

Creación Digital y Pensamiento Computacional

Juan Gualberto Gutiérrez Marín

Marzo 2023

Índice

1	Fundamentos de programación en Python	3
1.1	Conceptos iniciales	3
1.1.1	Instrucción	5
1.1.2	Secuencia	5
1.1.3	Algoritmo	5
1.1.4	Código	7
1.1.5	Tuplas	18
1.2	Estructuras de control selectivas e iterativas	19
1.2.1	Cuadrado: el bucle for..in range()	19
1.3	Funciones	21
1.4	Introducción al uso de funciones gráficas	22
2	Ciberseguridad	23
2.1	Criptografía	23
2.1.1	Introducción	23
2.1.2	Cifrado de llave simétrica o de una clave	23
2.1.3	Estenografía y estegoanálisis	24
2.1.4	Criptografía avanzada	25
2.1.5	Criptografía de llave asimétrica	26
2.2	Hacking ético	27
2.2.1	Introducción	27
2.2.2	Técnicas de búsqueda de información: Information gathering.	29
2.2.3	Escaneo: pruebas de PenTesting	30
2.2.4	Vulnerabilidades en sistemas	31
2.3	Incidentes de ciberseguridad	31
2.3.1	Análisis forense	31
2.3.2	Ciberdelitos	33
2.4	Conceptos previos	35
2.5	Big data. Características. Volumen de datos.	36
2.6	Visualización, transporte y almacenaje de los datos.	37
2.7	Recogida, análisis y generación de datos.	38
2.8	Simulación de fenómenos naturales y sociales	39
2.9	Descripción del modelo	40
2.10	Identificación de agentes	40
2.11	Implementación del modelo mediante un software específico, o mediante programación	41
2.12	Definición. Historia. El test de Turing	42

2.13 Aplicaciones. Impacto	42
2.14 Ética y responsabilidad social (transparencia y discriminación algorítmica)	42
2.15 Beneficios y posibles riesgos	42
2.16 Agentes inteligentes simples	42
2.17 Análisis y clasificación supervisada basada en técnicas de aprendizaje automático: reconocimiento de habla; reconocimiento de imágenes; y reconocimiento de texto . .	42
2.18 Generación de imágenes y/o música basado en técnicas de aprendizaje automático: mezcla inteligente de dos imágenes; generación de música; traducción y realidad aumentada	42

1 Fundamentos de programación en Python

“When you want to know how things really work, study them when they’re coming apart.”

— William Gibson, Zero History

Este tema forma parte del *Bloque I* (desarrollo de aplicaciones informáticas que procesan imágenes, audio y vídeo, como base de la creación digital) de la asignatura Creación Digital y Pensamiento Computacional.

En este tema aprenderemos:

- Conceptos de instrucción y secuenciación, algoritmo vs. código.
- Estructuras de control selectivas e iterativas (finitas e infinitas).
- Funciones.
- Introducción al uso de funciones gráficas (punto, línea, triángulo, cuadrado, rectángulo, círculo, elipse, sectores y arcos).

1.1 Conceptos iniciales

Partimos de la base que sabemos **qué es un ordenador**, tanto el hardware, que es su estructura física (circuitos electrónicos, cables, caja o gabinete, teclado, ratón, etc.), y el software, que es su parte intangible (programas, datos, información, documentación, etc.).

Desde el punto de vista funcional es una máquina que posee, al menos, una unidad central de procesamiento (CPU), una unidad de memoria y otra de entrada/salida (periféricos: pantalla, teclado, ratón, micrófono, cámara, tarjeta de red, puertos USB). Los periféricos de entrada permiten el ingreso de datos, la CPU se encarga de su procesamiento (operaciones aritmético-lógicas) y los dispositivos de salida los comunican a los medios externos. Es así, que la computadora recibe datos, los procesa y emite la información resultante, la que luego puede ser interpretada, almacenada, transmitida a otra máquina o dispositivo o sencillamente impresa; todo ello a criterio de un operador o usuario y bajo el control de un programa de computación.

Para *hablar* con los ordenadores, usamos lenguajes de programación.

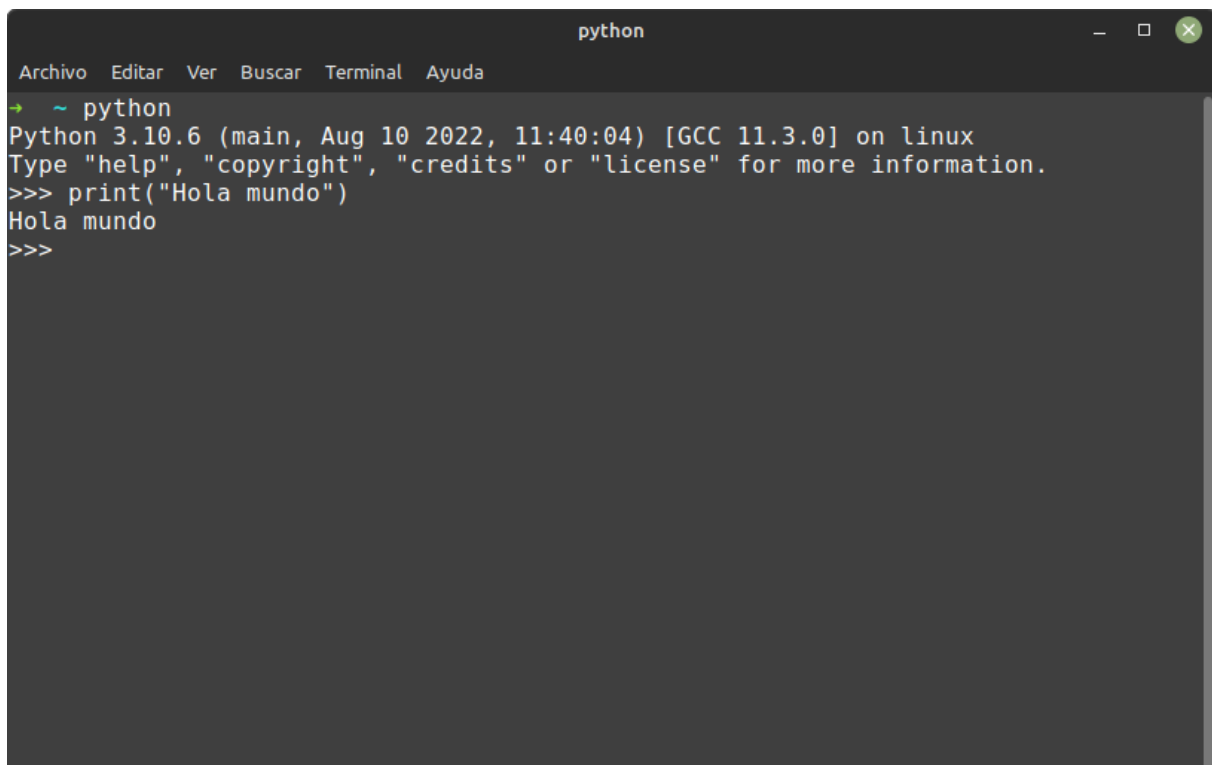
Un **lenguaje de programación** es un lenguaje formal (o artificial, es decir, un lenguaje con reglas gramaticales bien definidas) que le proporciona a una persona, en este caso el programador, la capacidad de escribir (o programar) una serie de instrucciones o secuencias de órdenes en forma de algoritmos con el fin de controlar el comportamiento físico o lógico de un sistema informático, de manera que se puedan obtener diversas clases de datos o ejecutar determinadas tareas. A todo este conjunto de órdenes escritas mediante un lenguaje de programación se le denomina programa informático.

Al igual que idiomas, existen gran cantidad y variedad de lenguajes de programación, entre los que destacan por ser los más usados lenguajes como Python, C, Java, o JavaScript. Puedes consultar la lista de los más usados en prestigiosos índices como:

- TIOBE
- IEEE

Nosotros vamos a usar el lenguaje de programación Python para todos nuestros ejercicios. Puedes instalar Python desde el App Store de tu sistema operativo o bien directamente desde su página Web, haciendo clic aquí mismo.. Debes instalar la versión 3.10 o superior porque hay estructuras de control como *switch-case* (lo veremos más adelante) que sólo están disponibles a partir de esta versión. Como entorno de desarrollo usaremos (Visual Studio Code)[<https://code.visualstudio.com/download>] con la “Python Extension Pack” añadida.

Cuando lo tengas instalado, abre una terminal de tu sistema operativo y ejecuta el comando `python` para entrar en modo interactivo. Cuando veas los símbolos `>>>` copia y pega las órdenes de este tutorial para ir comprobando qué hacen. También puedes ir guardando cada ejercicio en un fichero con extensión `.py` y desde la terminal ejecutando `python3 fichero.py` o bien desde el botón ejecutar de tu IDE (un IDE es un entorno de desarrollo, un software para programar) favorito.



```
python
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
~ python
Python 3.10.6 (main, Aug 10 2022, 11:40:04) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hola mundo")
Hola mundo
>>>
```

Figura 1: Python en modo interactivo

1.1.1 Instrucción

Llamamos instrucción a:

- cada una de las órdenes que una persona da al ordenador para que ejecute una operación.
- conjunto de datos insertados en una secuencia estructurada o específica que el procesador interpreta y ejecuta.

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

1.1.2 Secuencia

Una secuencia de instrucciones son dos o más operaciones que se ejecutan una detrás de otra, en orden secuencial, de ahí su nombre.

A continuación vemos un ejemplo de una secuencia de tres instrucciones (si estás en modo interactivo copia y ejecuta línea a línea o no funcionará):

```
1 print("Hola ¿cómo estás?")
2 nombre = input("¿Cómo te llamas? ")
3 print("Hola, ", nombre, " encantado/a de saludarte")
```

Sólo con estas tres líneas, estamos aprendiendo varios conceptos muy interesantes:

- el **operador** de *asignación*: cuando veas sólo un igual =, significa asignar, es decir, lo que haya a la derecha del igual se va a copiar en lo que haya a la izquierda del mismo.
- la **variable** *nombre*: una variable es como un nombre, es como llamamos a una zona de memoria RAM donde temporalmente almacenamos información que es relevante para el programa. En este caso la llamamos *nombre* para poder recordarla y consultarla cuando sea necesario. Fíjate cómo usamos el operador asignación (un igual) para indicar que la salida de la función que está a la derecha del mismo, se va a guardar en la variable *nombre* que está a la izquierda del igual.
- la **función** *print*: De manera muy intuitiva podemos afirmar que esta función lo que hará es mostrar un mensaje por pantalla. Fíjate cómo el mensaje va entre paréntesis. Es una instrucción de salida, porque la información fluye de
- la **función** *input*: Esta función, como su nombre indica, pide al usuario que teclee algo y lo guarda en la variable donde hacemos la asignación.

1.1.3 Algoritmo

Un algoritmo es un conjunto de instrucciones o reglas definidas y no-ambiguas, ordenadas y finitas que permite, típicamente, solucionar un problema, realizar un cómputo, procesar datos y llevar a cabo

A screenshot of a terminal window titled 'python'. The window has a menu bar with 'Archivo', 'Editar', 'Ver', 'Buscar', 'Terminal', and 'Ayuda'. The terminal shows the following text:

```
→ ~ python
Python 3.10.6 (main, Aug 10 2022, 11:40:04) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hola ¿cómo estás?")
Hola ¿cómo estás?
>>> nombre = input("¿Cómo te llamas? ")
¿Cómo te llamas? Juan
>>> print("Hola, ", nombre, " encantado/a de saludarte")
Hola,  Juan  encantado/a de saludarte
>>>
```

Figura 2: Ejecutando instrucciones una a una.

otras tareas o actividades.

