CONTENIDO TEMÁTICO
Introducción a la programación con Python

- Mi primer programa en Python
- Comentarios en Python
- Python como calculadora
  - Operadores matemáticos
- Manejo de variables y constantes
  - El operador de asignación
- Impresión de información por pantalla
  - Impresión multilinea
  - Concatenando datos para la impresión (Operadores de concatenación)
  - Impresión de información con formato
- Captura de datos por teclado
- Operadores (Lógicos – Relacionales)
- Tratamiento de texto (Funciones predeterminadas)
  - Mayúsculas, minúsculas, letra capital, espacios, otras

MI PRIMER PROGRAMA EN PYTHON

Recordemos: En muchos lenguajes de programación hasta hacer un simple hola mundo requiere de varias líneas; en esta oportunidad demostraremos lo sencillo que es hacer nuestro primer hola mundo en python; basta con utilizar la sentencia ***"print"***

```
print("Hola Mundo")
print('Mi primera aplicación en python')
```

```
Hola Mundo
Mi primera aplicación en python
```

- Cuando se desea mostrar el texto que se escribe al interior de los paréntesis, este se de be de escribir entre comillas
- Se pueden usar comillas dobles o simples, pero la buena práctica dice que deben ser co millas dobles

COMENTARIOS EN PYTHON

Al igual que en otros lenguajes de programación, los comentarios permiten documentar y leer el Código creado, este permite que otros desarrolladores conozcan el código código

Existen 2 maneras de crear comentarios en Python:

1. Cuando se desea comentar una sola línea de código se utiliza la almohadilla (#)

```
[ ]  # Este es un comentario de una sola línea en python
```

2. Cuando se desea comentar un conjunto de líneas se utiliza triple comillas para abrir y para cerrar el comentario.

```
[ ]  """ Este es
     un comentario
     de múltiples líneas
     en python """
```

PYTHON COMO CALCULADORA

Es mucha la sencillez para aprendizaje de este lenguaje de programación, pues acepta líneas de código de manera sencilla, tanto como poder aceptar operaciones matemáticas como se hace en una calculadora convencional.
Haremos unas operaciones básicas usando Python:

```
6 + 4
10
```

```
5 * 8
40
```

```
5 ** 3
125
```

Apreciamos una suma, una multiplicación y una potencia

La sentencia "*print*": Por ahora nos conformaremos con saber que esta función permite mostrar datos por pantalla, entonces ahora se podrá imprimir información de la siguiente manera:

```
print(3)
3
```

```
print(5 + 3)
8
```

```
print(16 / 4)
4.0
```

Se imprimen los resultados de las operaciones matemáticas. La sentencia print debe de contener los datos a mostrar dentro de los paréntesis. Estructura: ***print(Datos a mostrar)***

Jerarquía de los operadores aritméticos: al igual que en un proceso matemático aquí se aplica de igual manera este principio de izquierda a derecha de acuerdo al caso, en el siguiente orden jerárquico:

Prioridad

1: Destrucción de paréntesis
2: Radicales y potencias
3: Multiplicaciones y divisiones
4: Sumas y restas

Algunos ejemplos

```
print(2 * 5 + 3)
13
```

```
print(2 * (7 + 3) / 2)
10.0
```

```
print(5 + 2 * 10)
25
```

```
print(20 - 2 ** 2 + (5 - 3))
18
```

MANEJO DE VARIABLES Y CONSTANTES

Variable

- Es un tipo de objeto que funciona como contenedor, que almacena un valor que puede variar
- Cada variable debe de tener un nombre único dentro de un módulo; ejemplo: Nombre, Edad, sueldo…

Constante

- Es un tipo de variable, donde su contenido no cambia
- Los nombres de constantes son escritos en letra mayúscula, por ejemplo: FECHA_DE_NACIMIENTO

Reglas para dar nombres a las variables y a las constantes:

- No se deben de usar símbolos especiales (#$/()!...)
- Nunca se debe de iniciar con un nombre
- Solo se pueden usar letras, mayúsculas, minúsculas, números (Pero no al inicio) y el símbolo _

Tipos de datos asociados a las variables: Paython maneja 4 tipos de datos (Enteros-***int***, Flotantes-***float***, Booleanos-***bool*** y cadenas-***str***), pero python tiene la ventaja de que estos no son definidos con la variable, puesto que al asignar el valor a una variable esta adopta el tipo por defecto

