# Tema 1: Introducción a Python.

Bloque II: Lenguaje Interpretado

# Bibliografía

- Este material ha sido elaborado con ayuda de material previo de elaboración propia y conjunta con compañeros de la asignatura de Introducción a la Programación en otras titulaciones. Además se han utilizado los siguientes textos:

  - https://docs.python.org/3/reference/index.html.
  - Marzal A, Garcia I, García P. “Introducción a la programación con Python 3”. Publications de la Universitat Jaume I.
  - http://docs.linuxtone.org/ebooks/Python/Thinking_In_Python.pdf
  - https://automatetheboringstuff.com/

# Contenidos

- Introducción
  - Compiladores e Interpretes
  - Python
  - Consola de Python
  - Entornos virtuales
- Lenguaje Python
  - Operadores

# Compiladores e Intérpretes

- Para traducir un lenguaje de alto nivel a código máquina existen dos maneras diferentes:
  - Compilador:
    - Lee completamente un programa y lo traduce en su integridad.
    - Una vez disponible la traducción a código máquina se puede ejecutar cuantas veces se desee sin necesidad de volver a traducir el programa original.
  - Intérprete:
    - Lee un programa instrucción a instrucción, y para cada una de ellas, las traduce a las instrucciones de código maquina equivalentes, ejecutandolas inmediatamente.

# Compiladores e Intérpretes

- Para traducir un lenguaje de alto nivel a código máquina existen dos maneras diferentes:
  - Compilador:
    - …
  - Intérprete:
    - Lee un programa instrucción a instrucción, y para cada una de ellas, las traduce a las instrucciones de código maquina equivalentes, ejecutándolas inmediatamente.
    - Este proceso se repite para cada ejecución
    - No hay un proceso separado de traducción, ejecución.

# Python

- Lenguaje interpretado centrado en código legible.
- Creado en 1991 por Guido Van Rossum
- Multiplataforma y Multiparadigma:
  - Orientación a objetos, imperativo y funcional.
- Dos ramas activas, 3.x y 2.7.x
  - Cambio notable al pasar a la versión 3.0
- Expresivo: programas más corto.

# Python

- Muy adecuado para matemáticos y científicos.
  - Módulos dedicados a computo matemático e ingenieril.
  - Sintaxis similar a Matlab.
- Adecuado para docentes.
  - Fácil de aprender, adecuado para iniciarse en la programación.
- Gratis, multiplataforma.
- En constante evolución.
- Muy Extendido.
  - Google, Youtube, Intel, Nasa, JPMorgan, …

# Python

- Tipado:
  - Fuerte/Débil:
    - Toda variable tiene un tipo.
    - Operar con variables de distinto tipo hay que realizar una conversión.
    - Se permiten las conversiones entre tipos numéricos, son implícitas.
  - Estático/Dinámico:
    - Una misma variable puede tomar valores de distinto tipo.
    - El tipo de una variable se determina en tiempo de ejecución.

# Python

- La consola
  - El código python puede ser ejecutado desde una consola. Para ello hay que abrir el interprete de python.

- IDE y entornos virtuales:
  - Como IDE utilizaremos PyCharm. La versión Comunity es gratuita, para estudiantes tenéis disponible la versión PRO.
  - Es aconsejable no usar la instalación del sistema de Python sino definir entornos virtuales.

- Python Notebook:
  - Servicio web que exporta una consola de python.
  - Google Colab: https://colab.research.google.com/

# Python

- Mostrar resultados
  - Al usarlo en modo interprete, se imprimirá el valor de cada instrucción, pero solo de la última.
  - Al utilizar un Notebook se imprimirá sólo la última instrucción de la celda.
  - Para controlar lo que imprimimos y poder imprimir cuando se interprete un script, por ejemplo a traves de un IDE, usaremos la función print().

```
print("Hola Mundo!")
```

# Python

- Operadores aritmeticos:
  - Precedencia izquierda-derecha

| Operador | Tipo | Ejemplo | Resultado | Descripción | Precedencia |
|---|---|---|---|---|---|
| + | unario | +3 | 3 | Operador identidad | 2 |
| - | unario | -3 | -3 | Cambio de signo | 2 |
| + | binario | 2 + 3 | 5 | Suma | 4 |
| - | binario | 2 - 3 | -1 | Resta | 4 |
| * | binario | 2 * 3 | 6 | Multiplicación | 3 |
| / | binario | 2 / 3 | 0.6… | División Real | 3 |
| // | binario | 2 // 3 | 0 | División Entera | 3 |
| % | binario | 2 % 3 | 2 | Resto | 3 |
| ** | binario | 2 ** 3 | 8 | Potencia | 1 |

# Python

- Tipos numéricos
  - En Python existen 3 tipos numéricos, int, float y complex.
  - Por defecto, si no se escribe el punto decimal, se considera que el literal es un entero. 42 es int, 42.0 es float.
  - Se puede convertir entre tipos con sus constructores int(), float() y complex().
  - Existe conversión implícita entre tipos.