1.1.4 Código

El código fuente de un programa informático (o software) es un conjunto de líneas de texto con los pasos que debe seguir la computadora para ejecutar un programa.

El código fuente de un programa está escrito por un programador en algún lenguaje de programación legible por humanos, normalmente en forma de texto plano.

Este código fuente escrito en un lenguaje legible por humanos no es directamente ejecutable por la computadora en su primer estado, sino que debe ser traducido a otro lenguaje o código binario; así será más fácil para la máquina interpretarlo (lenguaje máquina o código objeto que sí pueda ser ejecutado por el hardware de la computadora). Para esta traducción se usan los llamados compiladores, ensambladores, intérpretes, transpiladores y otros sistemas de traducción.

1.1.4.1 Tipos de datos en Python Al igual que por ejemplo en matemáticas tenemos diferentes conjuntos para los números (naturales, enteros, racionales o reales, complejos o imaginarios...), en los lenguajes de programación, cada variable tiene asociado un tipo que indica qué representan los 0 y 1 que realmente están almacenados en la memoria RAM del ordenador.

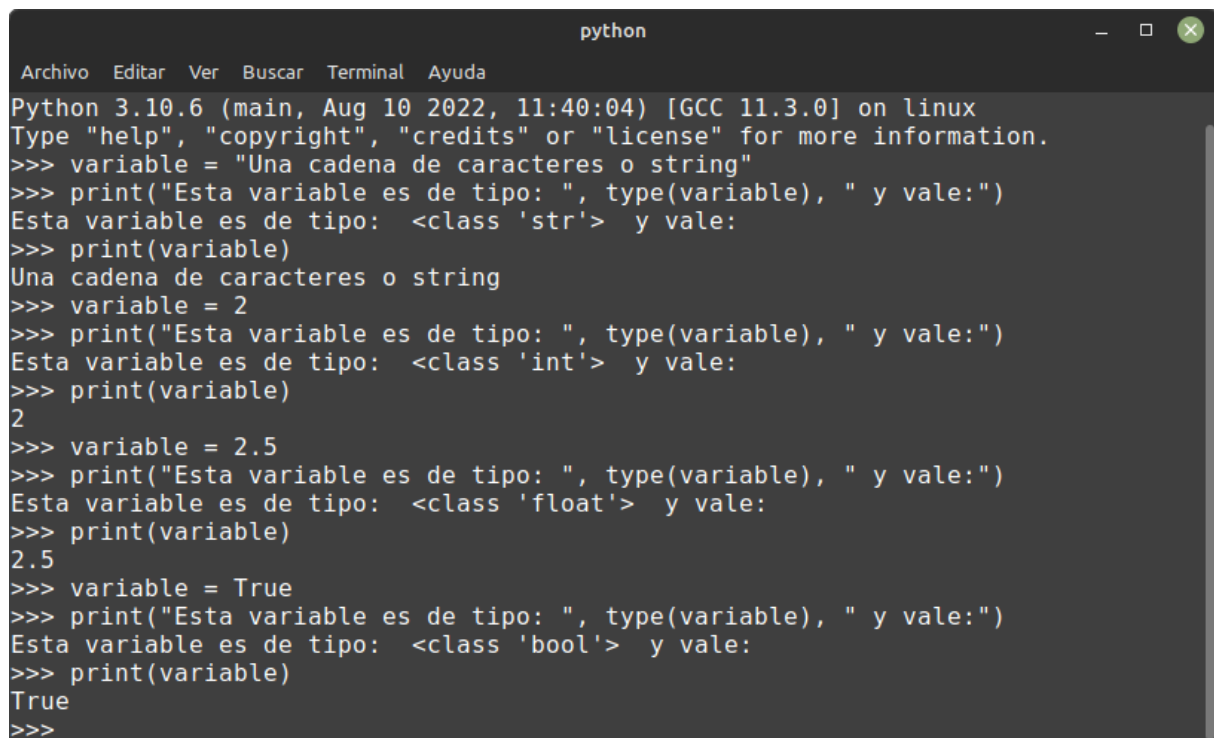
En Python, todo valor que pueda ser asignado a una variable tiene asociado un tipo de dato (aunque en Python todas son objetos). Así que los tipos de datos serían los esqueletos o clases (donde se definen las propiedades del objeto y qué se puede hacer con él, ej: un coche tiene matrícula, marca, modelo, color, titular) y las variables serían las instancias (objetos que representan algo del mundo real, mi coche con matrícula: J-1234-A, color: blanco, marca: Renault, modelo: Fuego...).

Ejemplos:

```
1 # cadena / string
2 variable = "Una cadena de caracteres o string"
3 print("Esta variable es de tipo: ", type(variable), " y vale:")
4 print(variable)
5 # number / entero
6 variable = 2
7 print("Esta variable es de tipo: ", type(variable), " y vale:")
8 print(variable)
9 # float / decimal
10 variable = 2.5
11 print("Esta variable es de tipo: ", type(variable), " y vale:")
12 print(variable)
13 # verdadero-falso / booleano
14 variable = True
15 print("Esta variable es de tipo: ", type(variable), " y vale:")
```



```
16 print(variable)
```



```
python
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Python 3.10.6 (main, Aug 10 2022, 11:40:04) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> variable = "Una cadena de caracteres o string"
>>> print("Esta variable es de tipo: ", type(variable), " y vale:")
Esta variable es de tipo: <class 'str'> y vale:
>>> print(variable)
Una cadena de caracteres o string
>>> variable = 2
>>> print("Esta variable es de tipo: ", type(variable), " y vale:")
Esta variable es de tipo: <class 'int'> y vale:
>>> print(variable)
2
>>> variable = 2.5
>>> print("Esta variable es de tipo: ", type(variable), " y vale:")
Esta variable es de tipo: <class 'float'> y vale:
>>> print(variable)
2.5
>>> variable = True
>>> print("Esta variable es de tipo: ", type(variable), " y vale:")
Esta variable es de tipo: <class 'bool'> y vale:
>>> print(variable)
True
>>>
```

Figura 3: Tipos básicos de datos

1.1.4.2 Números enteros Los números enteros, *integer* para Python, es cualquier número, positivo o negativo, sin decimales y, muy importante, **de longitud ilimitada**.

Ejemplos:

```
1 x = 2
2 y = 2342893749287492334356
3 z = -6555522
```

1.1.4.3 Números reales Los números reales, en programación los llamamos **de coma flotante**, son cualquier número positivo o negativo, que contenga al menos un decimal.

Ejemplos:

```
1 x = 3.141592
2 y = 1.0
3 z = -234.59
```

También es posible usar notación científica en Python (expresar números en potencia de 10):

```
1 x = 3.5321e4
2 y = 12E4
3 z = -127.7e100
```

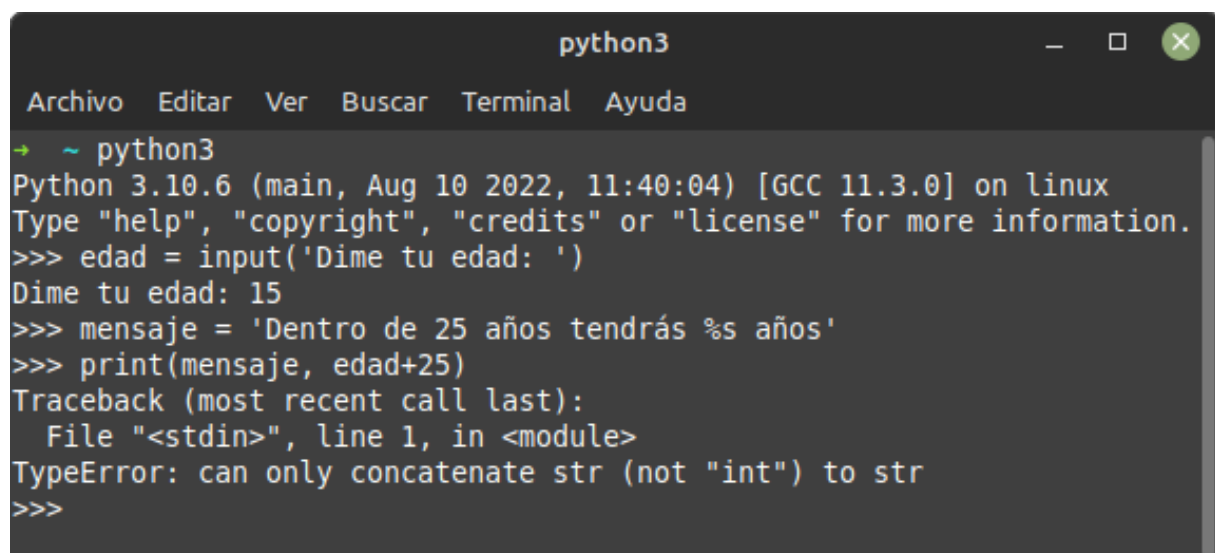
1.1.4.4 Números complejos Aunque los números complejos probablemente no los uses salvo en ecuaciones diferenciales, algunos tipos de integrales en aerodinámica, hidrodinámica y electromagnetismo entre otras.

A diferencia nuestra (probablemente uses i para la parte imaginaria del número), en Python se usa la letra **j**:

```
1 x = 3+5j
2 y = 5j
3 z = -5j
```

1.1.4.5 Conversión de números Como ya vimos en un apartado anterior, cuando quiero pedirle a un usuario que introduzca un texto por pantalla, uso la función `input`. Esta función pide **un texto**, aunque yo introduzca un número, seguirá siendo de tipo texto, es decir no es lo mismo asignar `x = 5`, que `x = '5'` (si pongo comillas será el texto 5 sin ellas, el número 5). Prueba el siguiente código (recuerda ir línea a línea copiando, pegando y ejecutando si estás en modo interactivo):

```
1 edad = input('Dime tu edad: ')
2 mensaje = 'Dentro de 25 años tendrás %s años'
3 print(mensaje, edad+25)
```

A screenshot of a terminal window titled 'python3'. The window has a menu bar with 'Archivo', 'Editar', 'Ver', 'Buscar', 'Terminal', and 'Ayuda'. The terminal shows the following interaction: the user runs 'python3', the prompt shows 'Python 3.10.6 (main, Aug 10 2022, 11:40:04) [GCC 11.3.0] on linux'. The user enters '>>> edad = input('Dime tu edad: ')', the prompt shows 'Dime tu edad: 15'. The user enters '>>> mensaje = \'Dentro de 25 años tendrás %s años\'', the prompt shows '>>> print(mensaje, edad+25)'. The user enters '>>>', and the terminal displays a 'Traceback (most recent call last):' followed by 'File "<stdin>", line 1, in <module>' and 'TypeError: can only concatenate str (not "int") to str'. The prompt '>>>' is shown at the bottom.

```
python3
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
→ ~ python3
Python 3.10.6 (main, Aug 10 2022, 11:40:04) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> edad = input('Dime tu edad: ')
Dime tu edad: 15
>>> mensaje = 'Dentro de 25 años tendrás %s años'
>>> print(mensaje, edad+25)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>>
```

Figura 4: Error al intentar sumar una cadena de caracteres con un número entero.

Para solventar el error que da, tenemos que convertir la cadena de caracteres almacenada en la variable edad a un número. Para convertir a número tenemos las siguientes funciones:

```
1 entero = int("27")
2 real = float("27.3")
3 imaginario = complex("2+3j")
```

Con estos ejemplos, ¿sabrías arreglar el error anterior? Inténtalo sin mirar la solución.

```
1 textoEdad = input('Dime tu edad: ')
2 edad = int(textoEdad)
3 mensaje = 'Dentro de 25 años tendrás %s años'
4 print(mensaje, edad+25)
```

A la conversión de tipos también se le llama **casting** en el argot de los programadores.

