



# **VIDEOJUEGOS** **INTRODUCCIÓN A PYTHON**

**CLASE 2**

# Contenido

PRORAMACIÓN I .....	4
Introducción .....	5
Repaso de variables .....	5
Asignación de valores a las variables .....	6
conversión de variables .....	7
Operadores Aritméticos.....	8
Operadores relacionales .....	8
Operadores Binarios .....	8
Comentarios.....	10
Entrada de datos .....	11
Salida de datos .....	12
Ambientes Virtuales.....	14
Librerías.....	14
OS .....	15
Math.....	15
Conjunto de instrucciones .....	16
Instrucciones de Decisión .....	17
.....	20
Bibliografía .....	20

“

Este espacio curricular permite fortalecer el espíritu crítico y la actitud creativa del futuro graduado. Lo alienta a integrar los conocimientos previos, recreando la práctica en el campo real, al desarrollar un proyecto propio.

”



**PYTHON**

# PRORAMACIÓN I

Comenzamos la clase 2, sin mucho preámbulo vamos a comenzar con Python. Es importante que, desde este momento, tengas a mano una hoja blanca. Te voy a proponer el siguiente desafío, cada instrucción nueva que veamos en el curso la anotarás en esa hoja: con una pequeña descripción y un ejemplo de código. Al terminar el cuatrimestre tendrás una guía personalizada, guía que podrás consultar no recuerdes bien algo de lo que vimos.

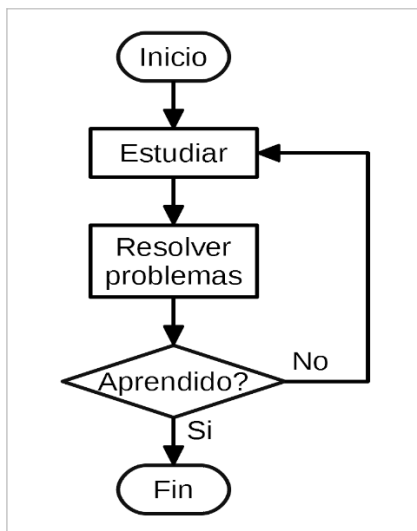
En esta clase vamos a repasar lo que vimos en la intro, recordando que es una variable y como hacer comentarios. Luego seguiremos con librerías e instrucciones de decisión.

¡¡Así que... A disfrutar... Que, al terminar el cuatrimestre, ¡¡estaremos jugando con nuestros propios juegos!!

# Introducción

Antes de comenzar, vamos a fortalecer el concepto de la palabra flujo. Ya habías visto que un algoritmo, es una secuencia ordenada de pasos que se ejecutan en forma secuencial. Lo que define el orden secuencial de ejecución, es el flujo.

El flujo, es el recorrido que hace la computadora al ejecutar cada línea de código. Existen muchas formas de representar esto, en el momento de análisis. Una de las formas es usando un diagrama de flujo.



Si observas el diagrama, vas a reconocer dos cosas. Una son las figuras, cada una de las figuras representa una instrucción de Python. La segunda, es la flecha. Esa flecha representa el flujo de ejecución. Es decir, te indica el recorrido que seguirá la computadora al ejecutar el código.



Si queremos almacenar valores Hexadecimales debemos anteponer el 0x (cero +x) delante del número, por ejemplo: **valorHex=0xe** Siendo e el valor hexadecimal.

De igual manera, usamos 0b para los binarios, y 0o para los octales

Es importante reconocer cual es el flujo de ejecución, porque tendremos instrucciones que nos permitirán alterar ese flujo cuando lo necesitemos.

## Repaso de variables

Una variable es un repositorio, nos permite almacenar en memoria un solo valor a la vez. Ese valor está representado por un tipo de dato. Recuerda que Python es un lenguaje de tipado dinámico, por lo que no es necesario indicar al declarar la variable que tipo de dato va a almacenar.

Los tipos de datos en Python son:

int	Número entero	n=4
float	Número con decimales	f=1.5
Booleano	Valor lógico Verdadero o Falso	b=True
String	Cadena de caracteres, se usa ""	s="Hola"

Existen otros tipos de datos, pero los veremos más adelante.

Desde ahora, usaremos la siguiente convención para ponerle nombres a las Variables

- 1- Constantes: no existe este tipo en Python, pero para indicar que el valor de la variable no va a cambiar, pondremos el nombre de la Variable en Mayúsculas.

Ej `PI=3.14`

- 2- Los nombres de las variables deben ser representativo.

Mal uso → `aaa=12`

Buen uso → `edad=12`

- 3- El nombre debe comenzar con una letra, o con un guion bajo. Usaremos el `_` para métodos o variables no públicas.

