

Curso básico de Python

Prof. Andrés Carranza



Requisitos:

- Una PC o laptop.
- Conexión a Internet.
- No requiere conocimientos previos.
- Muchas, muchas ganas de aprender.

Normas de clase:

- Estar puntual a la hora citada.
- Si hay alguna consulta, en cualquier momento puede presionar el botón de “levantar la mano”.
- Se pasará lista al iniciar y terminar la clase.
- Si hay algún inconveniente en estar en clase, escribirme por interno.
- Receso: 15 minutos.
- Clases son Lunes y miércoles de 7pm a 10pm.
- Nota mínima para pasar al siguiente módulo: 11

TEMARIO:

- Introducción a Python.
- Entradas y salidas, Control de flujos y funciones.
- Programación orientada a Objetos, archivos y directorios
- Uso de objetos y módulos.
- Introducción a la programación usando Python.

Introducción:

- ¿Qué es Python?
- Historia de Python.
- Características.
- Requisitos de Hardware.
- Instalación.
- Uso de IDE.
- PEP 8

¿Qué es Python?:

Python es un lenguaje de programación de alto nivel, es decir las sintaxis que se suele usar es fácil de leer para un ser humano, a comparación de otros lenguajes como java y c++, ya que la filosofía del lenguaje es proporcionar una sintaxis muy limpia y que beneficie con código leíble.

```
>>> print("Hello World!")  
Hello World!  
>>>
```


Un poco de historia y curiosidades...

Python fue creado a finales de los ochenta por Guido Van Rossum para las Matemáticas y la Informática en los países bajos, como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba.



Un poco de historia y curiosidades...

El Zen de Python es una colección de 20 principios de software que influyen en el diseño del Lenguaje de Programación Python, de los cuales 19 fueron escritos por Tim Peters en junio de 1999:

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.

Características:

- Lenguaje de alto nivel. Gramática sencilla y legible
- Tipado dinámico y fuerte.
- Orientado a objetos
- Código abierto (Libre y fuente abierta).
- Fácil de aprender.
- Indentado.
- Versátil.
- Incrustable.

Instalación...

Sintaxis básica:

Las funciones se definen con def

#Ejemplo de código en Python

```
import datetime as dt
```

```
def saludo(nombre):
```

```
    """Función de bienvenida al team DSRP"""
```

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

```
    print( "Fecha de registro:" , dt.date.today() )
```

```
    saludos = ["Welcome" , "Bienvenido" , "Bem-vindo"]
```

```
    for saludos in saludos :
```

```
        print(saludo)
```

```
    saludo("Javier")
```

```
    help (saludo)
```

Comentarios de una línea con el numeral (#)

Importación de bibliotecas que necesitamos

No olvidemos los dos puntos que acompañan un: if, def, while, for, etc.

Documentación de la función

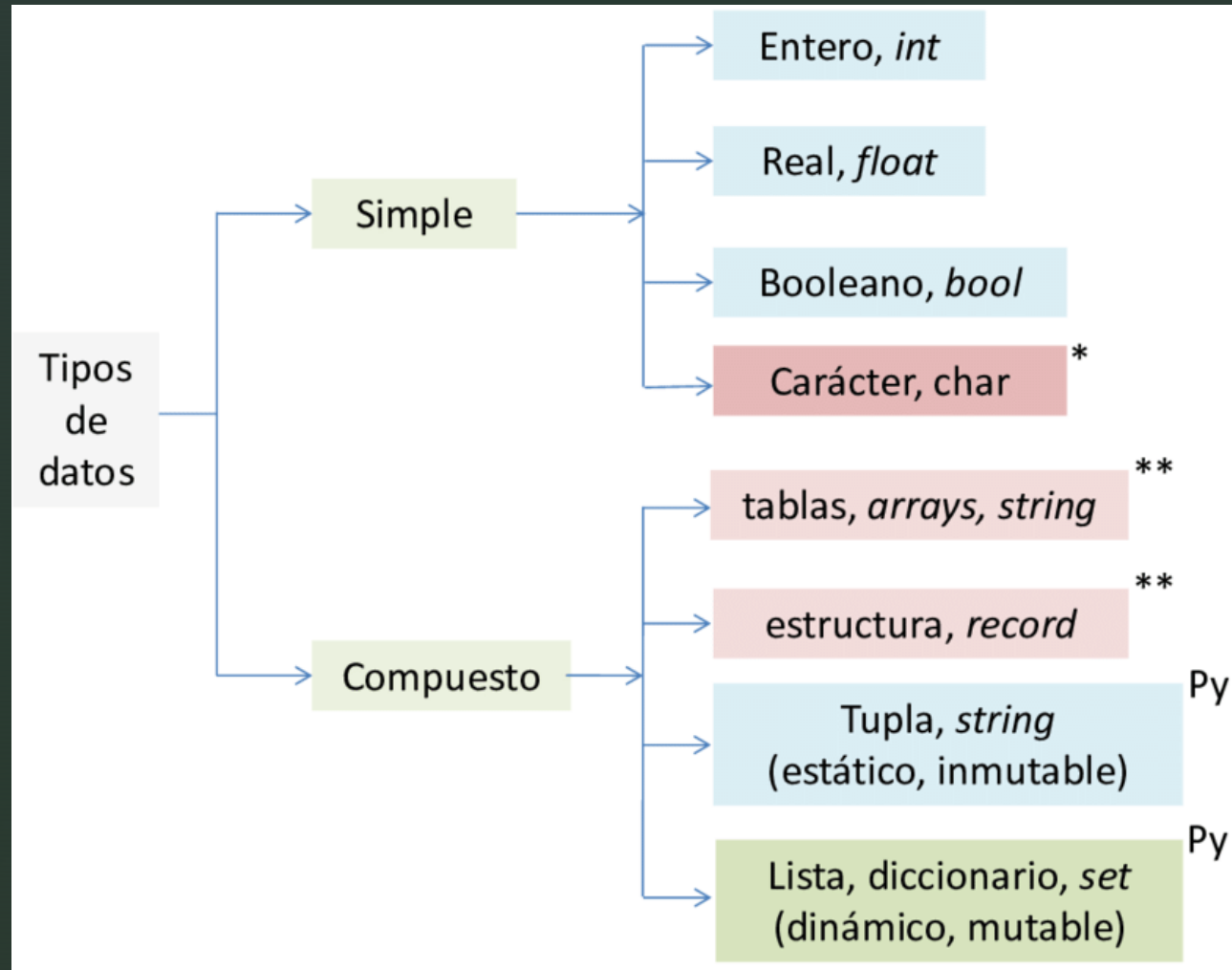
Mostramos nuestras variables en la consola