1.1.4.6 Operadores Operadores matemáticos:

Operador	Operación	Ejemplo
+	Suma	$x + y$
-	Resta	$x - y$
	Multiplicación	$x * y$
/	División	x / y
	Potencia	$x ** y$
%	Módulo	$x \% y$
//	División con redondeo	$x // y$

Al igual que pasa en matemáticas, en programación tenemos exactamente la misma precedencia de operadores, es decir en caso de concatenar operaciones, se harían primero las potencias, luego las multiplicaciones, divisiones, módulo y finalmente las sumas y restas. Compruébalo con estos ejemplos (verifica en tu ordenador qué resultado dan y piensa porqué):

```
1 print( 2+3*5 ) # 17
2 print( (2+3)*5 ) # 20
```

Otros operadores de asignación:

Operador	Ejemplo	Equivalente a
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
=	x	= 3
^=	x ^= 3	x = x ^ 3
»=	x »= 3	x = x » 3
«=	x «= 3	x = x « 3

Operadores de comparación:

Operador	Operación	Ejemplo
==	Igual a	x == y
!=	Distinto a	x != y
>	Mayor a	x > y
<	Menor a	x < y
>=	Mayor o igual a	x >= y
<=	Menor o igual a	x <= y

1.1.4.7 Booleanos Una vez vistos los operadores de comparación, vamos a seguir con ellos y ver para qué los podemos usar. Si tecleamos esto en nuestro Python en modo interactivo:

```
1 print(10 > 5)
2 print(5 == 9)
```

```
3 print(12 < 9)
```

Veremos que devuelve:

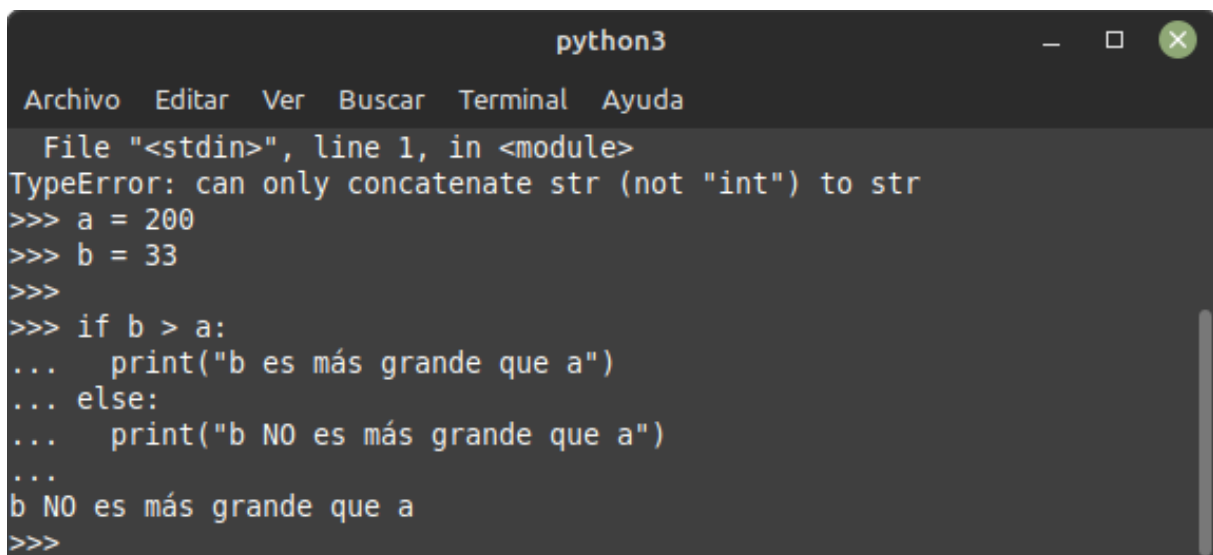
```
1 True
2 False
3 False
```

A esto es a lo que llamamos tipos booleanos, es decir, un tipo de dato binario que sólo tiene dos estados: verdadero y falso.

¿Para qué sirven estos tipos de datos? Nos ayudarán a tomar decisiones en nuestros programas y así ir por un camino u otro de nuestro código fuente, ejemplo:

```
1 a = 200
2 b = 33
3
4 if b > a:
5     print("b es más grande que a")
6 else:
7     print("b NO es más grande que a")
```

Si lo introduces en modo interactivo, copia y pega todo de golpe y al final tendrás que pulsar la tecla intro (o *Enter*) para indicar que la estructura de control ha terminado.

A screenshot of a Python3 interactive shell window. The window has a title bar with 'python3' and standard window controls. The menu bar includes 'Archivo', 'Editar', 'Ver', 'Buscar', 'Terminal', and 'Ayuda'. The main area shows the following text:

```
File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>> a = 200
>>> b = 33
>>>
>>> if b > a:
...     print("b es más grande que a")
... else:
...     print("b NO es más grande que a")
...
b NO es más grande que a
>>>
```

Figura 5: Ejemplo de uso de booleanos para tomar decisiones. Recuerda un intro extra al final.

1.1.4.8 Strings Los *strings* o cadena de caracteres son variables de tipo texto, es decir almacenan información que contienen caracteres de texto. Sabemos que hablamos de cadenas de caracteres

porque usamos comillas para darles valor, ejemplo:

```
1 cadena = "Esto es una variable de tipo cadena"
```

Operaciones con strings:

Con las cadenas de caracteres podemos:

Método	Descripción
capitalize()	Convierte a mayúsculas todas las letras
count('a')	Cuenta cuantas 'a' hay en el string (puedes poner otros caracteres a contar)
find('mundo')	Busca dónde aparece la subcadena <i>mundo</i> dentro de la cadena dada
format()	Sirve para dar formato
index(valor)	Busca y devuelve la primera ocurrencia de <i>valor</i> en la cadena
isdigit()	Devuelve verdadero si todos los caracteres son dígitos: 0-9
islower()	Devuelve verdadero si todas las letras están en mayúscula
isnumeric()	Devuelve verdadero si todos los caracteres son numéricos
isupper()	Devuelve verdadero si todas las letras están en mayúscula
join()	Convierte una lista en una cadena, es lo contrario de split()
lower()	Pasa a minúsculas todas las letras
replace(valor1, valor2)	Sustituye cualquier ocurrencia de valor1 por valor2
rfind(valor)	Busca por la derecha de la cadena la primera ocurrencia de un valor y devuelve su posición
rindex(valor)	Busca por la derecha de la cadena la primera ocurrencia de un valor y devuelve su posición
split(parametro)	Parte una cadena en una lista de cadenas según el separador <i>parámetro</i> , es lo contrario de join()

Método	Descripción
startswith()	Devuelve verdadero
strip()	Elimina los caracteres en blanco del principio y el final
swapcase()	Cambia las mayúsculas a minúsculas y al revés
title()	Convierte A Formato Título Con La Primera En Mayúscula
upper()	Pasa a mayúsculas todas las letras

Veamos algunos ejemplos:

El método `count()` retorna el número de veces que se repite un conjunto de caracteres especificado.

```
1 >>> s = "Hola mundo"
2 >>> s.count("Hola")
3 1
```

Los métodos `find()` e `index()` retornan la ubicación (comenzando desde el cero) en la que se encuentra el argumento indicado.

```
1 >>> s.find("mundo")
2 5
3 >>> s.index("mundo")
4 5
```

Difieren en que esta última lanza `ValueError` cuando el argumento no es encontrado, mientras que aquella retorna `-1`.

```
1 >>> s.find("world")
2 -1
3 >>> s.index("world")
4 Traceback (most recent call last):
5   File "<stdin>", line 1, in <module>
6   ValueError: substring not found
```

En ambos métodos la búsqueda ocurre de izquierda a derecha. Para buscar un conjunto de caracteres desde el final, utilícese del mismo modo `rfind()` y `rindex()`.

```
1 >>> s = "C:/python36/python.exe"
2 >>> s.find("/") # Retorna la primera ocurrencia.
3 2
4 >>> s.rfind("/") # Retorna la última.
5 11
```

`startswith()` y `endswith()` indican si la cadena en cuestión comienza o termina con el conjunto de

caracteres pasados como argumento, y retornan True o False en función de ello.

```
1 >>> s = "Hola mundo"
2 >>> s.startswith("Hola")
3 True
4 >>> s.endswith("mundo")
5 True
6 >>> s.endswith("world")
7 False
```

Ambos métodos son preferidos ante la opción de emplear slicing.

```
1 # Se prefiere startswith().
2 >>> s[:4] == "Hola"
3 True
```

Veamos más ejemplos:

```
1 cadena = "Hello world!"
2 # A partir del carácter 4 hasta el final
3 cadena[4:]
4 # Los 4 primeros caracteres
5 cadena[:4]
6 # Desde la posición 2 a la 4
7 cadena[2:4]
8 # El carácter (letra) de la posición 6
9 cadena[6]
```

Si lo probamos en modo interactivo, veremos lo siguiente:

```
1 >>> cadena = "Hello world!"
2 >>> cadena[4:]
3 'o world!'
4 >>> cadena[:4]
5 'Hell'
6 >>> cadena[2:4]
7 'll'
8 >>> cadena[6]
9 'w'
```

Los métodos `isdigit()`, `isnumeric()` e `isdecimal()` determinan si todos los caracteres de la cadena son dígitos, números o números decimales.

```
1 >>> "1234".isnumeric()
2 True
3 >>> "1234".isdecimal()
4 True
5 >>> "abc123".isdigit()
6 False
```


Si bien estas definiciones resultan a priori similares, no lo son. La primera, `isdigit()`, considera caracteres que pueden formar números, incluidos aquellos correspondientes a lenguas orientales. `isnumeric()` es más amplia, pues incluye también caracteres de connotación numérica que no necesariamente son dígitos (por ejemplo, una fracción). La última, `isdecimal()`, es la más restrictiva al tener en cuenta únicamente números decimales; esto es, formados por dígitos del 0 al 9.

`lower()` y `upper()` retornan una copia de la cadena con todas sus letras en minúsculas o mayúsculas según corresponda.

```
1 >>> "Hola Mundo!".lower()
2 'hola mundo!'
3 >>> "Hola Mundo!".upper()
4 'HOLA MUNDO!'
```

Las funciones `strip()`, `lstrip()` y `rstrip()` remueven los espacios en blanco que preceden y/o suceden a la cadena.

```
1 >>> s = " Hola mundo! "
2 >>> s.strip()
3 'Hola mundo!'
4 # Remueve los de la derecha.
5 >>> s.rstrip()
6 ' Hola mundo!'
7 # Remueve los de la izquierda.
8 >>> s.lstrip()
9 'Hola mundo! '
```

Por último, el método `replace()` -ampliamente utilizado- reemplaza una cadena por otra.

```
1 >>> s = "Hola mundo"
2 >>> s.replace("mundo", "world")
3 'Hola world'
```

El método de división de una cadena según un caracter separador más empleado es `split()`, cuyo separador por defecto son espacios en blanco y saltos de línea.

```
1 >>> "Hola mundo!\nHello world!".split()
2 ['Hola', 'mundo!', 'Hello', 'world!']
```

El separador puede indicarse como argumento.

```
1 >>> "Hola mundo!\nHello world!".split(" ")
2 ['Hola', 'mundo!\nHello', 'world!']
```

O bien, para separar únicamente según saltos de línea:

```
1 # Equivalente a split("\n").
2 >>> "Hola mundo!\nHello world!".splitlines()
```

```
3 ['Hola mundo!', 'Hello world!']
```

Un segundo argumento en `split()` indica cuál es el máximo de divisiones que puede tener lugar (-1 por defecto para representar una cantidad ilimitada).

```
1 >>> "Hola mundo hello world".split(" ", 2)
2 ['Hola', 'mundo', 'hello world']
```

Un segundo método de separación es `partition()`, que retorna una tupla de tres elementos: el bloque de caracteres anterior a la primera ocurrencia del separador, el separador mismo, y el bloque posterior.

```
1 >>> s = "Hola mundo. Hello world!"
2 >>> s.partition(" ")
3 ('Hola', ' ', 'mundo. Hello world!')
```

Strings dentro de strings:

Es posible sustituir dentro de una cadena de caracteres otra cadena (o incluso otro tipo de dato). Veámoslo con un ejemplo, pruébalo en tu ordenador y fíjate cómo se sustituye **%s** por el número **1000**:

```
1 puntos = 1000
2 mensaje = 'Has conseguido %s puntos'
3 print(mensaje % puntos)
```

- Primero usamos la cadena **%s** para indicar que eso debe ser sustituido.
- Luego usamos el operador **%** para decir que dentro de la variable *mensaje* sustituimos la cadena anterior por el contenido de la siguiente variable, que es *puntos*.