`nombre="Pablo"`

`_nombre="Pablo"`

- 4- Si el nombre tiene mas de una palabra, no puede tener espacios en blanco

Mal uso → `mi nombre="pablo"`

- 5- No usar palabras reservadas del lenguaje

Mal uso → `print="hola"`

#### Convención y formatos de nombre

- 6- Usaremos el formato **lowerCamelCase** para las variables. Es decir, la primera palabra en minúscula, y luego cada

palabra que sigue con la primera en Mayúsculas.

Ej `elNombreDePila="Pablo"`

- 7- Si queremos usar espacios, aplicamos el formato **snake\_case** para las funciones, en los módulos, o métodos

Ej `mi_funcion():`

- 8- Usaremos el formato **ScreamingSnakeCase** para las clases, y cosntantes. En este caso, todos los nombres tienen la primera letra en mayúscula.

Ej `BorrarPantalla`

## Asignación de valores a las variables

Asignar valores a una variable, es simplemente otorgarle un valor. Recordemos que las variables solo pueden almacenar un valor a la vez. Esto significa, que si la variable ya tenía un valor lo pierde al asignarle uno nuevo. Un ejemplo de asignación podría ser:

`nombre="Pablo"`

En este caso, estoy asignando la cadena pablo a la variable nombre. Python es de tipado dinámico, así que, en este caso asume que es una variable de tipo *string*.



Python es case-Sensitive, por lo tanto:

`a=5` y `A=5`

Son dos variables diferentes

Analizando lo anterior, voy a representarlos de modo gráfico, para explicarte como se debe leer:

Nombre ← "Pablo"

La lectura que debemos hacer de la asignación es siempre de derecha a izquierda. Esto se debe, porque si lo leemos de izquierda a derecha, esta línea de código sería incorrecto

```
edad=12
edad=6
```

Para explicarlo rápido, edad es igual a 12. Entonces el código dice que 12=6... 😊 y por supuesto, no sería correcto.

Continuando con el tema asignación, en Python, algunos trucos que pueden ser útiles. Te los enumero a continuación:

a,b,c=1,2,3	se asigna secuencialmente un valor a cada variable
a=b=c=10	asignamos 10 a todas las variables

## conversión de variables

convertir el valor de una variable, es cuando queremos cambiar un valor de un tipo de dato a otro. En Python, algunas cosas las tenemos resueltas, porque es dinámico. Por ejemplo, miremos el siguiente código:

```
edad=12
edad="Doce"
print(edad)
```

En otros lenguajes esto daría un error, pero no sucede en Python. En la primera línea, el lenguaje asume que la variable es entera. Pero en la segunda, cambia la variable a tipo cadena.

De todos modos, en algunas oportunidades, queremos forzar el tipo de dato o cambiarlo a uno en específico. Aquí es donde usamos la conversión. Por ejemplo, ejecutemos este código:

```
coordenadaX=10
moverHorizontal="2"
print(coordenadaX + moverHorizontal)
```

este código genera un error:

```
Traceback (most recent call last):
  File "d:\ejercicio.py", line 3, in <module>
    print(coordenadaX + moverHorizontal)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Esto sucede, porque no puedo sumar una letra a un número. Python, ofrece algunas funciones reservadas para convertir valor:

<b>str()</b>	Devuelve una cadena de caracteres
<b>int()</b>	Devuelve un entero
<b>float()</b>	Devuelve un flotante
<b>Bool()</b>	<u>Devuelve un booleano</u>

El error del código anterior, se resuelve de la siguiente manera:

```
coordenadaX=10
moverHorizontal=int("2")
print(coordenadaX + moverHorizontal)
```

Interesante ¿no?, pero no todo es tan simple. Por ejemplo, esta línea de código daría error. ¿Por qué?

```
moverHorizontal=int("hola"
```

```
Hola Mundo
```

## Operadores Aritméticos

Tenemos 3 tipos de operadores, uno de ellos es el operador aritmético. Como su nombre lo indica, permite hacer operaciones aritméticas con las variables.

Operador	Descripción
+	Suma dos números, o concatena dos string
-	Resta
*	Multiplicación
/	División con resultado siempre float
%	Módulo de la división
//	División con resultado int
**	Potencia

Por ejemplo, el siguiente código suma los valores de ambas variables:

```
coordenadaX=10  
mover=2  
coordenadaX= coordenadaX + mover
```

el código anterior, cambiar el valor de 10 en la coordenadaX a un nuevo valor de 12. Que tiene de interesante el +, que actúa diferente cuando las variables son cadenas:

```
cadena1="Hola"  
cadena2="Mundo"  
cadena3=cadena1 + " " + cadena2  
print(cadena3)
```

este código me muestra en pantalla:

## Operadores relacionales

Tenemos otro grupo de operadores llamados relacionales, también podemos conocerlos como comparativos. Su utilidad es comparar dos valores y devolver *True* o *False* como resultado según corresponda. Veamos un ejemplo:

```
print(2==3)
```

Este código nos muestra en pantalla:

```
False
```

La explicación es simple, como 2 “no es igual a” 3, el resultado es Falso.

Para entender mejor estos operadores, les presentamos la tabla con cada uno de ellos:

Operador	Descripción
==	Igual
!=	Distinto
>	Mayor
<	Menor
>=	Mayor o igual
<=	Menor o igual

## Operadores Binarios

Terminamos esta descripción de Operadores, presentando los operadores binarios and -or y



not. También tenemos los operadores bit a bit, & y |.

La tabla de operadores lo mostramos a continuación:

Operador bit a bit	Descripción
&	And (Y)
	Or (o)
	Not

Ahora bien, el uso de los operadores lógicos o bit a bit tiene dos caras, una compleja y una simple. Veamos la compleja primero. La comparación bit a bit con cada operando, obliga a convertir el valor en binario.

Veamos el resultado del código siguiente:

```
print(2 & 3)
```

Resultado es

```
2
```

¿No ayuda verdad?

Esto es porque tenemos que convertir cada operando en números binarios. En este caso el número 2 en binario es 10, y el 3 en binario es 11. Entonces, el código anterior quedaría como<sup>1</sup>:

```
print(bin(0b10 & 0b11))
0b10
```

<sup>1</sup> Recuerda que anteponer el 0b (cero + b) para representar números binarios. Por su parte la función bin() de Python convierte un numero decimal en binario

¿Esto ayuda menos verdad?

Qué largo lo estoy haciendo, pero antes de continuar te voy a mostrar la tabla de verdad<sup>2</sup> del **operador &**:

Operando		Salida
A	B	A&B
1	1	1
1	0	0
0	1	0
0	0	0

Ok, antes de entrar en desesperación veamos lo siguiente. El operador compara bit a bit

Entonces:

10  
11  
---  
10

Al comprar bit a bit, el resultado de 1 & 1 es → 1, por otro lado, el resultado de 0 & 1 es → 0. Ahora, podemos decir que el resultado de **0b10 & 0b11 es ob10**.

Sabiendo esto último, vamos a cambiar la tabla de verdad. Donde teníamos 1 pondremos True. Dónde teníamos 0, podremos False. La tabla para el operador and nos queda así:

Operando		Salida
A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

<sup>2</sup> La tabla de verdad proviene de la tecnología digital, nos muestra cual es el resultado final, cuando tenemos un determinado valor binario en la entrada de la compuerta.

Operador- Salida	
A	!A
True	False
False	True

¿Qué concluimos?

*Sólo tendremos True como resultado, cuando los dos operandos son True.*

Los otros dos operadores, funcionan de la misma manera, pero con resultados diferentes.

### El operador OR bit a bit

El símbolo que se usa es el pipe (|), representa el operador OR – se lee “o”, el carácter ASCII es ALT+124— funciona contrario al operador AND.

¿Cuál sería el resultado de...?

```
print(bin(2 | 3))
```

Correcto 0b11 😊

La tabla de verdad del operador OR es

Operando		Salida
A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

Por último, tenemos el **operador NOT**

Esto es simple, a la salida obtenemos lo contrario de la entrada

Los operadores bit a bit junto con los operadores relacionales, se usan mucho para construir condiciones. Algo que veremos más adelante.

## Comentarios

Los comentarios, con parte del código y se eliminan al compilar. Es información interna que podemos colocar en el código fuente. Tenemos que ser muy cautos al usar comentarios, el abuso puede ser contraproducente para la legibilidad del código.

En Python tenemos dos tipos de comentarios, de una sola línea o de múltiples líneas.

Los comentarios de una sola línea se usan también, como comentarios inline. Para los comentarios en una sola línea usamos el carácter #.

```
x,a,b=2,3,2
x = a if b else 0 #esto es un if
```

Los comentarios, no se usan sólo como comentario inline, también puede estar solos en una línea del código fuente.

```
# Asignamos valores a las variables
x,a,b=2,3,2
x = a if b else 0
```

Continuando con los comentarios, otra forma es hacerlo en más de una línea. Para ello, usamos las tres comillas simples (""" al comienzo y final.

