Python

SISTEMAS DE GESTIÓN EMPRESARIAL – 148FA (DAM)

Características

Lenguaje informático: lenguaje utilizado por ordenadores, cuyo objetivo es transmitir información. Tipos de lenguajes informáticos:

a) lenguajes de programación (Python, Java, PHP, Pearl, C,...)

b) lenguajes de especificación (UML)

c) lenguajes de consulta (SQL)

d) lenguajes de marcas (HTML, XML)

e) lenguajes de transformación (XSLT)

f) protocolos de comunicaciones (HTTP, FTP)

Características

- **Lenguaje de programación**: es un lenguaje informático, diseñado para expresar órdenes e instrucciones precisas, que deben ser llevadas a cabo por una computadora. Puede utilizarse para crear programas que controlen el comportamiento físico o lógico de un ordenador.
  Cada lenguaje está compuesto por una serie de símbolos, reglas sintácticas y semánticas que definen su estructura.

- **Lenguajes de alto nivel**: son aquellos cuya característica principal, consiste en una estructura sintáctica y semántica legible, acorde a las capacidades cognitivas humanas. A diferencia de los lenguajes de bajo nivel, son independientes de la arquitectura del hardware, motivo por el cual, asumen mayor portabilidad.

Características

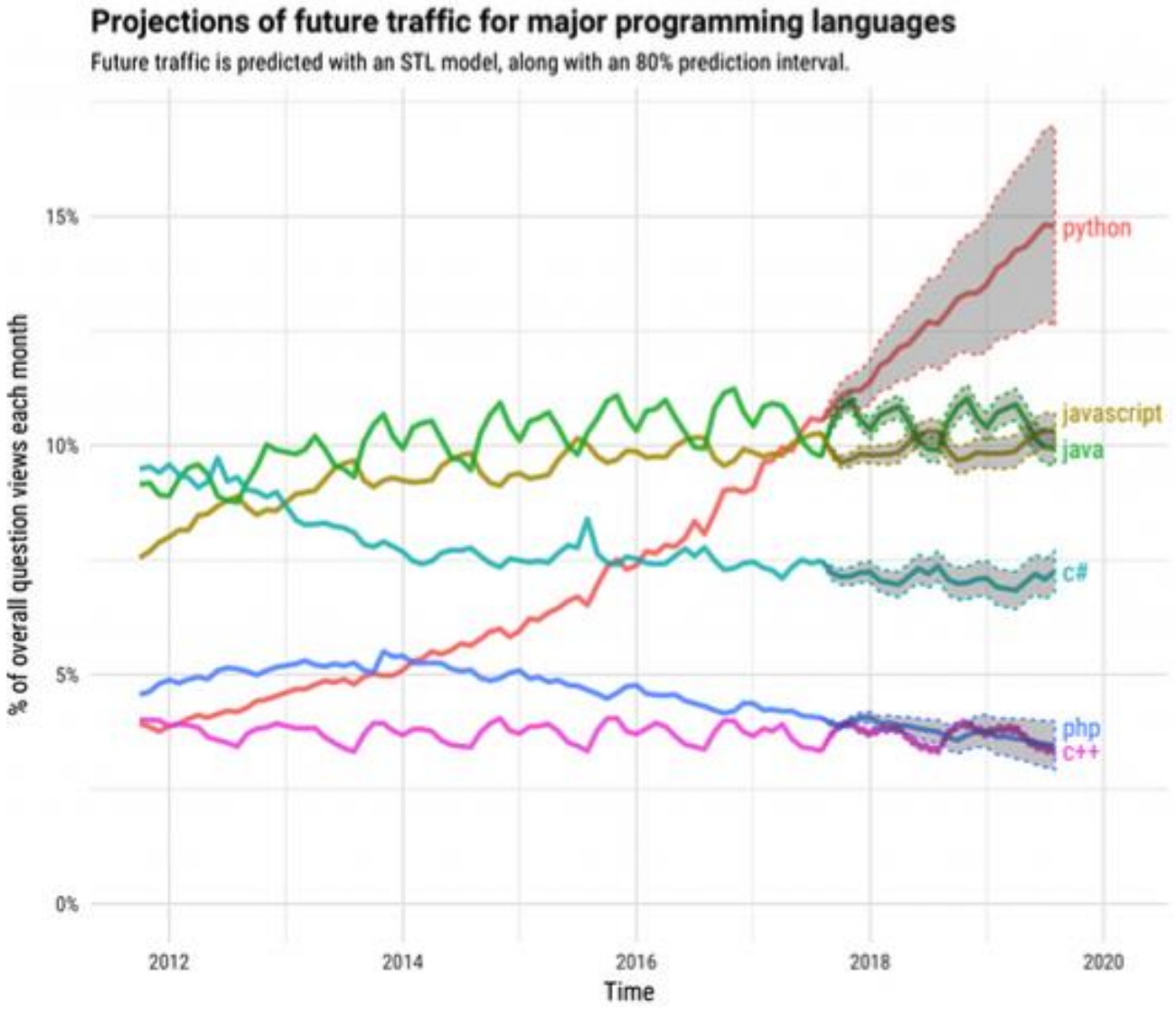
- **Lenguajes interpretados**: a diferencia de los compilados, no requieren de un compilador para ser ejecutados sino de un intérprete. Un intérprete, actúa de manera casi idéntica a un compilador, con la salvedad de que ejecuta el programa directamente, sin necesidad de generar previamente un ejecutable. Ejemplo de lenguajes de programación interpretado son Python, PHP, Ruby, Lisp, entre otros.