Multiplícame por cero:

Es posible multiplicar strings en Python y así conseguir curiosos efectos. Prueba a ver qué hace esto:

```
1 print('-'*10, '='*5, '-'*10)
```

1.1.4.9 Listas o arrays ¿Qué son las listas? Las listas, como en la vida real, representan un conjunto ordenado (es importante este punto, están en orden y cada uno en su lugar) de cosas. Veamos un ejemplo con esta lista de la compra de Saruman (mago arcano muy conocido):

```
1 listaCompra = ['patas de araña', 'alas de murciélago',
2               'ojo de tritón', 'dedo gordo de una rana']
3 print(listaCompra)
4 print(listaCompra[2])
5 listaCompra[3] = 'mantequilla de babosa'
6 print(listaCompra)
```

¿Has probado qué muestran las tres instrucciones *print* de estas tres líneas de código fuente?

La primera instrucción *print* te muestra la lista entera, la segunda, el elemento en tercer lugar, curioso porque hemos puesto un 2, no un 3, pero recuerda que eso es así porque en informática siempre empezamos a contar en 0 (el tercer elemento de 0, 1, 2, 3, es el 2, uno menos). La tercera muestra una lista diferente, algo ha cambiado, ejecútalo en tu ordenador, fíjate bien e intenta explicar qué y porqué ha cambiado.

Añadiendo elementos a una lista:

Para añadir elementos a una lista usamos el método *append*. Ejemplo:

```
1 listaCompra = ['patas de araña', 'alas de murciélago',  
2               'ojo de tritón', 'dedo gordo de una rana']  
3 print(listaCompra)  
4 listaCompra.append('cicuta')  
5 listaCompra.append('eructo de oso')  
6 print(listaCompra)
```

Quitando elementos de una lista:

Para eliminar elementos de la lista usamos el comando del (abreviatura del inglés *delete*):

```
1 listaCompra = ['patas de araña', 'alas de murciélago',  
2               'ojo de tritón', 'dedo gordo de una rana']  
3 print(listaCompra)  
4 del listaCompra[0]  
5 del listaCompra[2]  
6 print(listaCompra)
```

1.1.5 Tuplas

Son como las listas pero usan paréntesis y son inmutables. ¿Qué significa inmutable? Pues prueba este código y observa el error que da:

```
1 tupla = (1,2,3,4,5)  
2 tupla[2] = 'pepe'
```

Como habrás comprobado dice que las tuplas no admiten asignación, es decir, no se puede modificar el contenido de la tupla después de crearlo, a diferencia de las listas que sí. A cambio, cuando iteramos o recorremos la tupla, Python lo hace de manera mucho más rápida y eficiente que cuando recorremos o iteramos sobre una lista.

1.1.5.1 Diccionarios Los diccionarios podríamos verlos como un tipo de dato que representa un objeto. Los diccionarios se caracterizan por ir entre llaves y llevar un dato asociado a una clave de esta

manera:

```
1 {clave: valor}
```

Donde la clave nos sirve para explicar qué dato almacena (ese dato sería el valor):

```
1 persona = {'nombre': 'Juan', 'apellido': 'García', 'edad': 16}
2 persona['nombre']
```

1.2 Estructuras de control selectivas e iterativas

Para entender mejor las estructuras de control, vamos a usar un paquete de Python llamado **turtle**. Así, de camino que aprendemos estructuras de control, hacemos una introducción al uso de funciones gráficas. En Ubuntu necesitaremos instalar el paquete Tk para disponer de las bibliotecas necesarias para usar ventanas en nuestros programas.

```
1 sudo apt install python3-tk
2 sudo apt install python3-pip
3 sudo pip3 install PythonTurtle
```

Si no dispones de un ordenador (Windows o Linux) donde instalar Python, puedes hacer esta parte del tutorial también online en esta Web: <https://www.pythonsandbox.com/turtle>. En ese mismo sitio Web también hay un resumen muy interesante de los comandos que vamos a usar: <https://www.pythonsandbox.com/turtle>.

1.2.1 Cuadrado: el bucle `for..in range()`

Nuestro primer dibujo será un cuadrado, presta atención a las instrucciones avanza y gira a la izquierda.

```
1 import turtle as tortuga
2
3 pantalla = tortuga.getscreen()
4
5 tortuga.forward(100)
6 tortuga.left(90)
7
8 tortuga.forward(100)
9 tortuga.left(90)
10
11 tortuga.forward(100)
12 tortuga.left(90)
13
14 tortuga.forward(100)
15 tortuga.left(90)
```

```
16
17  tortuga.done()
```

En donde:

- La instrucción `tortuga.getscreen()` nos “pinta” la ventana donde aparece la tortuga (el pequeño triángulo).
- La instrucción `tortuga.done()` espera a que cerremos la ventana para que podamos ver el dibujo que hagamos.
- La instrucción `tortuga.forward()` hace que avance la tortuga el número de espacios que se le indica.
- La instrucción `tortuga.left()` gira a la izquierda el número de grados indicados.

Por supuesto, muchos ya estaréis pensando, ¿y si las partes que se repiten las ponemos dentro de un bucle? Pues que entonces tendremos un programa mucho más sencillo, algo como esto:

```
1  import turtle as tortuga
2
3  pantalla = tortuga.getscreen()
4
5  for contador in range(4):
6      tortuga.forward(100)
7      tortuga.left(90)
8
9  tortuga.done()
```

Fíjate bien en los espacios en blanco a la izquierda dentro del bucle `for`, así le indicamos a Python que esas instrucciones están *dentro* del `for`, A esto se le llama **identar** el código. Usaremos la tecla tabulador si el programa o IDE que usamos no lo hace de manera *automática* por nosotros.

RETOS: Acabamos de ver cómo dibujar un cuadrado, ¿serías capaz de hacer un pentágono? ¿y un hexágono? ¿y un octógono? PISTA: según el número de vértices, tienes que cambiar el número de vueltas (iteraciones) que da el bucle `for` y los grados que giras.

Los bucles se pueden anidar, esto es, poner unos dentro de otros. Es importante tener en cuenta la indentación, pues recuerda que según cómo lo movemos a la derecha con el tabulador, le indicamos a Python si una línea de código está dentro del bloque anterior. Prueba este código en tu ordenador a ver qué hace:

```
1  # ahora parametrizamos lado y metemos dos FOR anidados
2  # esto pinta polígonos dentro de polígonos
3  lado = 10
4  for k in range(10):
5      for i in range(8):
6          print("Vuelta número: ", i)
7          tortuga.forward(lado)
```

```
8     tortuga.left(45)
9     lado = lado + 20
```

RETO: Hemos visto cómo hacer que se pinten polígonos de más pequeño a más grande, pero, ¿serías capaz de hacerlo al revés? Pintar primero el más grande, luego dentro otro más pequeño y así sucesivamente. **PISTA:** Hay que *jugar* con la longitud del lado, pero ¡cuidado! *un lado no puede medir un número negativo*.

1.3 Funciones

Tras dibujar un cuadrado, un hexágono y un octógono, vemos que parte del código se repite, de manera que se podría *parametrizar* (hacerlo genérico usando variables o parámetros). Si sabemos el número de vértices y la longitud de cada lado, podríamos pintar cualquier polígono (el triángulo sería una pequeña excepción a esta regla), sabiendo que cada vértice tendrá $360/n$ grados.

```
1 import turtle as tortuga
2
3 pantalla = tortuga.getscreen()
4
5 def pintapoligono(lados, longitud):
6     if (lados > 2):
7         giro = 360 / lados
8         for contador in range(lados):
9             tortuga.forward(longitud)
10            tortuga.left(giro)
11     else:
12         print("Error: no hay polígonos de menos de 3 lados")
13
14 pintapoligono(2, 50)
15 pintapoligono(4, 25)
16 pintapoligono(6, 25)
17 pintapoligono(8, 25)
18
19 tortuga.done()
```

También son funciones las que usamos para comunicarnos con la tortuga así por ejemplo tenemos las siguientes:

FUNCIÓN	EXPLICACIÓN
tortuga.up()	Sube el lápiz (ya no pinta)
tortuga.down()	Baja el lápiz (ahora pinta)
tortuga.forward(n)	Avanza n “espacios”

FUNCIÓN	EXPLICACIÓN
<code>tortuga.backward(n)</code>	Retrocede n “espacios”
<code>tortuga.left(n)</code>	Gira a la izquierda “ n ” grados
<code>tortuga.right(n)</code>	Gira a la derecha “ n ” grados
<code>tortuga.reset()</code>	Borra la pantalla y la tortuga vuelve al inicio
<code>tortuga.clear()</code>	Borra la pantalla pero la tortuga se queda ahí
<code>tortuga.pensize(n)</code>	Establece el ancho de la línea a n píxeles
<code>tortuga.pencolor(“red”)</code>	Pone el color del lápiz a rojo
<code>tortuga.pencolor((R,G,B))</code>	Pone el color según la tupla R,G,B
<code>tortuga.fillcolor(“red”)</code>	Rellena con el color rojo
<code>tortuga.fillcolor((R,G,B))</code>	Rellena con el color según la tupla R,G,B

También podemos colorear combinando funciones, ejemplo:

```
1 turtle.color("black", "red")
2 turtle.begin_fill()
3 turtle.circle(80)
4 turtle.end_fill()
```

RETOS: ¿Te está gustando la tortuga de Python? Ahora te proponemos unos retos para que practiques:

1. Dibuja la bandera de España con la tortuga. Extra: ¡ponle un mástil!
2. Dibuja una casita (como la que hacen los niños pequeños, con figuras geométricas sólo: triángulos, cuadrados y rectángulos)
3. Dibuja un barco (como los que hacen los niños pequeños, con figuras geométricas sólo: triángulos, cuadrados y rectángulos)

1.4 Introducción al uso de funciones gráficas

Ya hemos visto con turtle lo básico (punto, línea, triángulo, cuadrado, rectángulo, círculo, elipse, sectores y arcos), ahora es el momento de pasar a TkInter, la biblioteca gráfica de Python.

TO-DO.

2 Ciberseguridad

2.1 Criptografía

2.1.1 Introducción

La criptografía es una técnica que se utiliza para proteger la información y mantenerla segura. Básicamente, se trata de convertir un mensaje en algo que sólo pueda ser entendido por la persona que tiene la clave para descifrarlo.