El operador de asignación (**=**): Permite asignar valores a las variables; veamos los siguientes ejemplos:

Variables enteras

```
edad = 26
peso = 65
print(edad)
print(peso)
```

```
26
65
```

Variables flotantes

```
estatura = 1.75
sueldo = 450000
print(estatura)
print(sueldo)
```

```
1.75
450000
```

Variables Booleanas

```
respuesta = True
opcion = False
print(respuesta)
print(opcion)
```

```
True
False
```

Variables string

```
nombre = "Deivys Morales"
direccion = "Calle 5 # 20 - 20"
print(nombre)
print(direccion)
```

```
Deivys Morales
Calle 5 # 20 - 20
```

Identificar tipos de datos con ***type()***

```
print(type(nombre))
print(type(respuesta))
print(type(sueldo))
print(type(estatura))
```

```
<class 'str'>
<class 'bool'>
<class 'int'>
<class 'float'>
```

MANEJO DE VARIABLES Y CONSTANTES

Pero como cambian los valores de una variable?: Como se había dicho, una variable puede cambiar su contenido con el tiempo, solo basta con asignarle un nuevo valor y perderá el dato anterior, por ejemplo

La variable decrementa

```
numero1 = 35
print(numero1)
numero1 += 10
print(numero1)
```

```
35
45
```

La variable incrementa

```
numero2 = 7
print(numero1)
numero2 -= 4
print(numero2)
```

```
45
3
```

La variable cambia 3 veces

```
numero3 = 16
print(numero3)
numero3 = numero3 / 4
print(numero3)
numero3 *= 8
print(numero3)
```

```
16
4.0
32.0
```

Se puede reasignar un valor a una variable dándole a esta el nuevo dato, o aumentando o disminuyendo su valor en ciertas proporciones como se evidencia en estos ejemplos.

IMPRESIÓN DE INFORMACIÓN POR PANTALLA

La sentencia print(), esta función permite imprimir lo que se le indique dentro de sus argumentos, es decir, lo que se indique dentro de los paréntesis, lo cual puede ser un valor de cualquier tipo o una variable, y puede admitir múltiples argumentos, los cuales se mostrarán en el orden en que se escriban.

```
print("Información personal")
```

```
Información personal
```

```
print('Información personal')
```

```
Información personal
```

Uso de comillas dobles o simples (“ “) (‘ ‘) Las cadenas que se quieran mostrar se deben de escribir entre comillas, y funciona con comillas dobles o simples (La buena práctica indica que se debe de hacer con comillas dobles)

```
print('este es un curso "Básico" de Python')
```

```
este es un curso "Básico" de Python
```

```
print("este es un curso \"Básico\" de Python")
```

```
este es un curso "Básico" de Python
```

También es posible usar la barra acostada (\) para imprimir ciertos operadores que son sentencias de python, por ejemplo en la expresión: “este es un curso Básico de Python” se utilizan (\) para indicar que las comillas dobles de python se deben de imprimir en la salida por pantalla.

Imprimiendo saltos de línea: con “\n” es posible imprimir un salto de línea si es necesario, e incluso este se podría incluir al final o al principio de una expresión que se desee mostrar por pantalla, la cual generará un salto de línea

```
print("\n")
```

IMPRESIÓN MULTILINEA

También es posible crear una impresión multilínea, es decir, que podemos imprimir de tal forma que se vea tal como se escribe en el código, o dando saltos de línea con ***\n***. Veamos cada uno de estos casos

Caso 1: usando triple comillas podemos dar un formato de salida tal como se escriba en el código:

```
print(""" ======MENÚ PRINCIPAL=======
      ---OPCIÓN 1
      ---OPCIÓN 2
      ---OPCIÓN 3
""")
```

```
 ======MENÚ PRINCIPAL=======
      ---OPCIÓN 1
      ---OPCIÓN 2
      ---OPCIÓN 3
```

Caso 2: Usando ***\n*** damos saltos de líneas:

```
print("======MENÚ PRINCIPAL=======\n     ---OPCIÓN 1\n     ---OPCIÓN 2\n     ---OPCIÓN 3")
```

```
======MENÚ PRINCIPAL=======
     ---OPCIÓN 1
     ---OPCIÓN 2
     ---OPCIÓN 3
```

CONCATENANDO DATOS PARA LA IMPRESIÓN (OPERADORES DE CONCATENACIÓN)

A este punto ya podemos realizar impresiones un poco más agradables a la vista del ojo humano, y es que se pueden realizar concatenaciones, conjugando los diferentes tipos de datos (Textos con números y viceversa), para esto es necesario hacer uso de los operadores (,) y/o (+) donde se pueden concatenar textos o datos que contienen las variables, y hacer juego de impresiones entre textos, números y valores boléanos.

