utilizamos para contar cuando ocurre un suceso.

Usos específico de variables: acumuladores

```
suma = 0;
for var in range(1,6):
    num = int(input("Dime un número:"))
    if num % 2 == 0:
        suma = suma + num
print("La suma de los números pares es ",suma)
```

El acumulador se **inicializa** a un valor inicial

El acumulador se **acumula**

Podemos acumular también **productos**

¿A qué valor inicializamos si vamos a acumular productos?

Un acumulador es una variable numérica que permite ir acumulando operaciones. Me permite ir haciendo operaciones parciales.

Usos específico de variables: indicadores

```
indicador  =  False;
for var in range(1,6):
    num = int(input("Dime un número:"))
    if num % 2 == 0:
        indicador  = True
if indicador:
    print("Has introducido algún número par")
else:
    print("No has introducido algún número par")
```

Se **inicializa** a un valor lógico:
No ha sucedido!!!!

Cuando ocurre el suceso **cambiamos** su valor

Al final debemos comprobar **si el suceso ha ocurrido**

Un indicador es una variable lógica, que usamos para recordar o indicar algún suceso.

Curso Introducción a la programación con Python3

Tipos de datos secuencias

OpenWebinars

Tipo de datos cadenas de caracteres

```
>>> cad1 = "Hola"
>>> cad2 = '¿Qué tal?'
>>> cad3 = '''Hola,
que tal?'''
```

Operadores

+: Concatenación
***: Repetición**
Indexación
Longitud
Recorrido

```
>>> cadena = "informática"
>>> for caracter in cadena:
...     print(caracter,end="")
...
informática
```

Operadores de pertenencia

```
>>> "a" in cadena
True
>>> "b" in cadena
False
>>> "a" not in cadena
False
```

Operadores

Slice

```
>>> cadena[2:5]
'for'
>>> cadena[2:7:2]
'frá'
>>> cadena[:5]
'infor'
>>> cadena[5:]
'mática'
>>> cadena[-1:-3]
''
>>> cadena[::-1]
'acitámrofni'
```

Conversión de tipos

```
>>> cad = str(7.8)
>>> type(cad)
<class 'str'>
>>> print(cad)
7.8
```

Las cadenas de caracteres son inmutables

No podemos cambiar los caracteres de una cadena de la siguiente forma:

```
>>> cadena = "informática"
>>> cadena[2]="g"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Esto implica que al usar un método la cadena original no cambia, el método devuelve otra cadena modificada. Veamos un ejemplo:

```
>>> cadena = "informática"
>>> cadena.upper()
'INFORMÁTICA'
>>> cadena
'informática'
```

Si queremos cambiar la cadena debemos modificar su valor con el operador de asignación:

```
>>> cadena = cadena.upper()
>>> cadena
'INFORMÁTICA'
```

Métodos principales de cadenas

Métodos de formato

```
>>> cad = "hola, como estás?"
>>> print(cad.capitalize())
Hola, como estás?

>>> cad = "Hola Mundo"
>>> print(cad.lower())
hola mundo

>>> cad = "hola mundo"
>>> print(cad.upper())
HOLA MUNDO

>>> cad = "Hola Mundo"
>>> print(cad.swapcase())
hOLA mUNDO

>>> cad = "hola mundo"
>>> print(cad.title())
Hola Mundo
```

Métodos de búsqueda

```
>>> cad = "bienvenido a mi aplicación"
>>> cad.count("a")
3

>>> cad.count("a",16)
2
>>> cad.count("a",10,16)
1

>>> cad.find("mi")
13
>>> cad.find("hola")
-1
```

Métodos principales de cadenas

Métodos de validación

```
>>> cad.startswith("b")
True
>>> cad.startswith("m")
False
>>> cad.startswith("m",13)
True
>>> cad.endswith("ción")
True
>>> cad.endswith("ción",0,10)
False
>>> cad.endswith("nido",0,10)
True
```

Otras funciones de validación:
isdigit(), islower(), isupper(), isspace(), istitle(),...

Métodos de sustitución

```
>>> buscar = "nombre apellido"
>>> reemplazar_por = "Juan Pérez"
>>> print ("Estimado Sr. nombre apellido:"replace(buscar, reemplazar_por))
Estimado Sr. Juan Pérez:

>>> cadena = "   www.eugeniabahit.com   "
>>> print(cadena.strip())
www.eugeniabahit.com
>>> cadena="00000001230000000000"
>>> print(cadena.strip("0"))
123
```

Métodos de unión y división

```
>>> hora = "12:23:12"
>>> print(hora.split(":"))
['12', '23', '12']