Dentro de la criptografía, hay tres conceptos clave que debes conocer: criptología, criptoanálisis y criptosistema. La criptología es la ciencia que estudia la criptografía, mientras que el criptoanálisis es la técnica para tratar de descifrar un mensaje sin tener la clave. Y finalmente, el criptosistema es el conjunto de técnicas y procedimientos que se utilizan para cifrar y descifrar mensajes.

Los elementos de un criptosistema incluyen el mensaje original, que se llama texto claro, y la clave que se utiliza para cifrarlo. El resultado de cifrar el mensaje es el texto cifrado. Para poder descifrar el mensaje cifrado, es necesario tener la clave correcta. Además, los criptosistemas también pueden incluir algoritmos matemáticos y otros procedimientos para hacer el cifrado más seguro.

En resumen, la criptografía es una técnica que se utiliza para proteger la información y mantenerla segura. Para ello, se utilizan técnicas como el cifrado y la utilización de claves, que forman parte de un criptosistema. La criptología estudia estos procesos, mientras que el criptoanálisis trata de romper la seguridad de los criptosistemas sin tener la clave adecuada.

2.1.2 Cifrado de llave simétrica o de una clave

Los cifrados de llave simétrica son un tipo de cifrado que utiliza la misma clave para cifrar y descifrar el mensaje. Esto significa que tanto el emisor como el receptor deben conocer la clave para poder comunicarse de forma segura.

Uno de los cifrados más conocidos de este tipo es el cifrado César. Este cifrado se basa en un desplazamiento de letras en el alfabeto. Por ejemplo, si usamos un desplazamiento de 3 posiciones, la letra “A” se cifraría como “D”, la letra “B” como “E”, y así sucesivamente. De esta manera, el mensaje original se convierte en un mensaje cifrado que parece ininteligible para alguien que no conoce el desplazamiento.

Por ejemplo, si queremos cifrar la palabra “HOLA” con un desplazamiento de 3 posiciones, el resultado sería “KROD”.

Hay muchos otros algoritmos de cifrado de llave simétrica, como el cifrado AES, DES y Blowfish. Estos algoritmos se utilizan en la vida cotidiana para proteger la información que se transmite a través de

internet, como las contraseñas y los datos bancarios.

Por último, el cifrado físico se refiere a la protección física de la información. Esto puede incluir la utilización de cajas fuertes, cerraduras, y otros dispositivos de seguridad para proteger documentos y objetos valiosos. Un ejemplo de cifrado físico es el uso de candados para proteger bicicletas o casilleros.

Los datos biométricos son medidas físicas o comportamentales únicas de una persona, como la huella digital, el iris, la voz o el rostro. Estos datos se pueden utilizar en la criptografía para proporcionar una capa adicional de seguridad en los sistemas de autenticación.

Por ejemplo, en lugar de utilizar una contraseña o una clave de seguridad, un sistema de autenticación biométrica puede utilizar el reconocimiento facial para identificar a una persona. El sistema puede escanear la cara del usuario y compararla con una base de datos de imágenes de caras autorizadas. Si hay una coincidencia, se le permite al usuario el acceso al sistema.

Los datos biométricos también se pueden utilizar para la encriptación de datos. En lugar de utilizar una clave de cifrado tradicional, se puede utilizar una clave generada a partir de los datos biométricos del usuario. Por ejemplo, se puede utilizar la huella dactilar del usuario para generar una clave de cifrado única. Esta clave sólo se puede desbloquear si la huella dactilar del usuario se escanea correctamente.

Sin embargo, es importante tener en cuenta que los datos biométricos pueden ser robados o falsificados. Por lo tanto, es importante que los sistemas de autenticación biométrica estén bien diseñados y sean seguros para evitar el acceso no autorizado. Además, es importante asegurarse de que los datos biométricos se almacenen de forma segura y se protejan contra el acceso no autorizado.

2.1.3 Estenografía y estegoanálisis

La esteganografía es una técnica de ocultación de información que se utiliza para esconder datos dentro de otros datos, como una imagen, un audio o un texto, de tal manera que el mensaje oculto no sea detectado por una persona que no tenga conocimiento de su existencia. Por otro lado, el estegoanálisis es el proceso de detectar y analizar la existencia de un mensaje oculto dentro de un archivo.

Existen varios algoritmos de esteganografía que se utilizan en el mundo real, entre ellos destacan:

- **Esteganografía en imágenes:** en este tipo de esteganografía se oculta información en el interior de una imagen. Los datos se pueden ocultar en los píxeles menos significativos de una imagen, lo que permite que el mensaje oculto no sea detectado a simple vista. Uno de los algoritmos más conocidos es el método de esteganografía LSB (Least Significant Bit), el cual consiste en reemplazar los bits menos significativos de la imagen con los bits del mensaje oculto.

- **Esteganografía en audio:** en este tipo de esteganografía se oculta información en el interior de un archivo de audio. Los datos se pueden ocultar en las frecuencias menos audibles, lo que permite que el mensaje oculto no sea detectado por el oído humano. Uno de los algoritmos más conocidos es el método de esteganografía MP3Stego, el cual consiste en reemplazar las muestras de audio menos significativas con los datos del mensaje oculto.
- **Esteganografía en texto:** en este tipo de esteganografía se oculta información dentro del propio texto de un mensaje. Uno de los algoritmos más conocidos es el método de esteganografía null cipher, el cual consiste en ocultar el mensaje en los espacios en blanco entre las palabras del texto.

En cuanto a los usos de la esteganografía en el mundo real, puede ser utilizada en situaciones en las que se necesita proteger la privacidad de la información, como en el ámbito militar o en la protección de datos sensibles. Por otro lado, también puede ser utilizada con fines malintencionados, como el ocultamiento de mensajes terroristas o para fines de espionaje.

El estegoanálisis se utiliza para detectar la existencia de un mensaje oculto en un archivo. Por ejemplo, puede ser utilizado por las fuerzas de seguridad para descubrir mensajes ocultos en imágenes o archivos de audio que puedan contener información valiosa para la prevención de delitos o actos terroristas.

2.1.4 Criptografía avanzada

La criptografía avanzada es una técnica de protección de la información que utiliza algoritmos y sistemas más complejos y sofisticados que los que se utilizan en la criptografía tradicional. La criptografía avanzada es utilizada para proteger información extremadamente sensible, como información bancaria, datos militares, información de gobiernos y empresas, entre otros.

La criptografía avanzada utiliza métodos matemáticos y computacionales avanzados para proteger la información, haciendo que sea muy difícil, si no imposible, para una persona no autorizada acceder a la información protegida. Los algoritmos utilizados en la criptografía avanzada son altamente seguros y se basan en la complejidad de ciertos problemas matemáticos, como el problema del logaritmo discreto y la factorización de números enteros.

La criptografía avanzada también utiliza técnicas de autenticación, como la firma digital y el intercambio de claves, para garantizar que la información sea auténtica y que solo las personas autorizadas tengan acceso a ella.

En resumen, la criptografía avanzada es una técnica de protección de la información altamente segura y compleja, que utiliza algoritmos y sistemas matemáticos avanzados para garantizar que la información sea protegida y solo accesible por personas autorizadas.

2.1.5 Criptografía de llave asimétrica

La criptografía de llave asimétrica, también conocida como criptografía de clave pública, es un tipo de criptografía que utiliza dos claves diferentes para cifrar y descifrar información. A diferencia de la criptografía de llave simétrica, donde ambas partes comparten la misma clave para cifrar y descifrar información, en la criptografía de llave asimétrica, cada parte tiene su propia clave: una clave pública y una clave privada.

La clave pública se utiliza para cifrar la información, mientras que la clave privada se utiliza para descifrarla. La clave privada es secreta y solo debe ser conocida por la persona que la posee, mientras que la clave pública puede ser compartida con cualquier persona.

La criptografía de llave asimétrica se utiliza en varios contextos, como la firma digital y la seguridad en servidores web. En la firma digital, el emisor de un mensaje utiliza su clave privada para firmar digitalmente el mensaje, lo que garantiza que el mensaje no ha sido modificado y que proviene del emisor legítimo. El receptor del mensaje utiliza la clave pública del emisor para verificar la firma digital y asegurarse de que el mensaje es auténtico.

Veamos un ejemplo: Si Bob quiere mandar un mensaje a Alice utilizando criptografía de llave asimétrica, podría utilizar un algoritmo como RSA o ECC.

Para asegurarse de que sólo Alice, destinataria del mensaje, pueda leerlo, Bob necesitaría la clave pública de Alice. Él utilizaría la clave pública de Alice para cifrar el mensaje antes de enviarlo. La clave privada, que es necesaria para descifrar el mensaje, sólo es conocida por Alice.

Una vez que Alice recibe el mensaje cifrado, ella utilizaría su clave privada para descifrarlo. Para hacer esto, primero debería importar su clave privada en su software de criptografía y luego usarla para descifrar el mensaje cifrado que recibió de Bob.

Es importante mencionar que, para que el sistema de criptografía de llave asimétrica sea seguro, es esencial mantener la clave privada en secreto. Si la clave privada se revela o se pierde, el sistema de seguridad se vería comprometido y el mensaje ya no estaría seguro.

En los servidores web, la criptografía de llave asimétrica se utiliza para proteger la información transmitida entre el servidor y el cliente. Cuando se establece una conexión segura entre un servidor y un cliente, el servidor envía su clave pública al cliente para cifrar la información, de modo que solo el servidor puede descifrarla con su clave privada. Esto evita que un atacante malintencionado pueda interceptar la información y descifrarla, lo que se conoce como un ataque de “man in the middle”.

Cuando visitas un sitio web seguro, como <https://www.iesvirgendelcarmen.com>, tu navegador web (por ejemplo, Firefox) utiliza criptografía de llave asimétrica para establecer una conexión segura con el servidor. Esto se logra mediante el protocolo SSL/TLS (Secure Sockets Layer/Transport Layer Security).

Durante el proceso de conexión, el servidor envía su certificado digital al navegador. Este certificado contiene información que confirma la identidad del servidor y su clave pública. El certificado es emitido por una entidad de certificación confiable (CA), como Verisign o Let's Encrypt, que garantiza la autenticidad del certificado.

El navegador verifica la autenticidad del certificado del servidor y confirma que el nombre de dominio en la URL del sitio web coincide con el nombre de dominio del certificado. Luego, el navegador utiliza la clave pública del servidor para cifrar una sesión de intercambio de claves de cifrado de datos. El servidor utiliza su clave privada para descifrar esta sesión de intercambio de claves.

Una vez que se ha establecido la conexión segura, el navegador y el servidor intercambian claves de cifrado de datos para proteger la privacidad y la integridad de la información transmitida entre ellos. Todo el tráfico entre el navegador y el servidor se cifra con estas claves.

Si un servidor malicioso intenta suplantar la identidad de un sitio web seguro, el navegador mostrará una advertencia de seguridad al usuario, ya que el certificado no se puede verificar o no coincide con el nombre de dominio en la URL. Por lo tanto, es importante verificar que el sitio web seguro que estás visitando esté usando un certificado emitido por una CA confiable y que el nombre de dominio en la URL coincida con el nombre de dominio en el certificado.

2.2 Hacking ético

2.2.1 Introducción

El hacking se refiere al acto de aprovechar vulnerabilidades en sistemas informáticos o redes para obtener acceso no autorizado o realizar modificaciones no autorizadas en los mismos. El hacking ético, por otro lado, es el uso de técnicas de hacking para identificar vulnerabilidades en sistemas o redes con el propósito de mejorar la seguridad y protegerlos contra ataques malintencionados.

