INDICE

INTRODUCCIÓN A PYTHON
ESTACION 1

1) Creación de archivos

(a) Como nombrar un archivo.

Es recomendado que los nombres de los archivos

- Sean escritos en minúscula
- No contengan espacios (en su lugar podemos usar guiones bajos para separar las palabras)
- Sean representativos de su contenido(esto significa, que sean lo mas descriptivos posibles).

Ejemplos:
Si quiero ordenar mis archivos por número de ejercicios: ejercicio_1.py, ejercicio_2.py.... o ej1.py, ej2.py, etc.
Por nombre de enunciado: factorial.py, max_lista.py, valor_medio.py, calculadora.py

(b) La extensión de un archivo.

También llamado “ampliación del nombre del archivo”, “ampliación de archivo” o “sufijo de archivo”; la *extension* es la última parte del nombre de un archivo. Este componente se separa del nombre real del archivo con un punto y está compuesto normalmente por tres o cuatro letras. Al guardar un archivo con la extension “.py” le estamos indicando a la maquina que el contenido de nuestro archivo es de lenguaje Python.

2) Python. Primeros pasos.

- Comentarios.

Son pequeñas notas o recordatorios, que sirven para **explicar algo**, para que otra persona pueda entender el código sin tener que leerlo todo, o simplemente para anotar algo que puedas querer leer más adelante.

Hay dos tipos de comentarios en Python: **comentarios de una línea,** donde utilizamos el simbolo “#” al inicio de la linea a comentar y los **comentarios de múltiples líneas**, donde utilizamos tres comillas simples “ ''' ” al principio y final de las lineas a comentar. Copia y pega el siguiente ejemplo en tu archivo, y veras que su presencia no altera el correcto desempeño de tu programa.

Ej:

```
# Soy un comentario de una línea
'''
Soy
```

un
comentario
de
varias
líneas
'''

- Variables

Es un nombre que se refiere a un objeto que reside en la memoria. En python a ese nombre se le llama identificador.

Cada identificador debe tener un nombre único. A la hora de elegirlo debemos seguir las siguientes reglas:

- No se pueden usar símbolos como !, @, #, $, %, etc.
- No deben tener espacios
- No pueden usarse como identificadores, las palabras reservadas de Python.
- Los nombres deben ser lo mas descriptibles posibles
- Los nombres que comienzan con guión bajo (simple_ o doble__) se reservan para variables con significado especial(que veremos mas adelante)

¿Como declaro una variable? Detallando su nombre, con el operador de asignacion "=", y su contenido.

Su estructura se detalla asì

nombre_variable = contenido

ej:

edad_usuario = 25
nombre_usuario = "Juan"

Resignación de valores:

Para modificar el valor de una variable en Python, basta con asignarle un nuevo valor en cualquier momento y lugar después de la definición.

Ej:
a = 13
print(a)

a = "roberto"
print(a)

En el ejercicio anterior, vemos como el programa inicia declarando la variable <a> que contiene el numero entero 13.

Llamamos a la funcion print, e imprimimos <a>.

Luego le reasignamos el valor “roberto.
Volvemos a llamar a la funcion print e imprimos <a>, que ya no guarda el numero entero 13, y ahora guarda la cadena de string “roberto”.

- Tipos de datos

En Python todo es un objeto. Sin embargo, su tratamiento puede variar dependiendo de las necesidades. Los 4 tipos de datos que encontramos en Python y en general en otros lenguajes de programación son:

Tipo de dato	en Python	Ejemplo
Integer(numeros enteros)	int	0 1 -100
Float (numeros decimales)	float	1.69 1.33
String (cadena de caracteres)	str	“Pepita” “pepita@gmail.com” “25r”
Boolean (booleanos, indican dos situaciones: verdadero o falso)	bool	True False

- Datos de entrada y salida

➔ SALIDA: función print()

La "salida" de un programa son los datos que el programa proporciona al exterior.
Probablemente ya tuviste un primer acercamiento a esta función, cuando instalamos el software necesario y ejecutamos nuestro

print(“Hola mundo!!”)

para corroborar que todo estuviese funcionando correctamente.
La funcion print, como lo dice su nombre, es una función que se encarga de imprimir informacion en nuestra terminal. Utilizamos paréntesis para indicarle a la función que es lo que debemos imprimir. Podemos ayudarnos con un “**n**” para crear saltos de linea en nuestras salidas. Mira el siguiente ejemplo.

print(”hola **n**pepe **n**como estas?”)