El bucle for nos indica el recorrido de una variable compuesta (lista, tupla, etc.), también podemos recorrer un rango de elementos usando range().

Llamamos a nuestra función enviando un parámetro de entrada

Pedimos información de la función

Tipos de datos:



Tipos de Operadores:

Operador	Descripción	Ejemplo
+	Suma	<pre>>>> 3 + 2 5</pre>
-	Resta	<pre>>>> 4 - 7 -3</pre>
-	Negación	<pre>>>> -7 -7</pre>
*	Multiplicación	<pre>>>> 2 * 6 12</pre>
**	Exponente	<pre>>>> 2 ** 6 64</pre>
/	División	<pre>>>> 3.5 / 2 1.75</pre>
//	División entera	<pre>>>> 3.5 // 2 1.0</pre>
%	Módulo	<pre>>>> 7 % 2 1</pre>

Tipos de Operadores:

Operador	Descripción	Ejemplo
==	¿son iguales a y b?	<pre>>>> 5 == 3 False</pre>
!=	¿son distintos a y b?	<pre>>>> 5 != 3 True</pre>
<	¿es a menor que b?	<pre>>>> 5 < 3 False</pre>
>	¿es a mayor que b?	<pre>>>> 5 > 3 True</pre>
<=	¿es a menor o igual que b?	<pre>>>> 5 <= 5 True</pre>
>=	¿es a mayor o igual que b?	<pre>>>> 5 >= 3 True</pre>

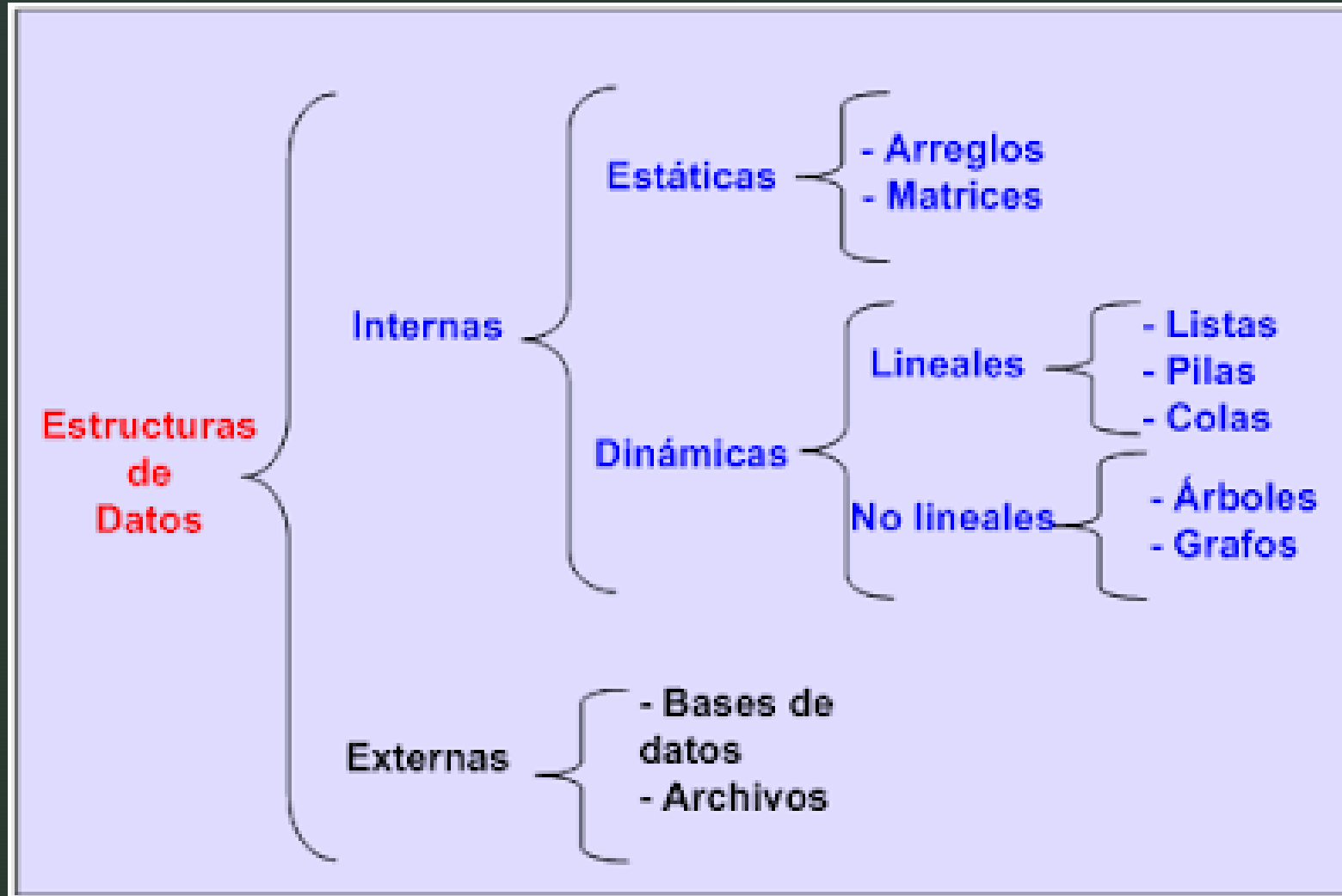
■ Prioridad de las operaciones en Python