- **Tipado dinámico**: un lenguaje de tipado dinámico es aquel cuyas variables, no requieren ser definidas asignando su tipo de datos, sino que éste, se auto-asigna en tiempo de ejecución, según el valor declarado. Debemos tener cuidado, a veces hay que hacer casting (transformar la variable).

Características

- **Multiplataforma**: puede ser interpretado en diversos Sistemas Operativos como GNU/Linux, Windows, Mac OS, Solaris, entre otros.

Uso de Python



Dónde se usa Python

- Aplicaciones de Inteligencia Artificial – Machine Learning
- Big Data
- Desarrollo Web
- Frameworks de pruebas
- Investigación: data science (tratamiento de datos)

Identación

En Python es muy importante la identación para escribir el código correctamente

Se trata de la sangría, como si estuviéramos hablando de un documento de Word, la cual va a identificar el fin de las sentencias if, los bucles, etc.

En el caso de Python, la identación es obligatoria, ya que de ella, dependerá su estructura

```
inicio de la estructura de control:
    expresiones
```

Comentarios

- Comentarios de una sola línea: # texto

  Ejemplo: # Es un comentario de una línea

- Comentarios de más de una línea

```
'''
Este es un comentario de más
de una línea
'''
```

Variables

```
nombre_de_la_variable = valor_de_la_variable
```

- Sin acabar con ;
- No hay :
- Tipado dinámico
- Posibilidad de hacer casting

CONSTANTES: nombre de la variable en mayúsculas

```
MI_VARIABLE_CONSTANTE = 12
```

Variables

Imprimir valor por pantalla: `print ( )`

```
# Ejemplos
mi_variable = 3
print (mi_variable)

print("Hola mundo")

otra_variable = "Hola Mundo"
print (otra_variable)
```

IMPRIMIR TEXTO Y VARIABLES POR PANTALLA

```
mi_variable = 2019
print ("Estamos en el año ", mi_variable)
```

Variables

Tipos de variables

- Cadena de caracteres o string
  ```
  cadena = “Hola Mundo”
  ```
- Número entero
  ```
  numero = 3
  ```
- Número entero octal
  ```
  v_octal = 043
  ```
- Número entero hexadecimal
  ```
  v_hexadecimal = 0x23
  ```
- Número real
  ```
  n_real = 34.25
  ```
- Booleano
  ```
  v_booleana = true
  ```

Operadores aritméticos

Símbolo	Significado	Ejemplo	Resultado
+	Suma	a = 5 + 5	10
-	Resta	a = 9 - 7	2
-	Negación	a = -3	-3
*	Multiplicación	a = 4 * 5	20
**	Exponente	a = 2 ** 4	16
/	División	a = 12.5 / 2	6.25
//	División entera	a = 12.5 / 2	6.0
%	Módulo	a = 18 % 4	2

Operadores relacionales

Símbolo	Significado	Ejemplo	Resultado
==	Igual que	3 == 2	False
!=	Distinto que	coche != moto	True
<	Menor que	4 < 16	True
>	Mayor que	8 > 7	True
<=	Menor o igual que	2 <= 2	True
>=	Mayor o igual que	4 >= 5	False

Operadores lógicos

if (si), elif (sino, si) y else (sino).