En ocasiones necesitaremos imprimir datos unidos a otros por defecto. En programacion le llamamos ***concatenar*** a juntar o pegar dos strings o cadenas de caracteres, en un solo mensaje. En

python podemos concatenar con dos operadores, la coma y el simbolo de suma. Mira los siguientes ejemplos:

```
print( '8' + '5' )
print('Estudiantes de Control ', 'y estudiantes de programación')
```

➔ Entrada: funcion input()

La "entrada" de un programa son los datos que llegan al programa desde el exterior.

Probablemente, un programa necesite interactuar con datos que le aporte el usuario final, como por ejemplo el nombre de un jugador.

Con la funcion input, podemos capturar datos que ingrese el usuario, y guardarlos en una variable para posteriormente trabajar con ellos. Observa el siguiente ejemplo:

```
nombre = input("¿Cómo se llama?: ")
print("Me alegro de conocerlo ” , nombre)
```

El programa comienza declarando la variable <nombre>, donde se guardara un dato que ingrese el usuario. ¿Como le pedirá el dato al usuario? Mostrandole el mensaje `"¿Cómo se llama?: " y aguardando a que el usuario ingrese el dato por teclado y presione ENTER.`

Posteriormente imprimiremos ese dato con un print, que concatene ese dato con la cadena “Me alegro de conocerlo ”

Por defecto, la funcion input recibe strings, por lo que si preveemos que el usuario debe ingresar un dato de tipo entero, o decimal, debemos indicarlo en su declaracion.Mira el siguiente ejemplo:

```
cantidad_tomates = int(input(“Indique cuantos kgs de tomates necesita: ”))
precio_por_tomate =  float(input(“Indique el valor del kg: ”))
print(“Usted compro ” , cantidad_tomates , “ kilos de tomates y pago por cada kg ” , precio_por_tomate , “ pesos”)
```

- Operadores

(a) Operadores aritméticos

Un operador aritmético toma dos operandos como entrada, realiza un cálculo y devuelve un resultado.

Considera la expresión, **"a = 2 + 3"**. Aquí, `2` y `3` son los *operandos* y `+` es el *operador aritmético*. El resultado de la operación se almacena en la variable `a`.

Operador	Acción	Ejemplo
+	Suma	2 + 5 (daría como resultado 7)
-	Resta	7 - 5 (daría como resultado 2)
*	Multiplicación	2 * 2 (daría como resultado 4)
**	Potencia	2 ** 3 (daría como resultado 8)
/	División	5 / 2 (daría como resultado 2.5)
//	División con resultado entero	5 // 2 (daría como resultado)
%	Modulo	5 % 2 (daría como resultado 1)

Nota: Para obtener el resultado en tipo flotante, uno de los operandos también debe ser de tipo flotante.

(b) Operadores relacionales

Un operador relacional se emplea para comparar y establecer la relación entre ellos. Devuelve un valor booleano (true o false) basado en la condición.