Las fases típicas de un proceso de hacking ético son las siguientes:

1. Recopilación de información: El objetivo es obtener información sobre el sistema o red que se va a probar.
2. Escaneo: Se utilizan herramientas de escaneo de vulnerabilidades para identificar puertos abiertos, servicios en ejecución y vulnerabilidades conocidas.
3. Enumeración: Se recopila información adicional sobre los sistemas y servicios identificados en la fase de escaneo.
4. Explotación: Se intenta aprovechar las vulnerabilidades identificadas para obtener acceso no autorizado o realizar modificaciones en el sistema o red.

5. Mantenimiento del acceso: Si se logra obtener acceso, se realiza un mantenimiento del mismo para mantener la persistencia y seguir explorando el sistema o red.
6. Informe: Se documentan las vulnerabilidades encontradas y se informa al dueño del sistema o red para que puedan ser corregidas.

Existen varios tipos de hackers, algunos de los cuales son:

1. White hat (sombrero blanco): Son hackers éticos que utilizan sus habilidades para mejorar la seguridad de los sistemas y redes.
2. Black hat (sombrero negro): Son hackers malintencionados que utilizan sus habilidades para obtener acceso no autorizado a sistemas y redes con fines ilícitos.
3. Grey hat (sombrero gris): Son hackers que no tienen intenciones malintencionadas, pero pueden realizar actividades ilegales o no éticas para identificar vulnerabilidades.
4. Script kiddies: Son personas que utilizan herramientas automatizadas y programas preconfigurados para realizar ataques sin comprender realmente cómo funcionan.
5. Hacktivistas: Son hackers que utilizan sus habilidades para hacer una declaración política o social.

Es importante destacar que el hacking ético es una actividad legítima que se realiza con el permiso del dueño del sistema o red que se está probando. El objetivo es mejorar la seguridad y proteger los sistemas y redes contra ataques malintencionados.

En ciberseguridad, el “red team” y el “blue team” son dos equipos de seguridad que se utilizan en las pruebas de penetración (pen testing) y en los ejercicios de seguridad para mejorar la postura de seguridad de una organización.

El “red team” es un equipo de ataque, compuesto por hackers éticos, que simula ataques cibernéticos reales con el objetivo de poner a prueba la seguridad de la organización. Utilizan técnicas de hacking para encontrar vulnerabilidades en los sistemas y redes, y prueban los controles de seguridad de la organización.

El “blue team” es un equipo de defensa que trabaja para proteger la organización de los ataques simulados del “red team”. Utilizan herramientas de monitoreo y análisis para detectar y responder a los ataques, y mejoran la postura de seguridad de la organización.

El “purple team” es un equipo que combina los objetivos y habilidades de ambos equipos para mejorar la colaboración y la comunicación entre el “red team” y el “blue team”. El objetivo del “purple team” es identificar las debilidades de la organización y mejorar la defensa contra futuros ataques.

La razón por la que se utilizan estos equipos es para mejorar la seguridad de una organización. Los ataques cibernéticos son cada vez más sofisticados y frecuentes, y es necesario asegurarse de que los

sistemas y redes de la organización estén protegidos y ayudan también a que se cumplan con las leyes, regulaciones y estándares de seguridad de organizaciones y gobiernos.

Existen otros tipos de equipos de ciberseguridad, como el equipo de respuesta a incidentes de seguridad (CSIRT), que se centra en la gestión y respuesta a los incidentes de seguridad en la organización, y el equipo de gestión de riesgos de seguridad, que se centra en la evaluación y mitigación de los riesgos de seguridad en la organización.

2.2.2 Técnicas de búsqueda de información: Information gathering.

La fase de recopilación de información o “information gathering” es crucial en el hacking ético y en la ciberseguridad en general. Esta fase implica la recopilación de información sobre el objetivo para poder identificar posibles vulnerabilidades y ataques.

Existen diferentes técnicas y herramientas que se pueden utilizar para recopilar información sobre un objetivo. A continuación, se presentan algunas de las técnicas más comunes:

1. OSINT (Open Source Intelligence): Se trata de la recopilación de información a partir de fuentes públicas de acceso libre, como redes sociales, foros, blogs, noticias, entre otros. El objetivo es obtener información relevante y valiosa sobre el objetivo que pueda ser utilizada en la siguiente fase de análisis.
2. Escaneo de puertos y servicios: Los hackers utilizan esta técnica para identificar los servicios que se ejecutan en el sistema objetivo y para identificar posibles vulnerabilidades en estos servicios. Algunas herramientas populares para esta técnica son Nmap, Masscan y Zmap.
3. Enumeración de DNS: Esta técnica implica la recopilación de información sobre el objetivo a través de la consulta de registros DNS, como registros MX, registros de subdominios, registros de host, entre otros. Algunas herramientas populares para esta técnica son NSLookup, Dig y Fierce.
4. Escaneo de vulnerabilidades: Se trata de la identificación de posibles vulnerabilidades en el sistema objetivo, utilizando herramientas como Nessus, OpenVAS y Nikto.
5. Ingeniería social: Los hackers utilizan esta técnica para obtener información valiosa de los usuarios de la organización objetivo, como contraseñas, información personal y credenciales de inicio de sesión. Algunas técnicas de ingeniería social incluyen la suplantación de identidad, el phishing y la ingeniería social por teléfono.

Es importante tener en cuenta que estas técnicas deben ser utilizadas únicamente con fines éticos y legales, como parte de una prueba de penetración autorizada.

En cuanto a programas, algunas herramientas populares para la fase de recopilación de información incluyen:

- Maltego: una herramienta de OSINT que permite la recopilación y visualización de información en tiempo real.
- Nmap: una herramienta de escaneo de puertos y servicios.
- Metasploit: una plataforma de pruebas de penetración que incluye herramientas de escaneo de vulnerabilidades.
- Social-Engineer Toolkit (SET): una herramienta de ingeniería social que incluye diferentes técnicas de phishing.

Recuerda que estas herramientas deben ser utilizadas con precaución y con fines éticos y legales.

2.2.3 Escaneo: pruebas de PenTesting

El red team ha recopilado previamente información sobre el objetivo y ha identificado posibles vulnerabilidades y, por tanto, ahora es el momento de realizar pruebas de penetración o pentesting para evaluar la seguridad del sistema. El objetivo de estas pruebas es simular un ataque real y determinar si un atacante podría explotar las vulnerabilidades identificadas y acceder a información o recursos críticos.

En general, las pruebas de pentesting se realizan en tres etapas: recolección de información, análisis de vulnerabilidades y explotación. En la primera etapa, el equipo de pentesting utiliza herramientas de búsqueda de información para recopilar información sobre el objetivo. En la segunda etapa, el equipo de pentesting utiliza herramientas de análisis de vulnerabilidades para identificar posibles vulnerabilidades en el sistema. Finalmente, en la tercera etapa, el equipo de pentesting intenta explotar las vulnerabilidades para demostrar que se pueden acceder a recursos o información críticos.

El objetivo de las pruebas de pentesting es simular un ataque real para evaluar la seguridad de un sistema o red. Durante estas pruebas, se intenta explotar las vulnerabilidades encontradas en la fase anterior para obtener acceso no autorizado al sistema o red y, de esta manera, determinar la efectividad de las medidas de seguridad existentes y encontrar posibles debilidades.

Existen diferentes herramientas que pueden ayudar en la realización de pruebas de pentesting, entre ellas se encuentran Nessus y OpenVAS. Estas herramientas realizan un escaneo de vulnerabilidades en el sistema o red y proporcionan un informe detallado de las posibles vulnerabilidades encontradas. También pueden realizar pruebas de autenticación, escaneo de puertos, detección de malware, entre otras funciones.

Por ejemplo, Nessus es una herramienta ampliamente utilizada en el mundo de la ciberseguridad para escanear vulnerabilidades en sistemas y redes. Su funcionamiento se basa en la realización de escaneos de puertos y servicios en busca de posibles vulnerabilidades. Además, cuenta con una amplia base de datos de vulnerabilidades conocidas que le permite identificar posibles problemas de seguridad en

el sistema o red escaneados. Por otro lado, OpenVAS es una herramienta similar a Nessus, pero de código abierto, lo que la hace una opción más accesible para aquellos que buscan una herramienta gratuita.

2.2.4 Vulnerabilidades en sistemas

Una vulnerabilidad en un sistema informático es una debilidad o fallo en el software, hardware o configuración del sistema que puede ser aprovechada por atacantes para comprometer la seguridad del sistema, obtener acceso no autorizado, robar información o causar daño. Las vulnerabilidades pueden ser causadas por errores de programación, configuración incorrecta, falta de parches de seguridad, entre otros factores.

Además de Metasploit Framework, existen otras herramientas de explotación de vulnerabilidades, como ExploitDB, que es una base de datos en línea de exploits y técnicas de hacking; Core Impact, que es una herramienta de pruebas de penetración comercial que permite identificar y explotar vulnerabilidades en sistemas; y CANVAS, otra herramienta de prueba de penetración que permite a los evaluadores de seguridad identificar y explotar vulnerabilidades en aplicaciones y sistemas.

Es importante tener en cuenta que estas herramientas de explotación de vulnerabilidades solo deben ser utilizadas por profesionales de la seguridad debidamente capacitados y autorizados, y en un entorno de prueba controlado. Su uso inapropiado o malicioso puede tener graves consecuencias legales y de seguridad.

2.3 Incidentes de ciberseguridad

2.3.1 Análisis forense

La creciente dependencia de la tecnología en nuestras vidas diarias ha llevado a una mayor cantidad de crímenes y delitos que involucran dispositivos electrónicos, como ordenadores, móviles y tabletas. En respuesta a esta tendencia, ha surgido un campo especializado conocido como análisis forense digital o “digital forensics”, que se centra en la recopilación, preservación, análisis y presentación de evidencia digital en casos judiciales.

El análisis forense digital es una disciplina que implica la aplicación de técnicas y métodos de investigación científica para examinar datos electrónicos y determinar su autenticidad, integridad y relevancia para una investigación o proceso judicial. Las investigaciones forenses digitales pueden ser solicitadas por agencias gubernamentales, empresas, abogados y particulares para una variedad de propósitos, incluyendo la investigación de delitos informáticos, la recuperación de datos, la identificación de amenazas a la seguridad, la resolución de disputas y la obtención de pruebas en casos civiles o penales.

El proceso de análisis forense digital involucra varias etapas, como la identificación de la evidencia, la recolección o adquisición y preservación de la evidencia, el análisis y la interpretación de la evidencia, y la presentación de los hallazgos. Durante la etapa de identificación, se busca identificar la evidencia digital relevante para el caso, y se deben tomar medidas para preservar la evidencia y evitar su alteración o destrucción. La etapa de recolección y preservación implica la recopilación de la evidencia digital y su almacenamiento en un entorno seguro y controlado. En la etapa de análisis, se utiliza una variedad de técnicas, herramientas y métodos para analizar la evidencia y extraer información relevante. Finalmente, en la etapa de presentación, los hallazgos del análisis se documentan y se presentan en un informe, que se utiliza como evidencia en los procedimientos legales.

Entre las técnicas y herramientas utilizadas en el análisis forense digital, se encuentran el análisis de registro, el análisis de tráfico de red, la recuperación de datos borrados, la recuperación de contraseñas, entre otras. Las herramientas de software también son una parte esencial del análisis forense digital, y existen numerosas herramientas que pueden ayudar a los investigadores en cada etapa del proceso.

En resumen, el análisis forense digital es una técnica esencial en la lucha contra los delitos informáticos, y su aplicación puede ayudar a las organizaciones a identificar y resolver incidentes de seguridad, así como a investigar delitos informáticos y llevar a los autores ante la justicia. Es importante que los profesionales de la seguridad informática y los investigadores estén capacitados en las técnicas y herramientas de análisis forense digital para poder enfrentar los desafíos de un mundo cada vez más digitalizado.