P - paréntesis
E - exponentes
M - multiplicación
D - división
A - adición
S - sustracción



Tipos de Operadores:

Operador	Descripción	Ejemplo
=	asigna valor a una variable	<pre>>>> r = 5 >>> r1 = r</pre>
+=	suma el valor a la variable	<pre>>>> r = 5 >>> r += 10; r 15</pre>
-=	resta el valor a la variable	<pre>>>> r = 5 >>> r -= 10; r -5</pre>
*=	multiplica el valor a la variable	<pre>>>> r = 5 >>> r *= 10; r 50</pre>
/=	divide el valor a la variable	<pre>>>> r = 5 >>> r /= 10; r 0</pre>
**=	calcula el exponente del valor de la variable	<pre>>>> r = 5 >>> r **= 10; r 9765625</pre>
//=	calcula la división entera del valor de la variable	<pre>>>> r = 5 >>> r //= 10; r 0</pre>
%=	devuelve el resto de la división del valor de la variable	<pre>>>> r = 5 >>> r %= 10; r 5</pre>

■ Estructura de datos:



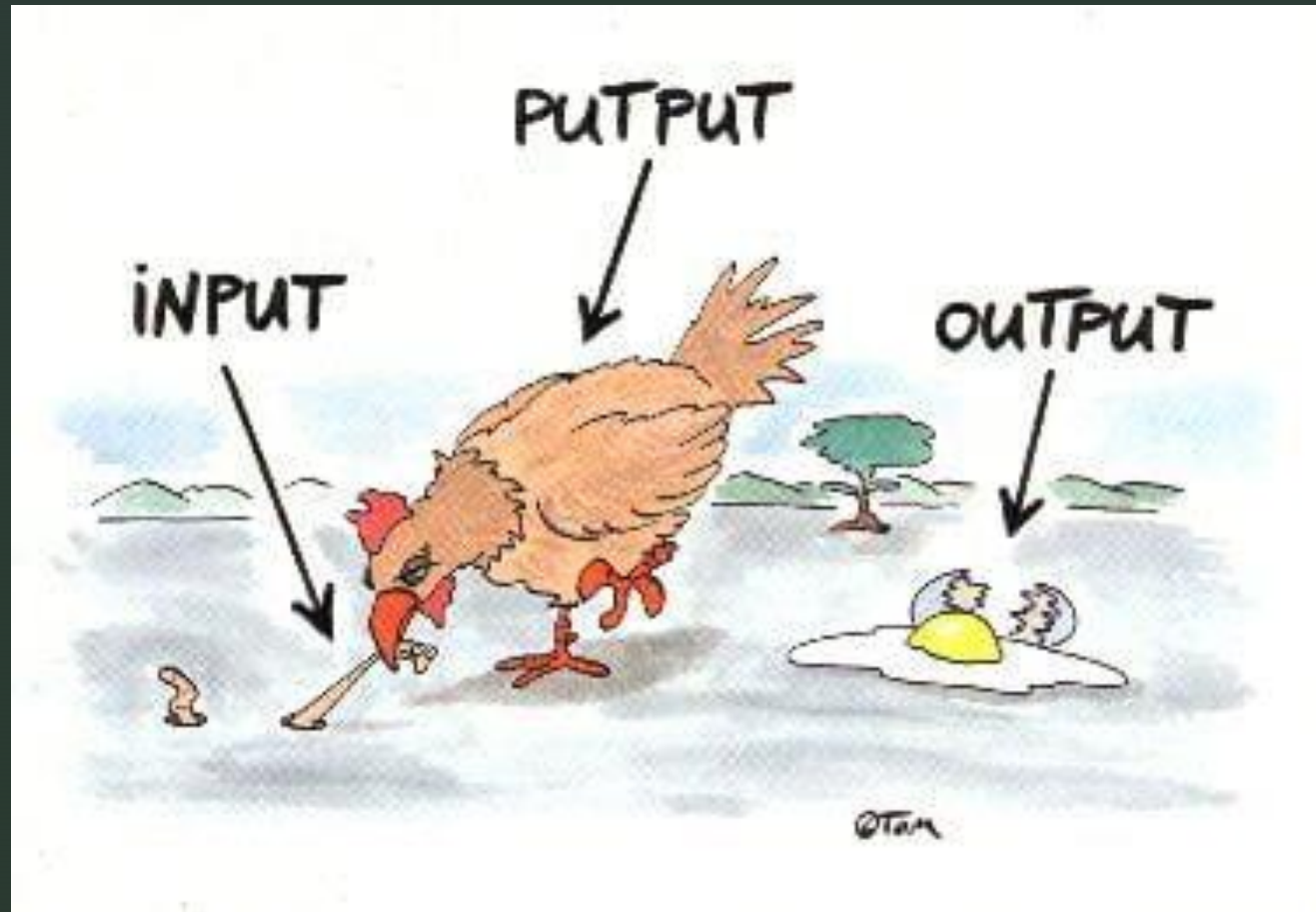
Python 2 vs Python 3:

PYTHON 2.X 	PYTHON 3.X
← LEGACY	FUTURE →
It is still entrenched in the software at certain companies	It will take over Python 2 by the end of 2019
 LIBRARY	LIBRARY 
Many older libraries built for Python 2 are not forwards compatible	Many of today's developers are creating libraries strictly for use with Python 3
 ASCII	UNICODE 
Strings are stored as ASCII by default	Text Strings are Unicode by default
 7/2=3	7/2=3.5 
It rounds your calculation down to the nearest whole number	This expression will result in the expected result
 print "WELCOME TO GEEKSFORGEEKS"	print("WELCOME TO GEEKSFORGEEKS") 
It rounds your calculation down to the nearest whole number	This expression will result in the expected result

■ E/S Control de flujos y funciones:

1. Entradas y salidas (E/S)
2. Operadores de asignación
3. Estructura de control de flujo
4. Manipulación de cadenas
5. Funciones e introducción a la programación funcional

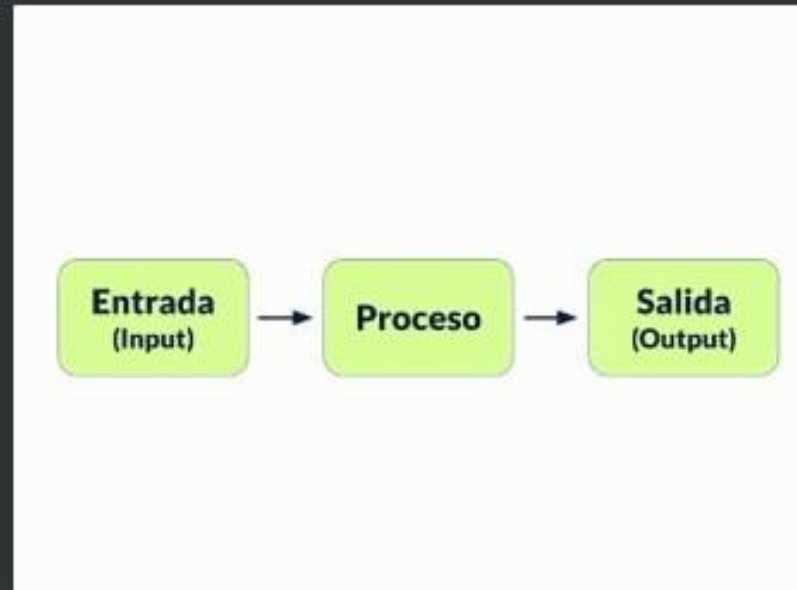
Entradas y Salidas



Entradas y Salidas

Debe existir una entrada y salida de nuestra información en las computadoras.

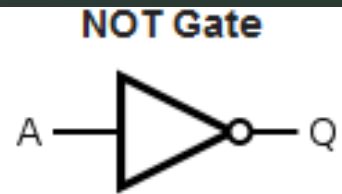
- **Input** : Es el proceso donde colocamos información de la que queremos tener resultado en nuestros cálculos.
- **Proceso**: Es la parte donde se realizan todos los cálculos con nuestra información de input.
- **Output**: Es el resultado que se nos da sobre el proceso de cálculo.



Operadores de asignación

Operador	Ejemplo	Equivalente
=	$x=7$	$x=7$
+=	$x+=2$	$x=x+2 = 7$
-=	$x-=2$	$x=x-2 = 5$
=	$x=2$	$x=x*2 = 14$
/=	$x/=2$	$x=x/2 = 3.5$
%=	$x\%=2$	$x=x\%2 = 1$
//=	$x//=2$	$x=x//2 = 3$
=	$x=2$	$x=x**2 = 49$
&=	$x\&=2$	$x=x\&2 = 2$
=	$x =2$	$x=x 2 = 7$
=^	$x^=2$	$x=x^2 = 5$
>>=	$x>>=2$	$x=x>>2 = 1$
<<=	$x\<\<=2$	$x=x\<\<2 = 28$

Tabla de valores booleanos (Compuertas lógicas):



$$Q = \text{NOT}(A)$$

Truth Table

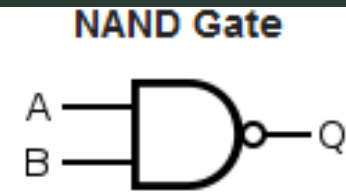
Input A	Output Q
0	1
1	0



$$Q = A \text{ AND } B$$

Truth Table

Input A	Input B	Output Q
0	0	0
0	1	0
1	0	0
1	1	1



$$Q = A \text{ NAND } B$$

Truth Table

Input A	Input B	Output Q
0	0	1
0	1	1
1	0	1
1	1	0

Tabla de valores booleanos (Compuertas lógicas):

AND			OR		
VARIABLES		ESTADO	VARIABLES		ESTADO
SA	SB	Y	SA	SB	Y
False	False	False	False	False	False
False	True	False	False	True	True
True	False	False	True	False	True
True	True	True	True	True	True

XOR			NOT	
VARIABLE		ESTADO	VARIABLE	ESTADO
SA	SB	Y	SA	Y
False	False	False	True	False
False	True	True	False	True
True	False	True		
True	True	False		

Listas en Python:

Las listas en Python son un tipo de dato que permite almacenar datos de cualquier tipo. Son mutables y dinámicas,.

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

```
lista = list("1234")
```

■ Estructuras de control de flujo:

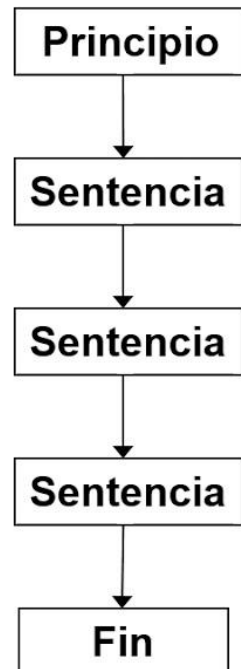
Las estructuras de control, son aquellas que permiten controlar el flujo de ejecución de un programa.