>>> texto = "Linea 1\nLinea 2\nLinea 3"
>>> print(texto.splitlines())
['Linea 1', 'Linea 2', 'Linea 3']
```

```
>>> lista1 = []
>>> lista2 = ["a",1,True]
```

Tipo de datos secuencia: listas

Operadores

```
lista = [1,2,3,4,5,6]
```

Recorrido

```
>>> for num in lista:
  ...   print(num,end="")
  123456

>>> lista2 = ["a","b","c","d","e"]
>>> for num,letra in zip(lista,lista2):
...   print(num,letra)
1 a
2 b
...
```

Operadores de pertenencia

```
>>> 2 in lista
True
>>> 8 not in lista
True
```

Concatenación (+)

```
>>> lista + [7,8,9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Repeteción (*)

```
>>> lista * 2
[1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]
```

Indexación

```
>>> lista[3]
4

>>> lista1[12]
...
  IndexError: list index out of range

>>> lista[-1]
  6
```

Tipo de datos secuencia: listas

Operadores

Slice

```
>>> lista[2:4]
[3, 4]
>>> lista[1:4:2]
[2, 4]
>>> lista[:5]
[1, 2, 3, 4, 5]
>>> lista[5:]
[6, 1, 2, 3, 4, 5, 6]
>>> lista[::-1]
[6, 5, 4, 3, 2, 1, 6, 5, 4, 3, 2, 1]
```

Listas multidimensionales

```
>>> tabla = [[1,2,3],[4,5,6],[7,8,9]]
>>> tabla[1][1]
5

>>> for fila in tabla:
...  for elem in fila:
...      print(elem,end="")
...   print()
```

Funciones

```
>>> lista1 = [20,40,10,40,50]
>>> len(lista1)
5

>>> max(lista1)
50

>>> min(lista1)
10

>>> sum(lista1)
150

>>> sorted(lista1)
[10, 20, 30, 40, 50]

>>> sorted(lista1,reverse=True)
[50, 40, 30, 20, 10]
```

Las listas son mutables

Los elementos de las listas se pueden modificar:

```
>>> lista1 = [1,2,3]
>>> lista1[2]=4
>>> lista1
[1, 2, 4]
>>> del lista1[2]
>>> lista1
[1, 2]
```

Los métodos de las listas modifican el contenido de la lista:

```
>>> lista1.append(3)
>>> lista1
[1, 2, 3]
```

¿Cómo se copian las listas?

Para copiar una lista en otra no podemos utilizar el operador de asignación:

```
>>> lista1 = [1,2,3]
>>> lista2 = lista1
>>> lista1[1] = 10
>>> lista2
[1, 10, 3]
```

El operador de asignación no crea una nueva lista, sino que nombra con dos nombres distintos a la misma lista, por lo tanto la forma más fácil de copiar una lista en otra es:

```
>>> lista1 = [1,2,3]
>>> lista2=lista1[:]
>>> lista1[1] = 10
>>> lista2
[1, 2, 3]
```

Métodos principales de listas

Métodos de inserción

```
>>> lista = [1,2,3]
>>> lista.append(4)
>>> lista
[1, 2, 3, 4]

>>> lista2 = [5,6]
>>> lista.extend(lista2)
>>> lista
[1, 2, 3, 4, 5, 6]

>>> lista.insert(1,100)
>>> lista
[1, 100, 2, 3, 4, 5, 6]
```

Métodos de eliminación

```
>>> lista.pop()
6
>>> lista
[1, 100, 2, 3, 4, 5]

>>> lista.pop(1)
100
>>> lista
[1, 2, 3, 4, 5]

>>> lista.remove(3)
>>> lista
[1, 2, 4, 5]
```

Métodos principales de listas

Métodos de ordenación

```
>>> lista.reverse()
>>> lista
[5, 4, 2, 1]

>>> lista.sort()
>>> lista
[1, 2, 4, 5]

>>> lista.sort(reverse=True)
>>> lista
[5, 4, 2, 1]

>>> lista=["hola","que","tal","Hola","Que","Tal"]
>>> lista.sort()
>>> lista
['Hola', 'Que', 'Tal', 'hola', 'que', 'tal']
```

Métodos de búsqueda

```
>>> lista.count(5)
1

>>> lista.append(5)
>>> lista
[5, 4, 2, 1, 5]

>>> lista.index(5)
0

>>> lista.index(5,1)
4

>>> lista.index(5,1,4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 5 is not in list
```

Tipo de datos secuencia: tuplas

```
>>> tupla1 = ()
>>> tupla2 = ("a",1,True)
```