A continuación, proporcionamos algunos ejemplos de herramientas que se utilizan comúnmente en cada una de las etapas del análisis forense digital:

1. Adquisición de datos:

- dd (para copiar datos bit a bit)
- FTK Imager (para adquirir imágenes de disco)
- EnCase Forensic Imager (para adquirir imágenes de disco)
- Helix3 Pro (para crear imágenes de disco)

2. Preservación de datos:

- FTK Imager (para crear imágenes forenses)
- Encase Forensic (para proteger los datos originales)
- WinHex (para preservar los datos de forma segura)

3. Análisis de datos:

- Autopsy (para analizar imágenes de disco)

- EnCase Forensic (para análisis de archivos y recuperación de datos)
- SANS SIFT Workstation (para análisis de redes)
- The Sleuth Kit (para análisis de archivos)

4. Presentación de resultados:

- Xplico (para reconstruir la comunicación en red)
- Wireshark (para capturar y analizar tráfico de red)
- Oxygen Forensic Detective (para análisis de dispositivos móviles)
- FTK (para presentar los resultados del análisis)

Es importante tener en cuenta que estas son solo algunas de las herramientas disponibles y que cada analista forense puede tener sus preferencias en cuanto a las herramientas que utiliza. Además, es importante mencionar que el uso de estas herramientas debe hacerse de manera legal y ética.

2.3.2 Ciberdelitos

Un ciberdelito es un delito que se comete en el ámbito digital, utilizando tecnologías de la información y la comunicación (TIC). También se le conoce como delito informático o delito cibernético. Los ciberdelitos pueden incluir una amplia variedad de actividades ilegales, como el acceso no autorizado a sistemas informáticos, el robo de información personal o financiera, la difusión de virus informáticos, el fraude en línea, el acoso en línea, el ciberespionaje, la extorsión y el phishing, entre otros.

Puede ser un delito informático en el que se utiliza una computadora o una red de computadoras para llevar a cabo actividades ilegales, como el acceso no autorizado a sistemas o el robo de información. También puede ser un delito que se comete utilizando internet como medio, como el acoso en línea o la difusión de material ilegal a través de la red.

Los ciberdelitos pueden ser llevados a cabo por individuos, grupos organizados o incluso por gobiernos. Algunos ejemplos de ciberdelitos incluyen:

- Hacking: El acceso no autorizado a sistemas informáticos con el fin de robar información o causar daños.
- Phishing: El uso de correos electrónicos falsos o sitios web falsos para engañar a las personas y obtener información confidencial, como contraseñas o números de tarjetas de crédito.
- Malware: La distribución de software malicioso que puede ser utilizado para espiar o dañar sistemas informáticos.

- Fraude en línea: El uso de internet para llevar a cabo estafas, como el fraude de inversión o la venta de productos falsos.
- Ciberacoso: La utilización de internet para hostigar, intimidar o amenazar a otras personas.

Los ciberdelitos son un problema creciente en todo el mundo, ya que la tecnología se ha vuelto cada vez más omnipresente en nuestras vidas cotidianas. Los delincuentes pueden utilizar herramientas y técnicas sofisticadas para cometer ciberdelitos y, a menudo, pueden hacerlo desde cualquier lugar del mundo, lo que dificulta su detección y enjuiciamiento.

Los ciberdelitos pueden tener graves consecuencias para las víctimas, incluyendo la pérdida de datos personales y financieros, la interrupción de los sistemas informáticos y la exposición a actividades ilegales. También pueden tener consecuencias más amplias, como la afectación de la economía o la seguridad nacional.

Para prevenir los ciberdelitos es importante tomar medidas de seguridad como mantener actualizados los sistemas y software, utilizar contraseñas seguras, evitar hacer clic en enlaces sospechosos y estar alerta a posibles intentos de phishing. También es importante contar con herramientas de seguridad como software antivirus, firewalls y herramientas de detección de intrusiones. En caso de sufrir un ciberdelito, es importante reportarlo a las autoridades y tomar medidas para mitigar los daños. # Big Data

En este tema aprenderemos:

- Big data. Características. Volumen de datos.
- Visualización, transporte y almacenaje de los datos.
- Recogida, análisis y generación de datos.
- Simulación de fenómenos naturales y sociales.
- Descripción del modelo.
- Identificación de agentes.
- Implementación del modelo mediante un software específico, o mediante programación.

El término “big data” se refiere a la enorme cantidad de datos que se generan a diario en diversas fuentes como sensores, redes sociales, transacciones financieras, registros médicos, entre otros. Las características principales de los datos que conforman el big data son el volumen, la variedad y la velocidad.

El volumen de los datos del big data es enorme y supera con creces la capacidad de los sistemas tradicionales de almacenamiento y procesamiento de datos. Además, los datos del big data suelen estar en constante crecimiento y actualización, lo que hace que el volumen sea cada vez mayor.

La variedad de los datos del big data hace referencia a la diversidad de formatos, fuentes y tipos de datos que se generan. Estos datos pueden ser estructurados (como bases de datos y hojas de cálculo) o no estructurados (como textos, imágenes y videos).

La velocidad de los datos del big data se refiere a la rapidez con que se generan y se deben procesar. En algunos casos, es necesario analizar y actuar sobre los datos en tiempo real para poder tomar decisiones rápidas y eficaces.

Para poder trabajar con big data es necesario contar con herramientas y tecnologías que permitan visualizar, transportar y almacenar los datos de manera eficiente. Algunas de estas herramientas son Hadoop, Spark, Cassandra y MongoDB.

La recogida, análisis y generación de datos son procesos fundamentales en el trabajo con big data. La recogida de datos se realiza a través de sensores, redes sociales, registros de transacciones, entre otros. El análisis de datos permite obtener información útil y relevante a partir de los datos del big data. La generación de datos puede ser realizada mediante la simulación de fenómenos naturales y sociales.

Para la simulación de fenómenos naturales y sociales es necesario crear un modelo que permita representar y simular el comportamiento de los elementos que conforman el fenómeno. Este modelo puede ser descrito mediante una serie de ecuaciones y algoritmos.

La identificación de agentes es un proceso importante en la creación de modelos de simulación. Los agentes son los elementos individuales que interactúan dentro del fenómeno que se está simulando. Estos agentes pueden ser personas, animales, objetos, entre otros.

La implementación del modelo de simulación puede ser realizada mediante un software específico de simulación o mediante programación. En ambos casos, es necesario definir las variables y parámetros del modelo y establecer los límites y condiciones para la simulación.

2.4 Conceptos previos

El término “big data” se refiere a la enorme cantidad de datos que se generan a diario en diversas fuentes como sensores, redes sociales, transacciones financieras, registros médicos, entre otros. Las características principales de los datos que conforman el big data son el volumen, la variedad y la velocidad.

El volumen de los datos del big data es enorme y supera con creces la capacidad de los sistemas tradicionales de almacenamiento y procesamiento de datos. Además, los datos del big data suelen estar en constante crecimiento y actualización, lo que hace que el volumen sea cada vez mayor.

La variedad de los datos del big data hace referencia a la diversidad de formatos, fuentes y tipos de datos que se generan. Estos datos pueden ser estructurados (como bases de datos y hojas de cálculo) o no estructurados (como textos, imágenes y videos).

La velocidad de los datos del big data se refiere a la rapidez con que se generan y se deben procesar. En algunos casos, es necesario analizar y actuar sobre los datos en tiempo real para poder tomar decisiones rápidas y eficaces.

Para poder trabajar con big data es necesario contar con herramientas y tecnologías que permitan visualizar, transportar y almacenar los datos de manera eficiente. Algunas de estas herramientas son Hadoop, Spark, Cassandra y MongoDB.

La recogida, análisis y generación de datos son procesos fundamentales en el trabajo con big data. La recogida de datos se realiza a través de sensores, redes sociales, registros de transacciones, entre otros. El análisis de datos permite obtener información útil y relevante a partir de los datos del big data. La generación de datos puede ser realizada mediante la simulación de fenómenos naturales y sociales.

Para la simulación de fenómenos naturales y sociales es necesario crear un modelo que permita representar y simular el comportamiento de los elementos que conforman el fenómeno. Este modelo puede ser descrito mediante una serie de ecuaciones y algoritmos.

La *identificación de agentes* es un proceso importante en la creación de modelos de simulación. Los agentes son los elementos individuales que interactúan dentro del fenómeno que se está simulando. Estos agentes pueden ser personas, animales, objetos, entre otros.

La *implementación del modelo de simulación* puede ser realizada mediante un software específico de simulación o mediante programación. En ambos casos, es necesario definir las variables y parámetros del modelo y establecer los límites y condiciones para la simulación.

2.5 Big data. Características. Volumen de datos.

Como ya comentamos anteriormente, el término “big data” se refiere a la enorme cantidad de datos que se generan a diario en diversas fuentes como sensores, redes sociales, transacciones financieras, registros médicos, entre otros. Pero antes de adentrarnos en el tema es necesario hablar de **las 7 “V” del Big Data, qué caracteriza al Big Data**:

1. *Volumen*: hace referencia a la enorme cantidad de datos generados y almacenados en diferentes fuentes de información.
2. *Velocidad*: se refiere a la velocidad a la que se generan los datos y a la necesidad de procesarlos en tiempo real.
3. *Variedad de los datos*: se refiere a la diversidad de tipos de datos y su complejidad, que van desde datos estructurados como bases de datos, hasta datos no estructurados como imágenes, vídeos, audios y texto.
4. *Veracidad de los datos*: se refiere a la confiabilidad y precisión de los datos. Es importante asegurar que los datos utilizados sean precisos y confiables para tomar decisiones importantes.
5. *Viabilidad*: se refiere a la capacidad de las empresas y organizaciones para gestionar, almacenar y procesar grandes cantidades de datos.
6. *Visualización de los datos*: se refiere a la habilidad de transformar datos complejos en información visual fácil de entender y analizar.

7. *Valor de los datos*: se refiere a la capacidad de los datos para generar valor y proporcionar una ventaja competitiva a las empresas y organizaciones que los utilizan de manera efectiva.

Un proyecto de Big Data típicamente consta de varias **capas tecnológicas** que trabajan juntas para procesar y analizar grandes cantidades de datos. A continuación, se describen algunas de las capas tecnológicas comunes de un proyecto de Big Data:

1. *Ingesta de datos*: Esta capa se encarga de la recolección de datos de diversas fuentes, como sensores, redes sociales, bases de datos, etc., y su transferencia a la siguiente capa para su procesamiento.
2. *Almacenamiento de datos*: Esta capa almacena los datos recolectados en una infraestructura escalable y distribuida, como Hadoop Distributed File System (HDFS), Apache Cassandra, Amazon S3, entre otros.
3. *Procesamiento de datos*: Esta capa se encarga de procesar y analizar los datos utilizando tecnologías como Apache Spark, MapReduce, Hadoop, entre otras.
4. *Análisis de datos*: Esta capa se utiliza para el análisis y la exploración de los datos, utilizando herramientas de minería de datos, aprendizaje automático y estadísticas.
5. *Visualización de datos*: Esta capa permite la visualización y el análisis interactivo de los datos procesados y analizados, utilizando herramientas de visualización de datos como Tableau, QlikView, entre otras.
6. *Integración con sistemas existentes*: Esta capa se encarga de integrar los resultados del análisis de datos en sistemas existentes, como aplicaciones de negocios, CRM, sistemas de gestión de la cadena de suministro, entre otros.
7. *Seguridad y gestión de datos*: Esta capa garantiza la seguridad de los datos y la gestión de los permisos de acceso a los datos, cumpliendo con las regulaciones y normativas existentes.