Existen tres principales tipos de estructuras:

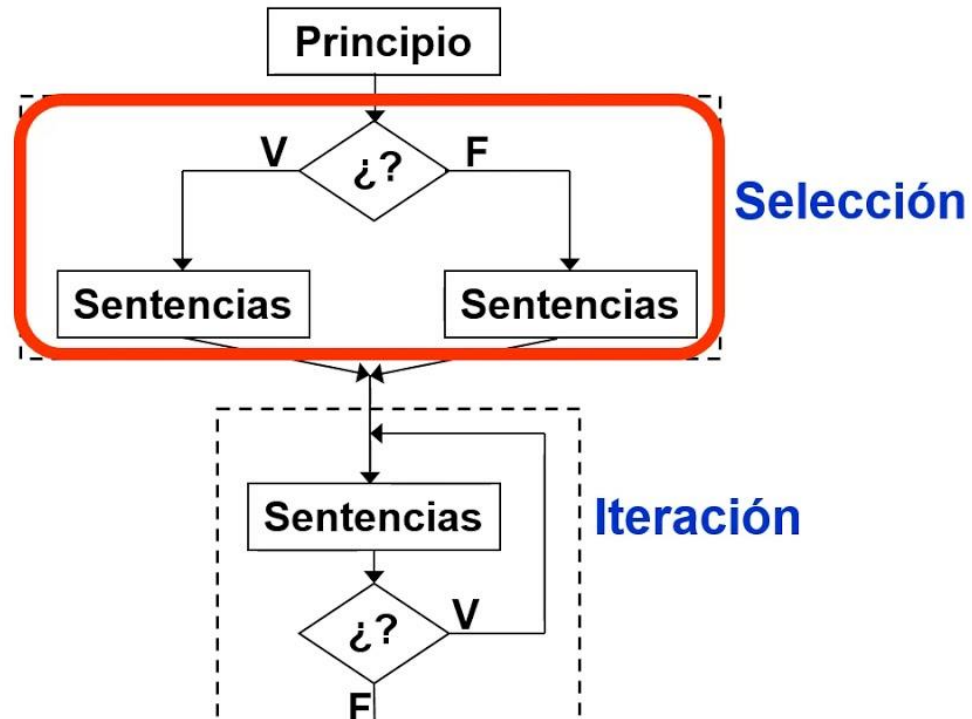
- Secuencial
- No secuenciales:
 - Instrucción condicional
 - Iteración (bucle de instrucciones)

Estructuras de control de flujo:

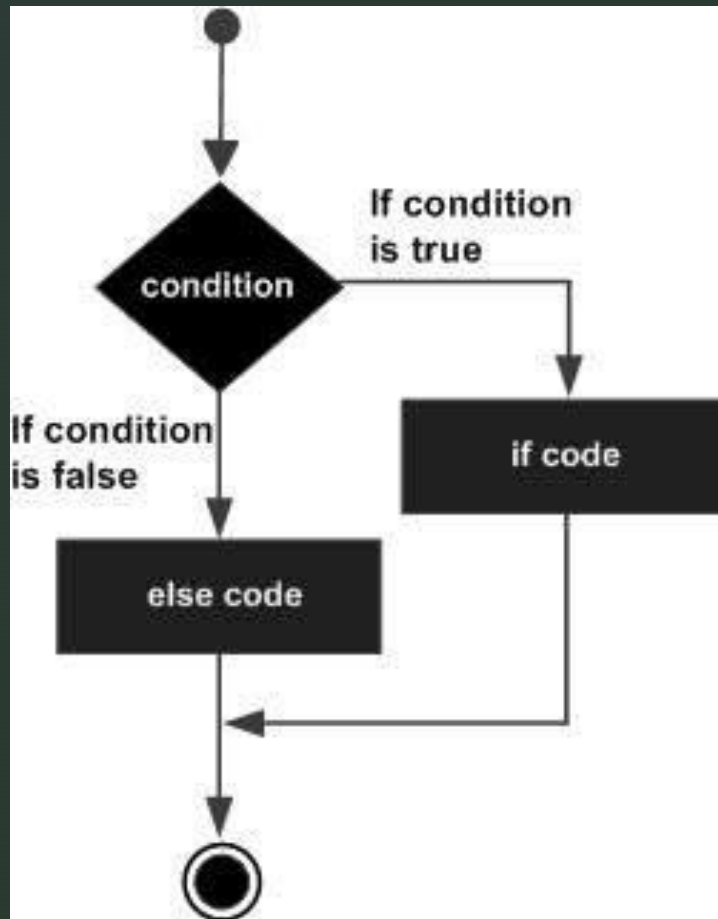
Estructura secuencial



Estructuras no secuenciales



■ Estructuras de control de flujo:

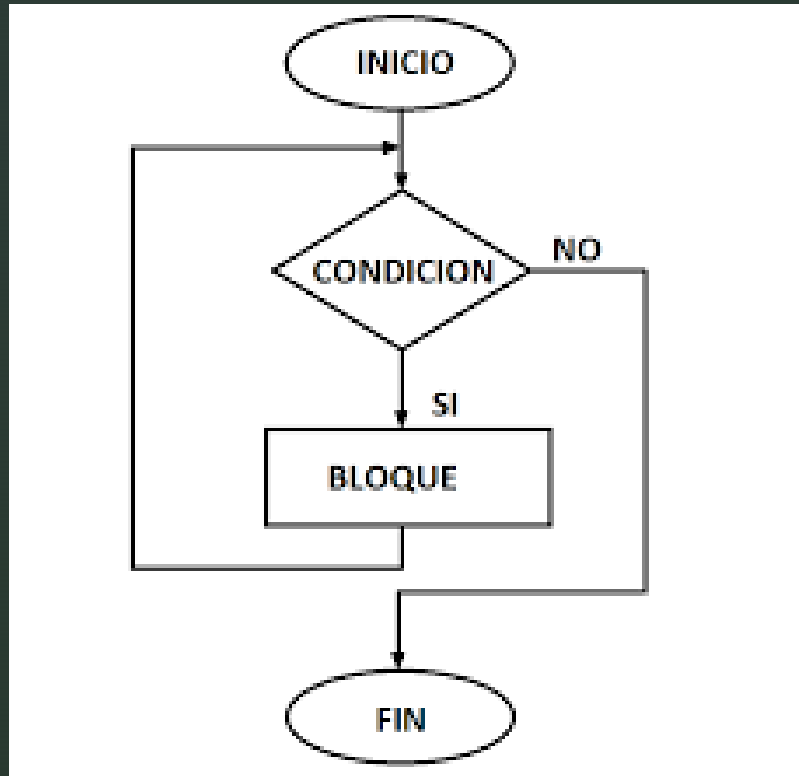


Estructuras de control de flujo: IF

```
num = 8

if num % 2 == 0:
    print('El número {} es
par'.format(num))
else:
    print('El número {} es
impar'.format(num))
```

■ Estructuras de control de flujo:

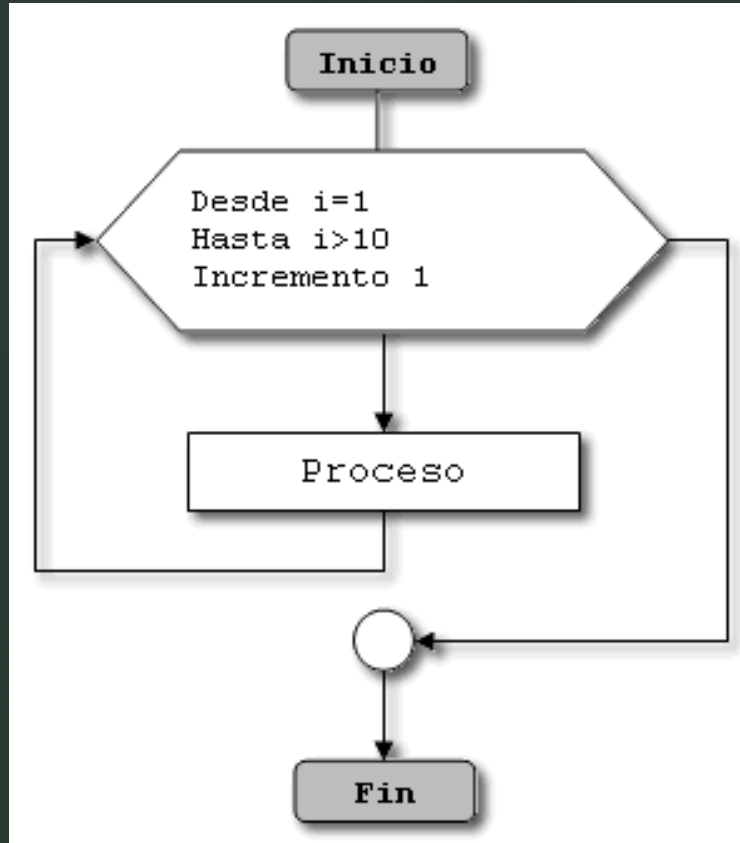


■ Estructuras de control de flujo: WHILE

```
periodo = 2008

while periodo <= 2015:
    print('Informe del año  
{0}'.format(periodo))
    periodo += 1
```

■ Estructuras de control de flujo:



Estructuras de control de flujo: FOR

```
for variable in elemento_recorrible  
(lista, cadena, range,...):  
    cuerpo
```

```
print('Inicio bucle for')  
for i in [0, 1, 2]:  
    print('Paso {}'.format(i))  
  
print('Final del bucle for')
```

Manipulación de cadenas:

```
>>> #CONCATENAR
>>> mensaje1 = 'Hola' + ' ' + 'Mundo'
... print(mensaje1)
... -> Hola Mundo
```

```
>>> #EXTENSIÓN
>>> mensaje4 = 'hola' + ' ' + 'mundo'
... print(len(mensaje4))
... -> 10
```

```
>>> #MULTIPLICAR
>>> mensaje2a = 'Hola ' * 3
... mensaje2b = 'Mundo'
... print(mensaje2a + mensaje2b)
... -> Hola Hola Hola Mundo
```

```
>>> #ENCONTRAR
>>> mensaje5 = "Hola Mundo"
... mensaje5a = mensaje5.find("Mundo")
... print(mensaje5a)
... -> 5
```

```
>>> #AÑADIR
>>> mensaje3 = 'Hola'
... mensaje3 += ' '
... mensaje3 += 'Mundo'
... print(mensaje3)
... -> Hola Mundo
```

```
>>> #MINÚSCULAS
>>> mensaje7 = "HOLA MUNDO"
... mensaje7a = mensaje7.lower()
... print(mensaje7a)
... -> hola mundo
```

Manipulación de cadenas:

```
>>> #REEMPLAZAR
>>> mensaje8 = "HOLA MUNDO"
... mensaje8a = mensaje7.replace("L", "pizza")
... print(mensaje8a)
... -> HOpizzaA MUNDO
```

```
>>> #CORTAR
>>> mensaje9 = "Hola Mundo"
... mensaje9a = mensaje9[1:8]
... print(mensaje9a)
... -> ola Mun
```

Funciones

- Una función agrupa un conjunto de sentencias.
- Puede tener argumentos
- Sintaxis:

```
def nombre_funcion(argumentos):  
    sentencias (código)  
    return retorno
```

Principios:

- Reusabilidad.
- Modularidad.