    3 * 2 + 1.0 -> float
  - Para averiguar el tipo de una variable se puede usar la función type().

# Python

- Otras operaciones con números
  - abs(x): devuelve el valor absoluto de x.

    abs(-3) -> 3

  - divmod(x, y): devuelve a la vez el resultado de la división y el resto.

    divmod(7, 3) -> (2, 1)

  - pow(x, y): similar a x ** y.

  - Si c es una variable de tipo complejo c.conjugate() devuelve el conjugado.

# Python

- Operadores de comparación:
  - Menor prioridad que las operaciones aritméticas

| Operador | Ejemplo | Resultado | Descripción |
|---|---|---|---|
| < | 3 < 4 | True (1) | Menor que |
| <= | 3 < 4 | True | Menor igual que |
| > | 3 > 3 | False | Mayor que |
| >= | 3 > 3 | True | Mayor igual que |
| == | 3 == 4 | False | Igual que |
| != | 3 != 4 | True | Distinto que |
| is | a is b | | Identidad obj |
| is not | a is not b | | Distinto obj |

  - Se pueden encadenar “a < b <= c”.

# Python

- Operadores booleanos:
  - None, False, 0, 0.0 y 0j evalúan como False

| Operador | Tipo | Ejemplo | Resultado | Precedencia |
|---|---|---|---|---|
| or | binario | True or False | True | 3 |
| and | binario | False and True | False | 2 |
| not | unario | not True | False | 1 |

  - Evaluación perezosa, se evita leer el segundo argumento si no es necesario.

  - Los operadores and y or devuelven uno de sus operadores.

# Python

- Sentencia condicional if:

```
if condicion:
   instrucción 1
   instrucción 2
   …
   instrucción N
```

  - No hay llaves ni palabras clave que delimiten el ámbito de lo que está dentro del if, lo delimita la indentación.

```
if condicion1:
   …
elif condicion 2:
   …
else:
   …
```

# Python

- Bucle for:

```
for variable in coleccion:
    instrucción 1
```

  - A diferencia de los for de otros lenguajes, en Python el for no itera hasta que deja de cumplirse una condicion.
  - En Python un for itera por cada uno de los elementos de una colección.

```
for name in [‘pepe’, ‘paco’, ‘juan’]:
    print(name)
```

  - Para iterar N veces, necesitamos una lista con N elementos. Podemos usar range()

```
for in in range(10):
    print(i * 2)
```

# Python

- Ejercicio Pirámide:
  - Dada una altura, pintar una pirámide como la siguiente:

```
    *
   ***
  *****
```

# Python

- String:
  - En Python, los string se representan encerrados entre comillas dobles o simples:
    - “Hola Mundo!”
    - ‘Esto es una frase’
  - Se pueden concatenar con la operación +:
    - “Hola ” + ‘Mundo’.
  - Para mostrar alguna de las comillas, se usa la contraria:
    - ‘Esta palabra está “entrecomillada”.’
  - La multiplicación también está definida:
    - ‘Hola’ * 3 == “HolaHolaHola”

# Python

- Ejercicio Pirámide:
  - Dada una altura, pintar una pirámide como la siguiente:

```
  *
 ***
*****
```

# Python

- Ejercicio Pirámide:
  - Dada una altura, pintar una pirámide como la siguiente:

```
  *
 ***
*****
```

  - ¿Cómo pintarías una pirámide invertida?

# Python

- Range:
  - Utilizado principalmente en los bucles for para gerenara rangos:
  - En realidad lo que devuelve es un objeto de tipo range, que es un “enumerador” o “generador”.
  - En Python 2.7.x se recomienda el uso de xrange().
  - La función tiene 3 parametros:
    - range(10) -> 0..9
    - range(-5, 5) -> -5..4
    - range(0, 100, 2) -> 0, 2,..., 98

# Python

- Ejercicio Pirámide:
  - Dada una altura, pintar una pirámide como la siguiente:

```
  *
 ***
*****
```

  - ¿Cómo pintarías una pirámide invertida?