2.6 Visualización, transporte y almacenaje de los datos.

En un proyecto de Big Data, la visualización, transporte y almacenamiento de los datos son tres aspectos críticos que deben ser cuidadosamente considerados.

- *Visualización de datos*: la visualización de datos es una parte crucial del análisis de Big Data. Permite presentar los datos de forma clara y fácilmente comprensible para que los usuarios puedan tomar decisiones informadas basadas en ellos. Las herramientas de visualización de datos se utilizan para crear gráficos, tablas, diagramas y otros tipos de representaciones visuales de los datos.
- *Transporte de datos*: para llevar los datos de un lugar a otro, se necesitan herramientas de transporte de datos. En un entorno de Big Data, esto puede implicar el movimiento de grandes cantidades de datos de una ubicación a otra. Las herramientas de transporte de datos incluyen

soluciones de ETL (extracción, transformación y carga) y de streaming, que se utilizan para mover datos en tiempo real.

- *Almacenamiento de datos*: el almacenamiento de datos es otro aspecto crítico del Big Data. La capacidad de almacenar grandes cantidades de datos de forma eficiente y segura es esencial. Las tecnologías de almacenamiento de datos incluyen bases de datos NoSQL, almacenamiento de objetos, almacenamiento en la nube y almacenamiento en Hadoop. Cada una de estas tecnologías tiene ventajas y desventajas, y la elección de una u otra depende de las necesidades específicas del proyecto.

2.7 Recogida, análisis y generación de datos.

En primer lugar, la recogida de datos implica la identificación de las fuentes de datos relevantes para el proyecto y la captura de dichos datos de manera eficiente y en tiempo real. Puede involucrar la integración de datos de múltiples fuentes, tanto internas como externas a la organización.

Una vez recopilados los datos, se procede al análisis de los mismos. Esto implica la limpieza, procesamiento y transformación de los datos para que sean útiles para el análisis y la toma de decisiones. Este proceso puede incluir la identificación y eliminación de datos duplicados o incompletos, la normalización de los datos y la creación de modelos de datos para facilitar el análisis.

Finalmente, la generación de datos implica el uso de técnicas de análisis de datos para extraer información valiosa y tomar decisiones informadas. Esto puede implicar la creación de modelos de predicción, el análisis de tendencias y patrones, y la identificación de insights ocultos en los datos.

Algunos ejemplos concretos de productos comerciales y libres que se utilizan en la recopilación, análisis y generación de datos en Big Data:

- Recogida de datos:
 - Producto comercial: Google Analytics es una herramienta popular para recopilar datos de sitios web y aplicaciones móviles. Proporciona información sobre el comportamiento del usuario, el rendimiento del sitio web, la adquisición de tráfico y mucho más.
 - Producto libre: Apache NiFi es una plataforma de integración de datos de código abierto que permite recopilar datos de diversas fuentes, como sensores, bases de datos, archivos y más. Facilita la transferencia de datos en tiempo real a través de flujos de datos seguros y escalables.
- Análisis de datos:
 - Producto comercial: Tableau es una herramienta de visualización de datos que permite a los usuarios crear informes y paneles interactivos. Ayuda a los analistas de datos a identificar patrones, tendencias y oportunidades de negocio a partir de grandes conjuntos de datos.

- Producto libre: Apache Spark es un motor de procesamiento de datos de código abierto que proporciona un análisis rápido de grandes conjuntos de datos. Proporciona una API unificada para trabajar con diferentes tipos de datos, como procesamiento de gráficos, aprendizaje automático y procesamiento de flujos.
- Generación de datos:
 - Producto comercial: Hootsuite Insights es una plataforma de escucha social que recopila datos de redes sociales y los utiliza para crear informes personalizados. Los usuarios pueden monitorear el sentimiento de la marca, identificar tendencias y realizar un seguimiento de la competencia.
 - Producto libre: Faker es una biblioteca de generación de datos de código abierto que crea datos falsos para pruebas y simulaciones. Se puede utilizar para generar datos de prueba para aplicaciones web y móviles, y se puede personalizar para crear datos que se ajusten a ciertos requisitos.

2.8 Simulación de fenómenos naturales y sociales

La simulación de fenómenos naturales y sociales en Big Data se refiere al uso de técnicas computacionales para recrear y modelar eventos que ocurren en la naturaleza o en la sociedad, con el objetivo de comprenderlos mejor y predecir su comportamiento futuro. Esta técnica se aplica en una amplia variedad de campos, desde la meteorología hasta la economía.

Un ejemplo concreto de aplicación de la simulación de fenómenos naturales en Big Data es el modelado del clima. La Agencia Meteorológica de Japón utiliza el superordenador más potente del mundo, llamado “Fugaku”, para simular el clima y predecir eventos como tifones, tsunamis y lluvias torrenciales. Este modelo utiliza una gran cantidad de datos históricos y en tiempo real, incluyendo datos de satélite, estaciones meteorológicas y boyas oceánicas, para predecir cómo el clima evolucionará en el futuro.

Otro ejemplo de simulación de fenómenos sociales en Big Data es el modelado de la propagación de enfermedades. Durante la pandemia de COVID-19, muchos países utilizaron modelos de simulación para predecir cómo se propagaría el virus y cómo podrían mitigarse sus efectos. Por ejemplo, la Universidad de Virginia desarrolló un modelo que utiliza datos de ubicación de teléfonos móviles para predecir la propagación del virus y recomendar medidas preventivas.

En ambos casos, la simulación de fenómenos naturales y sociales en Big Data requiere una gran cantidad de datos, así como el uso de técnicas avanzadas de análisis y modelado. Sin embargo, los beneficios potenciales son significativos, ya que pueden ayudar a prevenir desastres naturales y brotes de enfermedades, así como a mejorar la planificación y la toma de decisiones en una amplia variedad de campos.

2.9 Descripción del modelo

La descripción del modelo de simulación en big data se refiere a la documentación detallada de cómo se ha diseñado y construido el modelo de simulación. Esta descripción es importante para garantizar que el modelo pueda ser replicado, modificado y mejorado en el futuro.

La descripción del modelo debe incluir información sobre el propósito del modelo, el fenómeno que se está simulando, los datos utilizados para construir el modelo, los algoritmos y técnicas de análisis utilizados, y los resultados de la simulación. Además, se debe describir cómo se ha implementado el modelo en una plataforma de big data, incluyendo el hardware y software utilizados, y los detalles técnicos de la implementación.

Un ejemplo de modelo de simulación en big data es el sistema de simulación de tráfico desarrollado por la ciudad de Los Ángeles. Este modelo utiliza datos de sensores de tráfico, cámaras y GPS de vehículos para simular el flujo de tráfico en la ciudad. La descripción del modelo incluye detalles sobre cómo se recopilan y procesan los datos, cómo se construye el modelo y cómo se implementa en una plataforma de big data. Con este modelo, la ciudad de Los Ángeles puede predecir el tráfico en tiempo real y tomar medidas para reducir la congestión en las carreteras.

2.10 Identificación de agentes

La identificación de agentes en Big Data se refiere a la identificación de individuos, entidades o sistemas que interactúan dentro de un conjunto de datos para la simulación de eventos. Los agentes pueden ser personas, animales, máquinas o sistemas, y se les puede asignar una serie de características, como edad, género, ubicación geográfica, comportamiento, entre otras.

La identificación de agentes es fundamental para la simulación de fenómenos en Big Data, ya que permite crear modelos más precisos que representan de manera más realista la interacción entre los diferentes elementos que conforman el sistema. Por ejemplo, en una simulación de tráfico urbano, los agentes pueden ser identificados como vehículos, conductores, peatones, semáforos, entre otros. Al asignar a cada agente características específicas, como velocidad, capacidad de reacción y preferencias de ruta, se pueden simular diferentes escenarios y evaluar su impacto en el tráfico.

En resumen, la identificación de agentes en Big Data es un proceso clave en la simulación de fenómenos complejos, ya que permite crear modelos más precisos y realistas, lo que puede ser utilizado en diferentes áreas, como la ingeniería, la gestión de recursos naturales, la planificación urbana, la medicina y la economía, entre otras.

2.11 Implementación del modelo mediante un software específico, o mediante programación

La implementación de un modelo de simulación en big data involucra el uso de tecnologías y herramientas específicas para procesar grandes cantidades de datos y realizar simulaciones en tiempo real. La idea es capturar la información relevante de los datos y utilizarla para generar modelos de simulación precisos y escalables.

Para implementar un modelo de simulación en big data, es necesario seguir los siguientes pasos:

1. **Recopilar los datos relevantes:** La primera etapa consiste en recopilar y seleccionar los datos relevantes para la simulación. Estos datos pueden provenir de diversas fuentes, como sensores, redes sociales, dispositivos IoT, entre otros.
2. **Procesamiento de datos:** El siguiente paso es procesar los datos utilizando técnicas de big data, como el procesamiento distribuido, el procesamiento en tiempo real y el procesamiento de lenguaje natural. El objetivo es limpiar, integrar y estructurar los datos de manera que sean utilizables para el modelo de simulación.
3. **Selección del modelo de simulación:** Una vez que los datos están listos, es necesario seleccionar el modelo de simulación más adecuado. Por ejemplo, si se trata de una simulación de tráfico, se puede utilizar un modelo basado en agentes.
4. **Implementación del modelo de simulación:** El siguiente paso es implementar el modelo de simulación utilizando herramientas y tecnologías de big data, como Apache Spark, Hadoop, Cassandra, entre otros.
5. **Validación y ajuste del modelo:** Una vez implementado el modelo, se debe validar y ajustar su precisión utilizando diferentes técnicas, como la validación cruzada y la comparación de resultados con datos históricos.

Algunos ejemplos de aplicaciones de simulación de fenómenos naturales y sociales en big data son:

- **Simulación de tráfico:** Los modelos de simulación de tráfico en big data pueden ayudar a predecir y optimizar el tráfico en tiempo real. Un ejemplo de producto comercial para esto es el software PTV Vissim.
- **Simulación de eventos deportivos:** Los modelos de simulación en big data pueden ayudar a predecir los resultados de eventos deportivos y analizar la estrategia de los equipos. Un ejemplo de producto comercial para esto es el software SAS Sports Analytics.
- **Simulación de desastres naturales:** Los modelos de simulación de desastres naturales en big data pueden ayudar a predecir y planificar la respuesta a eventos como terremotos y huracanes. Un ejemplo de software libre para esto es el proyecto HAZUS. # Inteligencia Artificial

En este tema aprenderemos:

1. Definición. Historia. El test de Turing.
2. Aplicaciones. Impacto.
3. Ética y responsabilidad social (transparencia y discriminación algorítmica).
4. Beneficios y posibles riesgos.
5. Agentes inteligentes simples.
6. Análisis y clasificación supervisada basada en técnicas de aprendizaje automático: reconocimiento de habla; reconocimiento de imágenes; y reconocimiento de texto.
7. Generación de imágenes y/o música basado en técnicas de aprendizaje automático: mezcla inteligente de dos imágenes; generación de música; traducción y realidad aumentada.

2.12 Definición. Historia. El test de Turing

2.13 Aplicaciones. Impacto

2.14 Ética y responsabilidad social (transparencia y discriminación algorítmica)

2.15 Beneficios y posibles riesgos

2.16 Agentes inteligentes simples

2.17 Análisis y clasificación supervisada basada en técnicas de aprendizaje automático: reconocimiento de habla; reconocimiento de imágenes; y reconocimiento de texto

2.18 Generación de imágenes y/o música basado en técnicas de aprendizaje automático: mezcla inteligente de dos imágenes; generación de música; traducción y realidad aumentada