Funciones

- Para ejecutar una función se la debe invocar:

```
def nombre (parámetros)
```

- Y el valor retornado, puede asignarse a una variable:

```
Resultado = nombre (parámetros)
```

- O imprimirse en pantalla, por ejemplo:

```
print nombre (parámetros)
```

Funciones

- Una función tiene un encabezado y un cuerpo:

```
def nombre (parametros) :
```

Encabezado de la función

```
    sentencia1  
    sentencia2  
    sentencia3
```

Cuerpo de la función



¡CUIDADO CON LA IDENTACIÓN!

Funciones

Argumentos de entrada:

1. Argumentos por posición.
2. Argumentos por nombre.
3. Argumentos por defecto.
4. Argumentos de longitud variable.

Funciones

Sentencia Return:

Permite realizar básicamente dos cosas:

1. Salir de la función y transferir la ejecución de vuelta a donde se realizó la llamada.
2. Devolver uno o varios parámetros, fruto de la ejecución de la función.

```
def mi_funcion():  
    print("Entra en mi_funcion")  
    return  
    print("No llega")  
mi_funcion() # Entra en mi_funcion
```

Funciones

Documentación:

```
def mi_funcion_suma(a, b):  
    """  
    Descripción de la función. Como debe ser usada,  
    que parámetros acepta y que devuelve  
    """  
    return a+b
```

Funciones

Valor y referencia:

En muchos lenguajes de programación existen los conceptos de paso por valor y por referencia que aplican a la hora de como trata una función a los parámetros que se le pasan como entrada. Su comportamiento es el siguiente:

- Si usamos un parámetro pasado por valor, se creará una copia local de la variable, lo que implica que cualquier modificación sobre la misma no tendrá efecto sobre la original.
- Con una variable pasada como referencia, se actuará directamente sobre la variable pasada, por lo que las modificaciones afectarán a la variable original.

Funciones

Valor y referencia:

```
x = 10
def funcion(entrada):
    entrada = 0
funcion(x)

print(x) # 10
```

```
x = [10, 20, 30]
def funcion(entrada):
    entrada.append(40)
funcion(x)
print(x) # [10, 20, 30, 40]
```

Funciones

Problema 1:

Escribir una función a la que se le pase una cadena <nombre> y muestre por pantalla el saludo ; hola <nombre>!.

Problema 2:

Escribir una función que reciba un número entero positivo y devuelva su factorial

Funciones

Problema 3:

Escribir una función que calcule el máximo común divisor de dos números y otra que calcule el mínimo común múltiplo.

Problema 4:

Escribir una función que reciba una muestra de números en una lista y devuelva su media.

Programación orientada a Objetos: POO

- Los problemas suelen tener **varias soluciones** posibles.
- En programación existen **diversas metodologías** que nos ayudan a enfrentar un problema.
- Cada metodología tiene **diversos lenguajes** que las soportan.
 - Algunos lenguajes soportan varias metodologías.

Metodología	Lenguaje
Estructurada	Fortran, C, Pascal, Basic
Orientada a objetos (OOP)	C++ , Java, Smalltalk, Python
Orientada a eventos	VisualBasic

Programación Orientada a Objetos

Definición:

La **Programación Orientada a Objetos (OOP)** es un **método** de programación en el cual los programas se organizan en colecciones cooperativas de **objetos**, cada uno de los cuales representa una **instancia** de alguna **clase**, y cuyas clases son, todas ellas, miembros de una **jerarquía de clases** unidas mediante **relaciones de herencia**.

Comentarios:

- Usamos **objetos en lugar de algoritmos** como bloque fundamental
- Cada objeto es una **instancia** de una clase
- Las clases están relacionadas entre sí por relaciones tan complejas como la **herencia**

■ Ventajas de la POO

- Proximidad de los conceptos modelados respecto a objetos del mundo real
- Facilita la reutilización de código
 - Y por tanto el mantenimiento del mismo
- Se pueden usar conceptos comunes durante las fases de análisis, diseño e implementación
- Disipa las barreras entre el *qué* y el *cómo*

Desventajas de la POO

- Mayor **complejidad** a la hora de entender el flujo de datos
 - Pérdida de linealidad
- Requiere de un lenguaje de modelización de problemas más elaborado:
 - *Unified Modelling Language* (UML)
 - **Representaciones gráficas** más complicadas

Conceptos de la OOP

Conceptos básicos

- Objeto
- Clase

Características de la OOP

- Abstracción:
- Encapsulamiento:
- Modularidad:
- Jerarquía

Otros conceptos OOP

- Tipos
- Persistencia

Tipos de relaciones

- Asociación
- Herencia
- Agregación
- Instanciación

Representaciones gráficas

- Diagramas estáticos (de clases, de objetos...)
- Diagramas dinámicos (de interacción...)

Objeto y Clase

Un **objeto** es algo de lo que hablamos y que podemos manipular

- **Existen** en el mundo real (o en nuestro entendimiento del mismo)

Objeto:Clase

Atributo1=valor
Atributo2=valor
...

Una **clase** describe los objetos del mismo tipo

- Todos los objetos son **instancias** de una clase
- Describe las **propiedades** y el **comportamiento** de un tipo de objetos

Clase

Atributos

Operaciones

Conceptos OOP: Abstracción

- Nos permite trabajar con la **complejidad del mundo real**
 - Resaltando los aspectos relevantes de los objetos de una clase
 - Ocultando los detalles particulares de cada objeto
- Separaremos el **comportamiento** de la **implementación**
- Es más importante **saber qué se hace** en lugar de **cómo se hace**:

Un sensor de temperatura

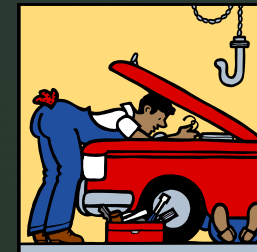
- Se define porque...
 - mide la temperatura
 - nos muestra su valor
 - se puede calibrar...
- No sabemos... (no nos importa)
 - cómo mide la temperatura
 - de qué está hecho
 - cómo se calibra

Conceptos OOP: Abstracción

- La abstracción **no es única**:

Un coche puede ser...