- El operador (+): Permite concatenar textos o variables tipo texto
- El operador (,): Permite concatenar textos con valores o variables numéricas ya sean estas enteras o flotantes

Veamos algunos ejemplos:

```
nombre = "Dagoberto Berdugo"
sueldo = 800000
empresa = "Colsubsidio"
print(nombre)
print(sueldo)
print(empresa)
```

```
Dagoberto Berdugo
800000
Colsubsidio
```

Se puede imprimir el contenido de una variable, digitando el nombre de la variable dentro del print sin comillas

```
nombre = "Dagoberto Berdugo"
sueldo = 800000
empresa = "Colsubsidio"
print(nombre + " trabaja desde el 2017 en " + empresa + " con una asignación salarial de $",sueldo)
```

```
Dagoberto Berdugo trabaja desde el 2017 en Colsubsidio con una asignación salarial de $ 800000
```

Notamos que al concatenar el texto con el operador (+) es necesario generar espacios dentro de las comillas, pero el operador (,) genera los espacios de separación de manera automática.

RETO: pruebe el ejercicio anterior haciendo cambios de los operadores (+) y (,); que sucede?

IMPRESIÓN DE INFORMACIÓN CON FORMATO

La instrucción (f) y el método format(): poco a poco vamos mejorando la presentación de cada una de nuestras impresiones, y en esto también pueden ayudar los métodos (*f*) y/o "***format()***", y es que estas dos sentencias permiten dar una salida organizada a los resultados que se quieran mostrar por pantalla

- ***Formateando con (f):*** Se inicia la cadena con *f* antes de la comilla de apertura, dentro de las comillas o de la expresión también se puede hacer referencia a valores o variables, para ello se usan las llaves ***{ }***
- ***Formateando con format():*** Se inicia la cadena dentro de las comillas incluyendo también los espacios donde irán los valores o las variables señalados por llaves **{ }**; una vez cerradas las comillas se agrega un punto (**.**) y la palabra ***format*** y dentro de los paréntesis se relacionan los nombres de las variables relacionadas en la leyenda dentro de las comillas.

Veamos ejemplos:

```
nombre = "Dagoberto Berdugo"
sueldo = 800000
empresa = "Colsubsidio"
print(f"{nombre} trabaja desde el 2017 en {empresa} con una asignación salarial de ${sueldo}")
```

```
Dagoberto Berdugo trabaja desde el 2017 en Colsubsidio con una asignación salarial de $800000
```

Uso de la instrucción (*f*)

```
nombre = "Dagoberto Berdugo"
sueldo = 800000
empresa = "Colsubsidio"
print("{} trabaja desde el 2017 en {} con una asignación salarial de ${}".format(nombre, empresa, sueldo))
```

```
Dagoberto Berdugo trabaja desde el 2017 en Colsubsidio con una asignación salarial de $800000
```

Uso del método *format()*

IMPRESIÓN DE INFORMACIÓN CON FORMATO

Usando el método ***format()*** también es posible experimentar diferentes formas de imprimir.

1. Usando los valores dentro del ***format***(En caso de que no se tengan variables)

```
print("{} trabaja desde el 2017 en {} con una asignación salarial de ${}".format("Dagoberto Berdugo", "Colsubsidio", 800000))
```

```
Dagoberto Berdugo trabaja desde el 2017 en Colsubsidio con una asignación salarial de $800000
```

2. Usando los índices de las variables que se encuentren dentro del ***format()***

```
nombre = "Dagoberto Berdugo"
sueldo = 800000
empresa = "Colsubsidio"
print("{2} trabaja desde el 2017 en {0} con una asignación salarial de ${1}".format(empresa, sueldo, nombre))
```

```
Dagoberto Berdugo trabaja desde el 2017 en Colsubsidio con una asignación salarial de $800000
```