Operador	Ejemplo	Explicación	Resultado
and	3 == 7 and 7 < 15	False and False	False
and	9 < 12 and 12 > 7	True and True	True
and	9 < 12 and 12 > 15	True and False	False
or	12 == 12 or 15 < 7	True or False	True
or	7 > 5 or 9 < 12	True or True	True
xor	4 == 4 xor 9 > 3	True o True	False
xor	4 == 4 xor 9 < 3	True o False	True

Bucles

While

```
curso = 2001
while curso <= 2019:
    print (‘Curso ’, str(curso))
    curso += 1
```

```
while True:
    nombre = input(“Identifíquese: ")
    if nombre:
        break
```

En consola estará esperando a que se escriba algo. Una vez hecho, saldrá del bucle

Bucles

For

```
v_lista = ['Estudiante1', ‘Estudiante2', ‘Estudiante3', ‘Estudiante4’]
for nombre in v_lista:
    print nombre
```

- ¿Cómo sería si tuviéramos una variable control que marca el inicio y el fin del bucle?
- ¿Cómo podríamos usar range() para definir el inicio y el final de un bucle *for*?

Ejemplo de range:

```
for i in range(0, 3):
    # i = 0, 1, 2
```

Tipos de datos complejos

- **Tuplas**: variables que guardan datos que no pueden ser modificados. Los datos pueden ser de diferentes tipos

```
v_tupla = (“cadena”, 3.7, 15, “otra cadena”, 75)
```

- **Listas**: son variables que guardan datos de diferentes tipos, pero que sí que pueden ser modificados

```
v_lista = [“cadena”, 3.7, 15, “otra cadena”, 75]
```

- **Diccionarios**: permiten utilizar una clave para declarar y acceder a un valor. En el caso de las listas y tuplas se accede a través de un valor de índice.

```
v_diccionario = {'clave_1': valor_1, 'clave_2': valor_2, 'clave_7': valor_7}
```

Tipos de datos complejos

Ejercicio con listas

- ¿Cómo se accede a un elemento concreto de una lista?
- ¿Cómo se accede a una porción de una lista?
- ¿Cómo añadir elementos a una lista al final?
- ¿Cómo añadir elementos a una lista al principio?
- ¿Cómo añadir elementos a una lista entre otros dos elementos de la misma?

1. Crea una lista con 5 datos de diferente tipo.
2. Para responder a cada pregunta, escribe el código necesario
3. Devuelve el resultado por pantalla

Listas

```
print v_lista[1]  # Devuelve el elemento que está en la posición 1 (el segundo)

print v_lista[1:4]  # Devuelve el segundo, tercer y cuarto elemento

print v_lista[-2]  # Devuelve el segundo elemento empezando por la derecha
```

Las listas permiten cambiar los datos una vez creados:

```
v_lista[1] = 2  # El segundo elemento de la lista ahora es un 2
```

Se pueden agregar más datos a la lista utilizando ***append***():

```
vi_lista.append('Nuevo Dato')     vi_lista.append(24.3)          vi_lista.append(variable)
```

Tipos de datos complejos

Ejercicio con tuplas

- ¿Cómo se accede a un elemento concreto de una tupla?
- ¿Cómo se accede a una porción de una tupla?
- ¿Cómo añadir elementos a una tupla al final?
- ¿Cómo añadir elementos a una tupla al principio?
- ¿Cómo añadir elementos a una tupla entre otros dos elementos de la misma?