Operador	Acción	Ejemplo
<	Devuelve True si el operando de la izquierda es ***menor que*** el de la derecha	7 < 5 Devuelve False
>	Devuelve True si el operando de la izquierda es ***mayor que*** el de la derecha	7 > 5 Devuelve True
==	Devuelve True si el operando de la izquierda es ***igual que*** el de la derecha	7 == 5 Devuelve False
!=	Devuelve True si el operando de la izquierda es ***distinto que*** el de la derecha	7 != 5 Devuelve True
>=	Devuelve True si el operando de la izquierda es ***mayor o igual que*** el de la derecha	7 >= 5 Devuelve True
<=	Devuelve True si el operando	7 <= 5

	de la izquierda es ***menor o igual que*** el de la derecha	Devuelve False

(c) Operadores de asignación

Operador	Descripción
=	Asignación. a = 5. El valor 5 es asignado a la variable a
+=	a += 5 es equivalente a a = a + 5
-=	a -= 5 es equivalente a a = a - 5
*=	a *= 3 es equivalente a a = a * 3
**=	a **= 3 es equivalente a a = a ** 3
/=	a /= 3 es equivalente a a = a / 3
//=	a //= 3 es equivalente a a = a // 3
%=	a %= 3 es equivalente a a = a % 3

(d) Operadores lógicos

Operador	Acción
and	Devuelve True si ambos operandos son True
or	Devuelve True si al menos uno de los operandos es True
not	Devuelve True si alguno de los operandos es false

(e) Operadores de pertenencia

Un operador de pertenencia se emplea para identificar pertenencia en alguna secuencia (listas, strings, tuplas).

Operador	Acción

in	Devuelve True si el valor identificado se encuentra en la secuencia
not in	Devuelve True si el valor identificado no se encuentra en la secuencia

(f) Operadores de identidad

Un operador de identidad se emplea para comprobar si dos variables emplean la misma ubicación en memoria.

Operador	Acción
is	Devuelve True si los operandos se refieren al mismo objeto.
is not	Devuelve True si los operandos no se refieren al mismo objeto.

- Estructuras de control

(a) Delimitadores en una estructura de control.

Hasta ahora nuestros algoritmos han consistido en simples secuencias de instrucciones unas después de otra. Pero en nuestros programas existen tareas más complejas que no pueden ser resueltas así, quizás necesitamos repetir una misma instrucción, realizar acciones diferentes en función del valor de una expresión, etc. Para esto existen las estructuras de control.

En una estructura de control, los únicos delimitadores existentes son los dos puntos (:) que funciona como un “entonces” y la **indentación** del código. Indentar es indicar mediante un TAB(la sangría de los textos), que la linea de código que lo posee, pertenece a un bloque de código que se ejecutará como parte de una estructura de control.

- Condicionales

Estas estructuras de control son de gran utilidad para cuando el algoritmo a desarrollar requiera una descripción más complicada que una lista sencilla de instrucciones. Este es el caso cuando existe un número de posibles alternativas que resultan de la evaluación de una determinada condición. Este tipo de estructuras son utilizadas para tomar decisiones lógicas, es por esto que también se denominan estructuras de decisión o selectivas.
En estas estructuras, se realiza una evaluación de una condición y de acuerdo al resultado, el algoritmo realiza una determinada acción. Las condiciones son especificadas utilizando expresiones lógicas.

Los condicionales pueden ser:

- Simples: if (si)
- Dobles: if – else (si- si no)

- Múltiples: if-elif-else (si – si – si no)

✔ IF:

Esta estructura lleva a cabo una acción siempre y cuando se cumpla una determinada condición .

La selección IF evalúa la condición y luego:

- Si la condición es verdadera, ejecuta el bloque de acciones que se encuentren indentadas dentro de la condicion.
- Si la condición es falsa, no ejecuta nada y continua con el resto del programa.

Su estructura se lee de la siguiente manera:

Si <condicion> es verdadera entonces
    se ejecutara esta accion
    y esta otra
    ...

Ej:

```
mi_edad = 55

if mi_edad == 55:
    print(“ok boomer”)

print(“bai bai”)
```

Este programa comienza declarando la variable <mi_edad> a la que se le asigna el dato de tipo entero 55.