```
'''
    Ejercicio 1
    Python
    Autor: Pablo
    Fecha:02-03-22
'''
```

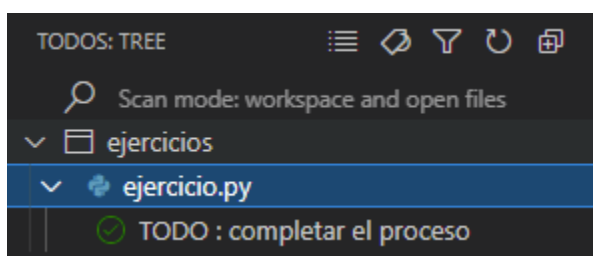
Continuamos con Python y comentarios, tenemos un par de opciones de “comentarios especiales”. Un caso es el TODO.

### #TODO

Se utiliza para indicar acciones pendientes en el código:

```
# TODO: completar el proceso
x,a,b=2,3,2
x = a if b else 0
```

Te invito a instalar la librería **Todo Tree**, y podrás ver todos los comentarios #TODO del código y navegar entre ellos



## Entrada de datos

En este apartado, nos vamos a concentrar solo en la entrada de datos por teclado. La palabra reservada de Python que usaremos es input.

Toda instrucción, tiene una sintaxis. En el caso de input, la que corresponde es:

### Input([Solicitud])

Esta instrucción de Python lee una línea de entrada del teclado, y convierte la entrada en una cadena de caracteres. Un ejemplo simple, lo podemos ver en el siguiente código:

```
a=input()
print(f"El valor ingresado es {a}")
```

el código anterior, pide un valor por teclado. El valor ingresado se asigna a la variable a. Luego se muestra en una leyenda el valor de a.

```
2
El valor ingresado es 2
```

Continuando con el código del ejemplo anterior, vamos a modificarlo ligeramente. Notaran en la figura que, no hay información para el usuario. Por lo que, al ingresar un valor, puedo no ser muy claro para él. La instrucción input, permite completar la orden con una leyenda que se mostrará en pantalla.

```
a=input("Ingresa un valor : ")
print(f"El valor ingresado es {a}")
```

El resultado en pantalla se vería de la siguiente manera:

```
Ingresa un valor : 2
El valor ingresado es 2
```

Como el valor que devuelve la instrucción input es siempre una cadena, podemos usar la

conversión de datos para almacenarla en la variable con el formato deseado. Por ejemplo, si lo que ingreso es un número:

```
a=int(input("Ingrese un valor : "))  
print(f"El valor ingresado es {a}")
```

## Salida de datos

La salida de datos por pantalla en Python se implementa con la instrucción print, la sintaxis de la instrucción se resume en lo siguiente:

```
print("mensaje", sep=",",end)
```

Siendo sep un argumento opcional que representa el carácter que se usará para separar dos mensajes.

Tenemos una lista importante de opciones para mostrar cosas en pantalla.

### Mostrar un mensaje

Es la variante más común, muestra una cadena de texto que se encuentra entre comillas, si es una variable y deseamos mostrar su valor, no usamos las comillas

```
print("El valor ingresado es ")  
print(a)
```

### Mostrar dos mensajes concatenados

La instrucción print, permite incluir más de un argumento, pero debemos separarlo por comas.

```
print("El valor","ingresado es ")
```

### una línea como separador

cuando deseamos generar espacio, para mejorar la visualización del mensaje. Podemos usar la instrucción print sin argumentos.

```
print()
```

### salto de línea

La instrucción print, ejecuta un salto de línea siempre.

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

Pueden notar que la cadena Mundo, se muestra en la línea de abajo

```
Hola  
Mundo
```

### Evitar el salto de línea

Agregamos el argumento end="" al final:

```
print("Hola",end="")  
print("Mundo")
```

Noten la diferencia

```
HolaMundo
```

Si queremos poner un espacio, sólo lo modificamos a end=" "

### Ignorar texto

Muchas veces, tenemos que poner un carácter que puede destruir la estructura del mensaje o sólo deseamos que se muestre como tal. Para eso usamos el carácter \ (ALT +92).

```
print("el \@ y la \" son caractreres")
```

### Las cadenas f

Se utilizan cuando quieres embeber texto dentro de otra cadena, y que Python la use como si estuviese formateado. Para ello, solo antepone la letra f al mensaje, y el texto incorporado se coloca entre llaves {}

```
edad=24
print(f"su edad es {edad}")
```

### el formato %

se utiliza para formatear la cadena de texto.

```
edad=str(24)
print("su edad es %s" % f"{edad}")
```

se puede colocar más de un texto, y la relación es secuencial. Con una variante en el format que también está permitida.