# Python

- Ejercicio Pirámide:
  - Dada una altura, pintar una pirámide como la siguiente:

```
    *
   ***
  *****
```

  - ¿Cómo pintarías una pirámide invertida?
  - Pinta un rombo.

# Python

- Bucle for:

```
for variable in coleccion:
    instrucción 1
    instrucción 2
```

- Bucle while:

```
while condición:
    instrucción 1
    instrucción 2
```

  - **Se puede salir de forma prematura de un bucle con “break”**
  - **Se puede forzar la iteración del bucle sin llegar al final con “continue”**

# Python

- Bucle for:

```
for variable in coleccion:
  instrucción 1
  instrucción 2
else:
  instrucción 3
```

- Bucle while:

```
while condición:
  instrucción 1
  instrucción 2
else:
  instrucción 3
```

# Python

- Ejemplo else en bucles:

```
for i in range(3):
  password = input(‘Enter password: ’)
  if password == ‘contraseña’:
    print(“Correcto!”)
    break
else:
  print(‘Has fallado 3 intentos.’)
```

# Python

- Funciones:

```
def mifuncion(parametro):
    instrucción 1
    instrucción 2
```

  - En Python las funciones no especifican el valor de retorno.

  - Por ser un lenguaje dinámico no se especifica el tipo de los parámetros.

```
def imprime(parametro):
    print(parametro)
```

  - El tipo estará definido en la llamada, seguirá siento tipado fuerte.

```
def concatena(parametro):
    print(“Hola ” + parametro)
```

# Python

- Modulos:
  - En Python, como en otros lenguajes, su funcionalidad puede ser completada con otras bibliotecas.
  - Una de estas bibliotecas es math, que nos añade ciertas funciones matemáticas, además de las constantes pi y e.

    ```
    math.pi
    math.sin(x)
    ```

  - Para poder hacer uso de una biblioteca necesitamos importarla.

    ```
    import math
    ```

  - Si sólo queremos importar cierta funcionalidad

    ```
    from math import e, ceil
    ```

# Python

- Ejercicios:
  - Crear una función que dada una altura, pinte un rombo.
  - Crear una función que devuelva el área de un rectángulo.
  - Crear una función que devuelva el perímetro de una circunferencia (utilizando math).
  - Crear una función que resuelve una ecuación de segundo grado recibiendo a, b, c.
  - Crear una función que calcule el factorial de n.

# Python

- Funciones 2:
  - Las funciones pueden tener argumentos con valores por defecto.

```
def area(base = 3, altura = 4):
    return (base * altura)
```

  - Los argumentos por defecto han de ir al final.

  - Se puede asignar valores a parámetros por nombre.

```
print(area())
print(area(5))
print(area(altura = 7))
```

# Python

- Ejercicios:
  - Modificar el ejercicio que calcula una ecuación de segundo grado para que utilice parámetros por defecto = 0.
  - ¿De cuantas formas puede llamar a la función con a= 3, b=0 y c= 7?
  - ¿Cuál es la solución a la ecuación de segundo grado con a = 1, b = 0 y c = -1?

# Python

- Tuplas:
  - Son colecciones heterogéneas, ordenadas e inmutables, una agrupación de elementos.

```
t = (‘Hola’, 4)
print(t)
```

  - Se puede acceder a sus elementos.

```
print(t[0])
```

  - Se pueden desempaquetar.

```
text, number = t
print(number)
```

  - Para saber el número de elementos len().

  - Como toda colección, se puede iterar en un bucle for.

```
for e in t:
 print(e)
```

# Python

- Ejercicios:
  - Modificar el ejercicio que calcula una ecuación de segundo grado para que utilice parámetros por defecto = 0.
  - ¿De cuantas formas puede llamar a la función con a= 3, b=0 y c= 7?
  - ¿Cuál es la solución a la ecuación de segundo grado con a = 1, b = 0 y c = -1?
  - Modificar el ejercicio de la ecuación de segundo grado para que devuelva todos los resultados.
  - Crear un programa que, haciendo uso de la función anterior, muestre la/s solución/es de una ecuación de segundo grado.

# Python

- Listas:
  - Son colecciones heterogéneas, ordenadas.

    ```
    l = [‘Hola’, 4]
    print(l)
    ```
  - Se puede acceder a sus elementos.

    ```
    print(l[0])
    ```
  - Se pueden añadir elementos.