En el caso 2 se evidencia que se accede a las variables de acuerdo al índice que se indica dentro de las llaves ***{ }***
Python también maneja un sistema de índices, al igual que otros lenguajes de programación los cuales siempre inician desde la posición cero (0), veamos en detalle los índices de las tres variables dentro del ***format():***

Índice	Posición #0	Posición #1	Posición #2
Variable dentro del format	empresa	sueldo	nombre

También es posible realizar operaciones dentro del formato y mostrarla directamente por pantalla así:

```
print("3 x 7 = {}, 8 x 8 ={}".format(3*7, 8*8))
```

```
3 x 7 = 21, 8 x 8 =64
```

CAPTURA DE DATOS POR TECLADO

La función ***input()*** permite ingresar datos desde el teclado a la aplicación. Los datos ingresados por defecto serán tipo texto ***(str),*** lo que implicaría si se quiere introducir otro tipo de datos el tener que convertirlo. Veamos algunos ejemplos:

```
input()
```

Se mostrará el cursor para ingresar el dato desde el teclado, pero también se podrá ingresar un dato directamente a una variable, que guardará el dato capturado.

```
nombre = input()
```

Para capturar un dato numérico entero se debe encerrar el input en el tipo de dato ya sea ***int()*** o ***float(),*** lo mismo ocurriría si se desea convertir a ***bool().***

```
edad = int(input("Digite su edad: "))
```

```
Digite su edad:
```

Un ejemplo completo se vería de la siguiente manera, donde se capturan diferentes tipos de datos y se imprimen por pantalla usando format().

```
print("Digite su nombre")
nombre = input()
apellido = input("Digite su apellido: ")
edad = int(input("Digite su edad: "))
print("Sus nombre y apellidos son: {} {}, y tiene {} años".format(apellido, nombre, edad))
```

```
Digite su nombre
Deivys
Digite su apellido: Morales
Digite su edad: 35
Sus nombre y apellidos son: Morales Deivys, y tiene 35 años
```

```
print("Digite su nombre", end=" ")
nombre = input()
apellido = input("Digite su apellido: ")
```

```
Digite su nombre Deivys
Digite su apellido: Morales
```

Para ingresar el dato y que este quede en la misma línea desde donde este se solicita se puede usar la sentencia ***end=“ ”.***

OPERADORES RELACIONALES

En un lenguaje de programación como python, los operadores relacionales son utilizados para realizar comparaciones o establecer relaciones entre datos y/o variables, lo cual genera un resultado lógico Verdadero o Falso ***(True – False)***

OPERADOR	INTERPRETACIÓN
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
==	Igual que
!=	Diferente de

```
8 < 9; 2 >= 2; 4 > 8; "Ana" < "Alba"
```

OPERADORES LÓGICOS

En ocasiones es indispensable unir ciertas comparaciones usando los operadores lógicos ***(and, or, not),*** para realizar diversas combinaciones de comparación según sea el caso; para entender estos operadores es necesario repasar un poco los conceptos de tablas de verdad.

P	Q	P or Q
V	V	V
V	F	V
F	V	V
F	F	F

P	Q	P and Q
V	V	V
V	F	F
F	V	F
F	F	F

P	not P
V	F
V	F
F	V
F	V

(P or Q): Solo será falso cuando todos los valores de sus premisas sean falsos
(P and Q): Solo será verdadero cuando todos los valores de sus premisas sean verdaderos
(not P): es equivalente a la negación de la proposición

Algunos ejemplos de aplicación:

- Se evidencian combinaciones entre operadores relacionales y operadores lógicos
- En python el texto se puede comparar usando operadores relacionales y se evidencia que diferencia entre las mayúsculas y las minúsculas.
- Compara las letras por la posición en la que se encuentra en el abecedario
- Identifica cuando una palabra está contenida en otra.

```
a = 8 < 9 and  2 >= 2          # (True)
a = 4 > 8 and 16 >= 17         # (False)
a = 9 != 8 and 4 == 5          # (False)
a = 9 >= 10 or 11 >= 10        # (True)
a = 5 > 2 or 1 < 2             # (True)
a = "coco" == "Coco"           # (False)
a = "andrea" < "alejandra"     # (False)
a = "socio" in "sociologo"     # (True)
b = 4 < 5                      # (True)
a = not 3 < 4 and not b        # (False)
print(a)
```