- Una cosa con ruedas, motor, volante y pedales (conductor)
- Algo capaz de transportar personas (taxista)
- Una caja que se mueve (simulador de tráfico)
- Conjunto de piezas (fabricante)



Conceptos OOP: Encapsulamiento

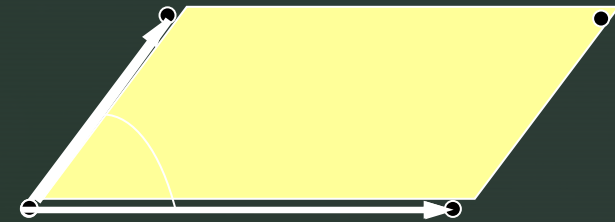
- Ninguna parte de un sistema complejo debe depender de los detalles internos de otra.
- Complementa a la abstracción
- Se consigue:
 - Separando la interfaz de su implementación
 - Ocultando la información interna de un objeto
 - Escondiendo la estructura e implementación de los métodos (algoritmos).
 - Exponiendo solo la forma de interactuar con el objeto

Conceptos OOP: Encapsulamiento

Ejemplo: Un paralelogramo

Vemos que se puede...

- Construir con:
 - 4 puntos (y restricciones)
 - 1 punto y 2 vectores
 - 1 punto, 1 vector, 1 ángulo y 1 lado
- Transformaciones:
 - Escalado
 - Rotación
 - Desplazamiento
- Dibujar



No vemos...

- Como está representado internamente
 - 4 puntos?
 - 1 punto y 2 vectores?
 - ...
- Como se modifica su escala
 - Guardando el factor?
 - Escalando en el momento?
- Idem para rotación, traslación, etc...

■ Conceptos OOP: Modularidad

- Consiste en separar el sistema en bloques poco ligados entre sí: módulos.
 - Organización del código
- Es una especie de encapsulamiento de más alto nivel.
 - El C++ no lo impone aunque lo soporta (namespace)
 - El Java es más formal (packages)
- Difícil pero muy importante en sistemas grandes.
 - Suele aplicarse refinando el sistema en sucesivas iteraciones
 - Cada módulo debe definir una interfaz clara

Conceptos POO: Jerarquía

- Es una **clasificación** u ordenamiento de las abstracciones
- Hay dos jerarquías fundamentales:
 - **Estructura de clases:**
 - Jerarquía “*es un/a*”
 - Relaciones de **herencia**
 - **Estructura de objetos:**
 - Jerarquía “*parte de*”
 - Relaciones de **agregación**
 - Está implementada de manera genérica en la estructura de clases

Conceptos OOP: Tipo

- Es el **reforzamiento** del concepto de clase
- Objetos de tipo diferente no pueden ser intercambiados
- El C++ y el Java son lenguajes fuertemente “tipados”
- Ayuda a corregir errores en tiempo de compilación
 - Mejor que en tiempo de ejecución

Programación Orientada a Objetos en Python

- Tendremos la definición de 3 conceptos:
 - CLASES: Paradigma para organizar un concepto de la vida real.
 - ATRIBUTOS: Características de dicho concepto
 - MÉTODOS: Funcionalidades o acciones que puede realizar un objeto.
 - OBJETO: Es creado a partir de las clases.

Programación Orientada a Objetos en Python

- CLASES: Sintaxis

```
# Creando una clase vacía  
class Perro:  
    pass
```

Creamos un objeto a partir de la clase creada “Perro”:

```
# Creamos un objeto de la clase perro  
mi_perro = Perro()
```

Programación Orientada a Objetos en Python

- ATRIBUTOS:
 - Atributos de instancia: Pertenecen a la instancia de la clase o al objeto. Son atributos particulares de cada instancia, en nuestro caso de cada perro.
 - Atributos de clase: Se trata de atributos que pertenecen a la clase, por lo tanto serán comunes para todos los objetos.

Programación Orientada a Objetos en Python

- ATRIBUTOS: Sintaxis

```
class Perro:
    # El método __init__ es llamado al crear el objeto
    def __init__(self, nombre, raza):
        print(f"Creando perro {nombre}, {raza}")
        # Atributos de instancia

        self.nombre = nombre

        self.raza = raza
```


Programación Orientada a Objetos en Python

- MÉTODOS: Sintaxis

```
class Perro:
    # Atributo de clase
    especie = 'mamífero'

    # El método __init__ es llamado al crear el objeto
    def __init__(self, nombre, raza):
        print(f"Creando perro {nombre}, {raza}")

        # Atributos de instancia
        self.nombre = nombre
        self.raza = raza

    def ladra(self):
        print("Guau")

    def camina(self, pasos):
        print(f"Caminando {pasos} pasos")
```

Programación Orientada a Objetos en Python

- HERENCIA:
- La herencia es un proceso mediante el cual se puede crear una clase hija que hereda de una clase padre, compartiendo sus métodos y atributos. Además de ello, una clase hija puede sobrescribir los métodos o atributos, o incluso definir unos nuevos.

Programación Orientada a Objetos en Python

- HERENCIA: Sintaxis

```
# Definimos una clase padre
class Animal:
    pass

# Creamos una clase hija que hereda de la padre
class Perro(Animal):
    pass
```

```
print(Perro.__bases__)
# (<class ' __main__.Animal'>,)
```

```
print(Animal.__subclasses__())
# [<class ' __main__.Perro'>]
```