    ```
    l.append(3e-2)
    ```
  - Para saber el número de elementos len().
  - Como toda colección, se puede iterar en un bucle for.

    ```
    for e in l:
     print(e)
    ```

# Python

- Ejercicios:
  - Crear una función que devuelva una lista con los números primos de 0 a 100.
  - Dada una lista, imprimir los elementos en posición par.
  - Un número es perfecto si la suma de sus divisores es igual a si mismo, ejemplo el 28. Crear una función que devuelva si un número es perfecto.
  - Crear una función que recibe una lista de números y devuelve una lista de tuplas por cada elemento. Cada tupla tendrá el elemento, su cuadrado y su cubo:

    [1, 2, 3] -> [(1, 1, 1), (2, 4, 8), (3, 9, 27)]

# Python

- String 2:
  - Conversiones de cadenas a números y viceversa:
    - int(), float()
    - str()
  - Pasar a minusculas, mayusculas o capitalizar:
    - “Hola”.lower()
    - ‘Hola’.upper()
    - ‘Hola que tal’.title()
  - Docstring: String literal con saltos de linea:
    - ‘’’Esta cadena puede
      tener saltos de linea.‘’’

# Python

- Funciones 3:
  - Debido a la falta de tipado en los parametros de una función, se aconseja (en esta asignatura se obliga) documentar la función.

```
def area(base = 3, altura = 4):
  “””Calcula el are de un rectángulo.
  Args:
    base (Number): base del rectángulo (3).
    altura (Number): altura del rectángulo (4).
  return:
    Number: area del rectángulo”””
  return (base * altura)
```

  - Los argumentos pueden estar en una lista:

```
rectangle = [7, 8]
area(*rectangle)
```

# Python

- Slice:
  - Se puede acceder a un elemento de un contenedor:
    **a = (1, ‘Hola’, 3.5) # a[1] = ‘Hola’**
  - Si se intenta acceder fuera de rango Python fallará.
  - Si se usa un indice negativo, se accede por el final:
    **a[-1] # 3.5**
  - Utilizando el operador “:” se puede acceder a rangos de valores:
    **“Hola”[0:2] # ‘Ho’**
    **“Hola”[1:3] # ‘ol’**
    **“Hola”[1:] # ‘ola’**
    **“Hola”[:] # ‘Hola’**
    **“Hola”[::2] # ‘Hl’**
  - Si el contenedor es mutable, se puede reescribir el subrango:
    **[0, 1, 2, 3][::2] = [4, 6] # [4, 1, 6, 3]**

# Python

- Ejercicios:
  - Crear una función que devuelva el máximo y mínimo de 2 números.
  - Crear un función que devuelva el máximo y mínimo de una lista de números.
  - Crea una función que recibe una cadena y la devuelve en sentido inverso.
  - Crea una función que recibe una cadena y devuelve la misma cadena con sólo la primera letra mayúscula.
  - Crea una función que imprima por pantalla una matriz (lista de listas).

# Python

- Listas 2:
  - Cuidado al asignar una lista a otra:

```
l1 = [1, 2 ,3]
l2 = l1
l1[0] = 7
print(l2)
```

  - Para copiar listas utilizar el operador copia.

```
l2 = l1[:]
```

  - Se puede extraer un elemento de la lista (se elimina).

```
n = l1.pop(1) # extrae el 2 de la lista
l1.pop() # extrae el último elemento de la lista
```

  - Se puede insertar un elemento en una posición concreta:

```
l1.insert(0, 42) # añade 42 al principio de la lista
```

# Python

- Ejercicios:
  - Implementar las funciones:

    **`apilar(pila, elemento)`**

    **`desapilar(pila)`**

    **`cima(pila)`**

    Que simulan una pila usando una lista.
  - Implementar las funciones:

    **`encolar(cola, elemento)`**

    **`desencolar(cola)`**

    **`primero(cola)`**

    Que simulan una cola usando una lista.
  - Implementa un programa que devuelva si en una cadena que recibe de entrada, los paréntesis que en ella aparecen están balanceados.
  - Practica 1: Evaluación de expresiones aritméticas sencillas.

# Python

- Set:
  - Colección desordenada y mutable de objetos hasheables.
  - Para crear un set, elementos entre llaves ({}).

    ```
    s = {“A”, “B”, 3}
    ```
  - También existe el constructor set().