El programa tiene una condicional simple(if), que analiza si el contenido de la variable <mi_edad> es igual a 55. Si ésto es asi, entonces mostrará por pantalla el mensaje “ok boomer”. Si esto no es así, seguira con el resto del programa.

Fijate que la siguiente linea de codigo no se encuentra identada, por lo tanto no forma parte del if, por lo que el programa mostrara por pantalla el mensaje “bai bai” y finalizará.

✔ IF – ELSE:

La estructura anterior es muy limitada y normalmente se necesitará una estructura que permita elegir entre dos opciones o alternativas posibles, en función del cumplimiento o no

de una determinada condición. Si la condición es verdadera, se ejecuta la acción el bloque de codigo que componga al if y, si es falsa, se ejecuta el bloque de codigo que componga al else.

Su estructura se lee de la siguiente manera:

Si <condicion> es verdadera entonces
    se ejecutara esta accion
    y esta otra
    ...
Si es falsa entonces
    se ejecutara esta otra
    y esta otra
    …

ej:

```
mi_edad = 55

if mi_edad >= 55:
    print(“ok boomer”)
else:
    print(“aun soy un pendex”)

print(“bai bai”)
```

Este programa comienza declarando la variable <mi_edad> a la que se le asigna el dato de tipo entero 55.

El programa tiene una condicional simple(if), que analiza si el contenido de la variable <mi_edad> es igual o mayor a 55. Si ésto es asi, entonces mostrará por pantalla el mensaje “ok boomer”.

Si esto no es asi, es decir, si <mi_edad> contiene un numero que no es 55 y es menor a éste, entonces mostrará por pantalla el mensaje “aun soy un pendex”

Como la siguiente linea de codigo no forma parte del else, entonces se ejecutará y se mostrara por pantalla el mensaje “bai bai”.

- ✔ IF- ELIF-ELSE:

Muchas veces vamos a tener más de dos alternativas para elegir, o una variable que puede tomar varios valores. En esta estructura, se evalúa una condición o expresión que puede tomar n valores. Según el valor que la expresión tenga en cada momento se ejecutan las acciones correspondientes al valor. Es importante mencionar, que la primer condicion siempre se indicara con un if, las siguientes con un elif, y si ninguna de esas condiciones se cumplen, se ejecutara el else.

Su estructura se lee de la siguiente manera:

Si <condicion1> es verdadera entonces
    se ejecutara esta accion
    y esta otra
    …
Si <condicion1> es verdadera entonces
    se ejecutara esta accion
    y esta otra
    …
Si <condicion2> es verdadera entonces
    se ejecutara esta accion
    y esta otra
    …
Si <condicion3> es verdadera entonces
    se ejecutara esta accion
    y esta otra
    …
Si todas las condiciones anteriores son falsas entonces
    se ejecutara esta otra
    y esta otra
    …

ej:

```
mi_edad = 55

if mi_edad != 100 and mi_edad >= 55:
    print(“ok boomer”)
elif mi_edad == 100:
    print(“ahrre mirtha”)
else:
    print(“aun soy un pendex”)

print(“bai bai”)
```

Este programa comienza declarando la variable <mi_edad> a la que se le asigna el dato de tipo entero 55.

El programa tiene una condicional simple(if), que analiza si el contenido de la variable <mi_edad> es distinto a 100 y si es igual o mayor a 55. Si ésto es asi, entonces mostrará por pantalla el mensaje “ok boomer”.

Si esto no es asi, continuara analizando la condicion indicada en el elif. Si <mi_edad> es igual a 100, entonces se mostrara por pantalla el mensaje “ahrre mirtha”.

Si ninguna de las condiciones anteriores se cumplieron, entonces mostrará por pantalla el mensaje “aun soy un pendex”.

Como la siguiente linea de codigo no forma parte del else, entonces se ejecutará y se mostrara por pantalla el mensaje “bai bai”.