Operadores

- **Recorrido**
- **Operadores de pertenencia**
- **Concatenación (+)**
- **Repeteción (*)**
- **Indexación**
- **Slice**

Entre las funciones definidas podemos usar: **`len`, `max`, `min`, `sum`, `sorted`.**

Las tuplas son inmutables

```
>>> tupla = (1,2,3)
>>> tupla[1]=5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Métodos de búsqueda: count, index

```
>>> tupla = (1,2,3,4,1,2,3)
>>> tupla.count(1)
2

>>> tupla.index(2)
1
>>> tupla.index(2,2)
5
```

Curso Introducción a la programación con Python3

Tipos de datos mapas

OpenWebinars

Tipo de datos mapas: diccionarios

```
>>> dicccionario = {'one': 1, 'two': 2, 'three': 3}

>>> dict1 = {}
>>> dict1["one"]=1
>>> dict1["two"]=2
>>> dict1["three"]=3
```

Operadores

Longitud

```
>>> len(a)
  3
```

Indexación

```
>>> a["one"]
  1
>>> a["one"]+=1
>>> a
{'three': 3, 'one': 2, 'two': 2}
```

Eliminación

```
>>> del(a["one"])
>>> a
{'three': 3, 'two': 2}
```

Operadores de pertenencia

```
>>> "two" in a
  True
```

Los diccionarios son mutables

Los elementos de los diccionarios se pueden modificar:

```
>>> a = {'one': 1, 'two': 2, 'three': 3}
>>> a["one"]=2
>>> del(a["three"])
>>> a
{'one': 2, 'two': 2}
```

¿Cómo se copian los diccionarios?

Para copiar un diccionario en otra no podemos utilizar el operador de asignación:

```
>>> a = {'one': 1, 'two': 2, 'three': 3}
>>> b = a
>>> del(a["one"])
>>> b
{'three': 3, 'two': 2}
```

En este caso para copiar diccionarios vamos a usar el método **copy()**:

```
>>> a = dict(one=1, two=2, three=3)
>>> b = a.copy()
>>> a["one"]=1000
>>> b
{'three': 3, 'one': 1, 'two': 2}
```

Métodos principales de diccionarios

Métodos de eliminación

```
>>> dict1 = {'one': 1, 'two': 2, 'three': 3}
>>> dict1.clear()
>>> dict1
{}
```

Métodos de creación

```
>>> dict1 = {'one': 1, 'two': 2, 'three': 3}
>>> dict2 = dict1.copy()

>>> dict1 = {'one': 1, 'two': 2, 'three': 3}
>>> dict2 = {'four':4,'five':5}
>>> dict1.update(dict2)
>>> dict1
{'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5}
```

Métodos principales de diccionarios

Métodos de retorno

```
>>> dict1 = {'one': 1, 'two': 2, 'three': 3}
>>> dict1.get("one")
1
>>> dict1.get("four")
>>> dict1.get("four","no existe")
'no existe'

>>> dict1.pop("one")
1
>>> dict1
{'two': 2, 'three': 3}
>>> dict1.pop("four")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'four'
>>> dict1.pop("four","no existe")
'no existe'
```

Recorridos de diccionarios

```
>>> for clave in dict1.keys():
...  print(clave)
one
two
three

>>> for valor in dict1.values():
...  print(valor)
1
2
3

>>> for clave,valor in dict1.items():
...   print(clave,"->",valor)
one -> 1
two -> 2
three -> 3
```

Curso Introducción a la programación con Python3

Excepciones

OpenWebinars

Excepciones

Errores sintácticos

```
>>> while True print('Hello world')
  File "<stdin>", line 1
    while True print('Hello world')
                   ^
SyntaxError: invalid syntax
```

Errores en tiempo de ejecución (Excepciones)

```
>>> 4/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero

>>> a+4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined

>>> "2"+2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Manejando excepciones

```
>>> while True:
...    try:
...       x = int(input("Introduce un número:"))
...       break
...    except ValueError:
...       print ("Debes introducir un número")
```

```
>>> try:
...    print (10/int(cad))
... except ValueError:
...    print("No se puede convertir a entero")
... except ZeroDivisionError:
...    print("No se puede dividir por cero")
... except:
...    print("Otro error")
```

Manejando excepciones

```
>>> try:
...   print (10/int(cad))
... except ValueError:
...   print("No se puede convertir a entero")
... except ZeroDivisionError:
...   print("No se puede dividir por cero")
... else:
...   print("Otro error")
```

```
>>> try:
...   result = x / y
... except ZeroDivisionError:
...   print("División por cero!")
... else:
...   print("El resultado es", result)
... finally:
...   print("Terminamos el programa")
```

Curso Introducción a la programación con Python3

Introducción a los módulos

OpenWebinars

Introducción a los módulos

Módulo: Cada uno de los ficheros **.py** que nosotros creamos se llama módulo. Los elementos creados en un módulo (funciones, clases, ...) se pueden importar para ser utilizados en otro módulo. El nombre que vamos a utilizar para importar un módulo es el nombre del fichero.