1. Crea una tupla con 5 datos de diferente tipo.
2. Para responder a cada pregunta, escribe el código necesario
3. Devuelve el resultado por pantalla

Tipos de datos complejos

Ejercicio con diccionario

```
v_diccionario = {'clave_1': valor_1, 'clave_2': valor_2, 'clave_3': valor_3}
```

- ¿Cómo se accede a un elemento concreto de este diccionario?

1. Crea una diccionario como el ejemplo de arriba
2. Devuelve el resultado por pantalla

```
print mi_diccionario['clave_3']  # Salida: valor_3
```

Tipos de datos complejos

Ejercicio con diccionario

```
v_diccionario = {‘nombre_1': valor_1, ‘nombre_2': valor_2, ‘nombre_3': valor_3}
```

- ¿Cómo se agregan elementos nuevos al diccionario?
- ¿Cómo se eliminan elementos del diccionario?

1. Crea un diccionario como el de arriba.

2. Crea el comando para agregar un elemento.

```
v_diccionario['nombre_4'] = valor_4
```

3. Crea el comando para eliminar un determinado elemento

```
del(mi_diccionario['nombre_2'])
```

Tipos de datos complejos

Ejercicio con diccionario

1. Crea un diccionario, en el que la clave sea un número y el valor, el que elijas (por ejemplo, dorsal como clave y nombre de jugador de baloncesto como valor de esa clave).

2. Saca por pantalla el dorsal, acompañado de una flecha (-->) y el nombre del jugador al que corresponde ese dorsal.

   Por ejemplo,

   4 --> Luis Scola

   5 --> Pablo Prigioni

Ejercicios

1. Crea el código en Python que
    - Pida un número por pantalla. Guárdalo en una variable
    - Súmale 3
    - Devuelve el número por pantalla
    - Devuelve una cadena de caracteres con el resultado obtenido en el paso b

2. **Cadenas de caracteres**. Crea el código que
    - Pida al usuario el nombre y la edad
    - Que devuelva: **“Hola NOMBRE, naciste en el año AÑO”**. En el que NOMBRE es el nombre que has escrito y el año se debe calcular

Ejercicios

3. **Par o impar**.
   - Haz que se pida un número por pantalla
   - Que devuelva si es par o impar
   - Que diga también si es múltiplo de 3, y si no lo es que devuelva otro mensaje

4. **Conversor de temperatura**
   - Introduce una temperatura
   - Que pregunte si es Celsius o Fahrenheit
   - Que muestre por pantalla la temperatura original y la equivalente en la otra unidad de medida

5. **Divisores**. Crea el código en Python que tras pedir un número muestre todos sus divisores. Después, que muestre también los primeros 5 múltiplos.

Ejercicios

6. **Listas**.
   - Crea una lista de números enteros, de 10 elementos
   - Devuelve por pantalla los elementos que sean pares y que al mismo tiempo sean menores que 10.
   - Además de devolver uno a uno cada elemento que cumpla esa condición, devuelve una lista con los mismos números.

Ejercicios

7. **Calculadora simple**

- Crea una calculadora simple que admita **sumar**, **restar**, **multiplicar** y **dividir**.
- El planteamiento sería:
  - Se pide un primer número
  - Se pide un segundo número una vez introducido el primero
  - Se elige un operador
  - Devuelve el resultado “El resultado de OPERACIÓN es RESULTADO”
    - ✓ OPERACIÓN: responde al tipo de operación que se ha realizado
    - ✓ RESULTADO: el resultado de la operación con los dos números

Funciones

```
def mi_funcion():
    # código de la función
```

Llamada a la función: simplemente usando su nombre → `funcion()`

Cuando se **devuelven resultados** (*return*):

```
def funcion():
    return "Hola Mundo"

frase = funcion()
print (frase)
```

Funciones

Parámetros en las funciones

- Los parámetros se incluyen dentro de la función a modo de variables, para utilizarlos dentro de ésta
- Estos parámetros serán variables de ámbito local (variables locales)
- Si intentamos acceder a una de estas variables fuera de la función, nos dará un error

```
funcion(valor1, valor2)
```

```
x=1
y=2
funcion(x, y)
```