    ```
    s1 = set()  # set vacio
    s2 = set(“A”, “B”, 3)
    ```
  - Como el resto de colecciones, soporta:

    ```
    len(s1)  # longitud
    x in s2  # consulta
    for e in s:  # iterar
    ```
  - Para modificar el conjunto:

    ```
    s.add(“C”)  # añade “C” al conjunto
    s.pop()  # devuelve uno de sus elementos y lo elmina
    s.remove(“B”)  # elimina el elemento “B”
    ```

# Python

- Ejercicios:
  - Reimplementar la función del calculo de los 100 primeros primos haciendo uso de conjuntos.
  - Crear una función que recibe un número y devuelve una lista con sus dígitos.
  - Un número es cubifinito si al elevar al cubo sus dígitos y sumarlos da como resultado 1 u otro número cubifinito. Crear una función que reciba un número y devuelva si es cubifinito.
  - Implementar las funciones sobre conjunto, unión, intersección, diferencia y copia.

# Python

- Set 2:
  - Los set, al ser mutables, no son hasheables.
  - Los frozenset son set inmutables.

    ```
    s = frozenset(1, 2, 4)
    ```
  - Operadores de comparación en set, isdisjoint(), <, >.

    ```
    s1.issubset(s2) # s1 <= s2
    s2.issuperset(s1)  # s2 >= s1
    ```
  - Constructoras no generadoras:

    ```
    s3 = s1.union(s2)          # s3 = s1 | s2
    s3 = s1.intersection(s2)   # s3 = s1 & s2
    s3 = s1.difference(s2)     # s3 = s1 - s2
    ```
  - Copiar un conjunto:

    ```
    s2 = s1.copy()
    ```

# Python

- Ejercicios:
  - En una lista hay números en parejas, positivos y negativos (si está el 3, está el -3) pero en posiciones desconocidas. Todos los números tienen su pareja de signo opuesto excepto uno. Crea una función que dada una lista de este tipo, devuelva el elemento desparejado.
  - En una lista de números, todos están repetidos excepto uno. Crea una función que devuelva el número no repetido.
  - Dado un conjunto de números, devolver una tupla con 2 conjuntos, el primero contendrá los pares y el segundo los impares.
  - Crear una función que devuelva el conjunto potencia.

# Python

- Diccionarios:
  - Colección de parejas clave valor, la clave ha de ser hashable, el valor puede ser arbitrario.
  - Para crear un diccionario, parejas de elementos entre llaves ({}).

    ```
    d = {“one” : 1, “two” : 2, “three” : 3}
    ```
  - También existe el constructor dict().

    ```
    d1 = dict()  # igual que s1 = {}
    d2 = dict({“one” : 1, “two” : 2, “three” : 3})
    ```
  - Como el resto de colecciones, soporta:

    ```
    len(d1)  # longitud de parejas
    x in d2  # consulta por clave
    for k, v in d:  # iterar por parejas
    ```
  - Para modificar el diccionario:

    ```
    d[“cuatro”] = 4
    ```

# Python

- Ejercicios:
  - Modificar la Practica 1 haciendo uso de diccionarios.
  - Crear una función que devuelva el conjunto cartesiano de dos conjuntos (conjunto con todos los pares del primero con el segundo).
  - Crear una función que devuelva el número de apariciones de cada carácter en una cadena.
  - Crear una función que devuelva el número de apariciones de cada palabra en una frase.
  - Utilizando el modulo random, crear una función que simule N tiradas de 2 dados y cuente las veces que aparece un resultado.

# Python

- Diccionarios 2:
  - Puede construirse con una lista de tuplas:
  - **d = dict([(“one”, 1), (“three”, 3), (“two”, 2)])**
  - Tambíen funciona con 2 listas separadas y zip():
    **d = dict(zip([“one”, “two”, “three”], [1, 2, 3]))**
  - La funcion zip() se puede usar en un for:
    **for text, number in zip(texts, numbers):**
  - Se puede devolver su contenido en listas:
    **d.keys()  # lista de claves**
    **d.values() # lista de valores**
    **d.items()  # lista de clave valor**
  - Devolver un elemento:
    **d.get(key[, default])  # No existe, default/None**
    **d.pop(key[, default]) # No existe, default/KeyError**

# Python

- Ejercicios:
  - Reimplementar las funciones que cuentan caracteres y palabras.
  - Crear una función que devuelva el diccionario inverso.
  - Crear una función que gestione una agenda, permitirá agregar números de teléfono a personas.
  - Crear una función que escribe una cadena en Morse.
  - Crear la función inversa, de una cadena en Morse devuelve la traducción.