Importación de módulos

```
>>> import math
>>> math.sqrt(9)
3.0
>>> from math import
sqrt
>>> sqrt(9)
3.0
```

Módulos matématicos

- **math**

```
>>> math.sqrt(9)
```

- **functions**
- **random**

```
>>> random.randint(1,10)
```

Módulos de hora y fecha

- **time**

```
>>> time.sleep(1)
```

- **datetime**

Módulos del sistemas

- **os**

```
>>> os.system("clear")
```

Curso Introducción a la programación con Python3

Programación estructurada

OpenWebinars

Programación estructurada

Con la **programación modular y estructura**, un problema complejo debe ser dividido en varios **subproblemas más simples**, y estos a su vez en otros subproblemas más simples. Esto debe hacerse hasta obtener subproblemas lo **suficientemente simples** como para poder ser resueltos fácilmente con algún algoritmo (divide y vencerás). Además el uso de subrutinas nos proporciona la **reutilización del código** y no tener **repetido instrucciones** que realizan la misma tarea.

```
>>> def factorial(n):
...     """Calcula el factorial de un
número"""
...     resultado = 1
...     for i in range(1,n+1):
...         resultado*=i
...     return resultado
```

```
>>>
factorial(5)
120
```

Para **llamar a una función** se debe utilizar su nombre y entre paréntesis los **parámetros reales** que se mandan. La llamada a una función se puede considerar una expresión cuyo valor y **tipo** es el retornado por la función.

Ámbito de variables

Variables locales: se declaran dentro de una función y no se pueden utilizar fuera de esa función

```
>>> def operar(a,b):
...  suma = a + b
...  resta = a - b
...  print(suma,resta)
...
>>> operar(4,5)
9 -1
>>> resta
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'resta' is not defined
```

Variables globales: son visibles en todo el módulo.

```
>>> PI = 3.1415
>>> def area(radio):
...  return PI*radio**2
...
>>> area(2)
12.566
```

Parámetros formales y reales

```
def CalcularMaximo(num1,num2):
    if num1 > num2:
        return num1
    else:
        return num2
```

Parámetros formales: Son las variables que recibe la función, se crean al definir la función.

```
numero1 = int(input("Dime el número1:"))
numero2 = int(input("Dime el número2:"))
num_maximo = CalcularMaximo(numero1,numero2)
print("El máximo es ",num_maximo;)
```

Parámetros reales: Son la expresiones que se utilizan en la llamada de la función, sus valores se “copiarán” en los parámetros formales.

Paso de parámetro por valor o por referencia

En Python el **paso de parámetros es siempre por referencia**. El lenguaje no trabaja con el concepto de variables sino objetos y referencias.

Si se pasa un valor de **un objeto inmutable**, su valor **no se podrá cambiar** dentro de la función.

```
>>> def f(a):
...  a=5
>>> a=1
>>> f(a)
>>> a
1
```

Sin embargo si pasamos **un objeto de un tipo mutable**, **si podremos cambiar** su valor:

```
>>> def f(lista):
...   lista.append(5)
...
>>> l = [1,2]
>>> f(l)
>>> l
[1, 2, 5]
```

Devolución de información
Una función en python puede devolver información utilizando la instrucción **return**. La instrucción **return** puede devolver cualquier tipo de resultados, por lo tanto **es fácil devolver múltiples datos guardados en una lista, tupla o diccionario.**

Aunque podemos cambiar el parámetro real cuando los objetos pasados son de tipo mutables, **no es recomendable hacerlo en Python.**

LLamadas a una función

Cuando se **llama a una función** se tienen que indicar los **parámetros reales** que se van a pasar. **La llamada a una función se puede considerar una expresión cuyo valor y tipo es el retornado por la función.** Si la función no tiene una instrucción **return** el tipo de la llamada será **None**.

```
>>> def cuadrado(n):
...    return n*n

>>> a=cuadrado(2)
>>> cuadrado(3)+1
10
>>> cuadrado(cuadrado(4))
256
>>> type(cuadrado(2))
<class 'int'>
```

Una **función recursiva** es aquella que al ejecutarse hace **llamadas a ella misma**. Por lo tanto tenemos que tener **"un caso base"** que hace terminar el bucle de llamadas. Veamos un ejemplo:

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

Curso Introducción a la programación con Python3

Programación orientada a objetos

OpenWebinars

Introducción a la Programación orientada a Objetos

```
import math
class punto():
""" Representación de un punto en el
plano, los atributos son x e y
que representan los valores de las
coordenadas cartesianas."""

def __init__(self,x=0,y=0):
   self.x=x
   self.y=y

def mostrar(self):
          return str(self.x)+":"+str(self.y)

def distancia(self, otro):
  """ Devuelve la distancia entre ambos puntos. """
   dx = self.x - otro.x
   dy = self.y - otro.y
   return math.sqrt((dx*dx + dy*dy))
```

Llamamos **clase** a la representación abstracta de un concepto. Las clases se componentes de **atributos** y **métodos**.

Los **atributos** definen las características propias del objeto y modifican su estado.

Los **métodos** son bloques de código (o funciones) de una clase que se utilizan para definir el comportamiento de los objetos.

El **constructor** `__init__`, que nos permite inicializar los atributos de objetos. Este método se llama cada vez que se crea una nueva instancia de la clase (objeto).

Todos los métodos de cualquier clase, recibe como primer parámetro a la instancia sobre la que está trabajando. (**self).**

Introducción a la Programación orientada a Objetos

```
>>> punto1=punto()
>>> punto2=punto(4,5)
>>> print(punto1.distancia(punto2))
6.403124237432848 5
```

Para crear un objeto, utilizamos el nombre de la clase enviando como parámetro los valores que va a recibir el constructor.

```
>>> punto2.x
4
>>> punto2.x = 7
>>> punto2.x
7
```

Podemos acceder y modificar los atributos de objeto.

Encapsulamiento

La característica de no acceder o modificar los valores de los atributos directamente y utilizar métodos para ello lo llamamos **encapsulamiento**.

```
>>> class Alumno():
...def __init__(self,nombre=""):
...    self.nombre=nombre
...    self.__secreto="asdasd"
...
>>> a1=Alumno("jose")
>>> a1.__secreto
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Alumno' object has no
attribute '__secreto'
```

Las variables que comienzan por un doble guión bajo __ la podemos considerar como **atributos privados**.

Encapsulamiento

- En Python, las **propiedades (getters)** nos permiten implementar la funcionalidad exponiendo estos métodos como atributos.
- Los métodos **setters** son métodos que nos permiten modificar los atributos a través de un método.

```
class circulo():
    def __init__(self,radio):
      self.radio=radio

    @property
    def radio(self):
      print("Estoy dando el radio")
      return self.__radio

    @radio.setter
    def radio(self,radio):
      if radio>=0:
           self.__radio = radio
      else:
           raise ValueError("Radio positivo")
           self.__radio=0
```

```
>>> c1=circulo(3)
>>> c1.radio
Estoy dando el radio
3
>>> c1.radio=4
>>> c1.radio=-1
Radio debe ser positivo
>>> c1.radio
0
```

Herencia

La clase **punto3d** hereda de la clase punto todos sus propiedades y sus métodos. En la clase hija hemos añadido la propiedad y el setter para el nuevo atributo z, y hemos modificado el constructor (sobreescritura) el método mostrar y el método distancia.

```
class punto3d(punto):
    def __init__(self,x=0,y=0,z=0):
      super().__init__(x,y)
      self.z=z
    @property
    def z(self):
     print("Doy z")
     return self.__z

    @z.setter
    def z(self,z):
     print("Cambio z")
     self.__z=z

    def mostrar(self):
     return super().mostrar()+":"+str(self.__z)

    def distancia(self,otro):
     dx = self.__x - otro.__x
     dy = self.__y - otro.__y
     dz = self.__z - otro.__z
     return (dx*dx + dy*dy + dz*dz)**0.5
```

```
>>> p3d=punto3d(1,2,3)
>>> p3d.x
1
>>> p3d.z
3
>>> p3d.mostrar()
1:2:3
>>> p3d.y = 3
```

La función **super()** me proporciona una referencia a la clase base. Nos permite acceder a los métodos de la clase madre.

Delegación

Llamamos **delegación** a la situación en la que una clase contiene (como atributos) una o más instancias de otra clase, a las que delegará parte de sus funcionalidades.

```
class circulo():
    def __init__(self,centro,radio):
      self.centro=centro
      self.radio=radio

    def mostrar(self):
      return "Centro:{0}-Radio:{1}".format(self.centro.mostrar(),self.radio)
```

```
>>> c1=circulo(punto(2,3),5)
>>> print(c1.mostrar())
Centro:2:3-Radio:5
>>> c1.centro.x = 3
```