- Estructuras repetitivas(o bucles)

Durante el proceso de creación de programas, es muy común, encontrarse con que una operación o conjunto de operaciones deben repetirse muchas veces. Para ello es importante conocer las estructuras de algoritmos que permiten repetir una o varias acciones, un número determinado de veces.

Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan bucles , y se denomina iteración al hecho de repetir la ejecución de una secuencia de acciones.

Todo bucle tiene que llevar asociada una condición, que es la que va a determinar cuándo se repite el bucle y cuando deja de repetirse.

- ✔ WHILE:

La estructura while ejecuta un bloque de instrucciones mientras se cumple una condición.

La condición se comprueba antes de empezar a ejecutar por primera vez el bucle, por lo tanto, si la condición se evalúa a «false» en la primera iteración, entonces el bloque de instrucciones no se ejecutará ninguna vez. Este tipo de estructura es muy útil cuando no sabemos cuantas veces necesitamos ejecutar un bloque de código, pero sí conocemos cuando necesitamos que deje de ejecutarse.

Mientras la <condición> sea verdadera, entonces se repetirá
    este bloque de código

ej:

```
mi_edad = 55

while mi_edad != 0:
    print(“hola”)
```

Este programa comienza declarando la variable <mi_edad> y asignándole el entero 55.

Luego, la sentencia while evalúa si el contenido de la variable <mi_edad> es distinta a 0. Si esto es así, se imprimirá por pantalla el mensaje “hola”.

Luego, el programa volverá a evaluar si el contenido de <mi_edad> sigue siendo distinta a 0.

Si esto es así, se imprimirá nuevamente el mensaje “hola”, y así sucesivamente.

Como verás, en ningún momento hemos cambiado el valor de la variable analizada, por lo tanto este programa imprimirá “hola” infinitas veces.

- ✔ FOR:

La estructura for proporciona una forma compacta de recorrer un rango de valores cuando la cantidad de veces que se deber iterar un bloque de código es conocida.

Su estructura se lee de la siguiente manera:

Por <item> en (secuencia) entonces
    bloque de codigo a ejecutar

ej:

```
for numero in range(10):
    print(numero)
```

En el programa anterior, comenzamos declarando un bucle for, donde la variable <numero> en cada vuelta, va a tomar el valor de los números comprendidos en un rango del 0 al 9. Cuando <numero> valga 10, entonces se detendrá el bucle.

Se imprimirá el contenido de la variable en cada vuelta. Dando como resultado

```
0
1
2
3
4
5
6
7
8
9
```

3) Breve introducción a la Estacion 2.

- Funcion randint

Para generar números aleatorios en Python de valor *entero*, se suele utilizar la función `randint()`. La función `randint(a, b)` devuelve un número *entero* comprendido entre `a` y `b` (ambos inclusive) de forma aleatoria. Ejemplos útiles de esta función: determinar quién comienza una partida (jugador/PC); mira este ejemplo

```
import random

# ¿Quién comienza?

comienza = random.randint(0, 1)
```

```
if comienza == 0:
    print('Comienza el jugador')
else:
    print('Comienza el PC')

# Número aleatorio generado por el programa
numero = random.randint(1, 6)
```

- Listas:

Una lista es una estructura de datos y un tipo de dato en python con características especiales. Lo especial de las listas en Python es que nos permiten almacenar cualquier tipo de valor como enteros, cadenas y hasta otras funciones; por ejemplo:

```
lista = [1, 2.5, 'pepito', [5,6] ,4]
```

Una lista es un arreglo de elementos donde podemos ingresar cualquier tipo de dato, para acceder a estos datos podemos hacer mediante un índice.

```
print lista[0] # 1
print lista[1] # 2.5
print lista[2] # pepito
print lista[3] # [5,6]
print lista[3][0] # 5
print lista[3][1] # 6
print lista[1:3] # [2.5, 'pepito']
print lista[1:6] # [2.5, 'pepito', [5, 6], 4]
print lista[1:6:2] # [2.5, [5, 6]]
```