FUNCIONES PREDETERMINADAS PARA EL TRATAMIENTO DE TEXTO

Función	Definición	Ejemplo	Salida
print()	Imprime un argumento por pantalla	print("Colsubsidio")	Colsubsidio
len()	Determina la longitud en caracteres de una cadena.	len("Colsubsidio")	11
join()	Convierte en cadena utilizando una separación	lista = ("Cursos", "Colsubsidio") print("-".join(lista))	Cursos-Colsubsidio
split()	Convierte una cadena con un separador en una lista	docente = "Dagoberto Berdugo" lista_docente = docente.split() print(lista_docente)	[“Dagoberto”, “Berdugo”]
replace()	Reemplaza una cadena por otra	lenguajes = "Python y Java" print(lenguajes.replace("y", "o"))	Python o Java
upper()	Convierte una cadena en Mayúsculas	curso = "python" print(curso.upper())	PYTHON
lower()	Convierte una cadena en Minúsculas	curso = "PYTHON" print(curso.lower())	python
title()	Convierte la primera letra de cada palabra en mayúscula	curso = "PYTHON JAVA" print(curso.capitalize())	Python Java
capitalize()	Convierte la primera letra de una frase en mayúscula	curso = "PYTHON JAVA" print(curso.title())	Python java
strip()	Elimina espacios en blanco (iniciales y finales)	curso = " PYTHON " print(curso.strip())	“python”

FUNCIONES PREDETERMINADAS PARA EL TRATAMIENTO DE TEXTO

Función	Definición	Ejemplo	Salida
str()	Convierte un valor numérico a texto	str(45)	“45”
int()	Convierte a valor entero	int("45")	45
float()	Convierte un valor a decimal	float("3.1416")	3.1416
max()	Determina el máximo entre un grupo de números	lista = [2, 5, 3, 9, 6] print(max(lista))	9
min()	Determina el mínimo entre un grupo de números	lista = [2, 5, 3, 9, 6] print(min(lista))	2
sum()	Suma el total de una lista de números	lista = [2, 5, 3, 9, 6] print(sum(lista))	25
round()	Redondea después de la coma de un decimal	x = 3.1416 print(round(x))	3
type()	Devuelve el tipo de un elemento	x = 3.1416 print(type(x))	<class “float”>

Algunos ejemplos de aplicación

```
palabra = input("Digite una palabra ")
palabra =palabra.upper()
print(palabra)
```

```
Digite una palabra deivys
DEIVYS
```

```
palabra = input("Digite una palabra ").strip().capitalize()
print(palabra)
```

```
Digite una palabra      DEIVYS
Deivys
```

```
palabra = input("Digite una palabra ")
print(palabra.lower())
```

```
Digite una palabra DEIVYS
deivys
```

```
palabra = input("Digite una palabra ").title()
print(palabra)
```

```
Digite una palabra DEIVYS MORALES
Deivys Morales
```

EJERCICIO DE APLICACIÓN

```
# Se tiene una tarifa por hora de 20000 y se desea solicitar por teclado la cantidad de horas trabajadas para un día de trabajo
# Se desea mostrar por pantalla el valor a pagar por el empleador por el día trabajado descontando un 20% de impuestos
# Se desea presentar dos salidas formateadas, una que muestre el pago antes de impuestos y otra que la muestre con el descuento

TARIFA_POR_HORA = 20000
horas_trabajadas = int(input("Indique las horas trabajadas: "))
salario = TARIFA_POR_HORA * horas_trabajadas
print("Ganaste : ${:,.2f} antes de descontar impuestos".format(salario))

#descontar el 20% de impuestos
salario *= 0.8
print("Despues de descontar el impuesto te quedan ${:,.2f}".format(salario))
```

```
Indique las horas trabajadas: 5
Ganaste : $100,000.00 antes de descontar impuestos
Despues de descontar el impuesto te quedan $80,000.00
```

NOTA: se pueden aplicar diferentes tipos de formatos alas salidas, en este caso a la salida del dato flotante se le está dando un formato de (,) para los miles y millones y de (.) para los decimales, además de querer mostrar 2 decimales.