```
edad=str(24)
print("la edad de %s es %s" % ("Pablo",edad))
```

el método str.format

es similar al %, pero para indicar los valores dentro de la cadena se usa {} ó {#}

```
edad=str(24)
print("la edad de {} es {}".format("Pablo",f"{edad}"))
```

variante con números e igual cambio permitido en el format.

```
edad=str(24)
print("la edad de {0} es {1}".format("Pablo",edad))
```

```
la edad de Pablo es 24
```

### Separadores

Usamos un carácter para separar dos argumentos

```
print("Columna 1","Columna2", sep="-")
```

### Separar en ancho de columna

```
print("{0:15}{1:15}{2:15}".format("Columna 1","Columna 2","Columna 3"))
```

```
Column 1      Column 2      Column 3
```

Mostrar flotante con decimales

Es similar al anterior, pero se determina la cantidad de decimales y el tipo de dato numérico

```
temperatura=25.656566
print("La temperatura es
{0:.2f}".format(temperatura))
```

## Ambientes Virtuales

Estamos avanzando mucho con Python, así que vamos a hacer un paréntesis para implicar algo que puede ser útil en el futuro... Los ambientes virtuales.

El ambiente virtual, nos permite encapsular en un proyecto todas las versiones de los paquetes que usamos. Esta funcionalidad está en la librería virtualenv, en lugar de instalar todo en todo el entorno de nuestro pc. No viene en el instalador del interprete por lo que debemos instalarla con el comando pip3.

Pasos para seguir

- 1- Entramos en la terminal menú>>Terminal>>new terminal (CTRL+Shift+ñ)

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\PVilaboar>
```

- 2- Instalamos la librería **virtualenv**

**pip install virtualenv**

- 3- En la línea de comandos, nos dirigimos a la carpeta de trabajo. dentro de esa carpeta escribimos el comando para crear el entorno virtual:

```
python -m virtualenv venv_tp1
```

- 4- Luego, debemos activar el ambiente virtual

```
.\venv_tp1\Scripts\Activate
```

- 5- Ya tenemos el ambiente creado

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
PS D:\Users\pablo\proyecto> .\venv_tp1\Scripts\activate
(venv_tp1) PS D:\Users\pablo\proyecto> python --version
Python 3.10.0
(venv_tp1) PS D:\Users\pablo\proyecto> pip freeze
(venv_tp1) PS D:\Users\pablo\proyecto> pip install pygame
Collecting pygame
  Downloading pygame-2.1.2-cp310-cp310-win_amd64.whl (8.4 MB)
    8.4/8.4 MB 6.2 MB/s eta 0:00:00
Installing collected packages: pygame
Successfully installed pygame-2.1.2
(venv_tp1) PS D:\Users\pablo\proyecto> pip freeze
pygame==2.1.2
(venv_tp1) PS D:\Users\pablo\proyecto>
```

Podes ver el siguiente video



<https://youtu.be/dB8Eaxrvxo4>

## Librerías

¡¡Bien!!

Estamos entrando en librerías, eso significa que ya vamos a estar haciendo cosas más potentes.

Las Librerías son como los powerups de los videojuegos, al importantes en nuestro código fuente nos da más poderes. Desde luego, hay muchas librerías y no estamos obligados a importar todas. Sólo lo hacemos con aquellas que vamos a usar.

Para importar librerías usamos la palabra reservada import

```
import math
```

noten el detalle de indicar a que librería pertenece, al completar la instrucción con math.pi:

```
import math
print (math.pi)
```

También, nos permite importar y renombrar la librería, este es un ejemplo

```
import math as mate
print (mate.pi)
```

Usar la palabra import, indica que importamos toda la librería math. Pero también podemos usar un import parcial. Esta acción se conoce como importación de submodulo. Si sólo queremos importar el método pi, usamos la instrucción from

```
from math import pi
print ((-pi))
```

en este caso, podemos comprobar que no todas las funciones de math, están disponibles.

```
from math import degrees

print (radians(pi))
```

En este caso el mensaje es que radians no está disponible. El código correcto para el ejemplo es:

```
from math import degrees, radians, pi
print (radians(pi))
```

## OS

Es una librería que provee funciones que me facilitan el trabajo con acciones dependientes del sistema operativo. Abarca desde abrir y gestionar archivos, crear carpetas o limpiar la consola.

En este nivel lo usaremos para incluir la función de limpiar pantalla:

```
#importar la librería
import os

'''
usando el método system, pasamos
como argumento cls para borrar la
consola
'''
os.system("cls")
```

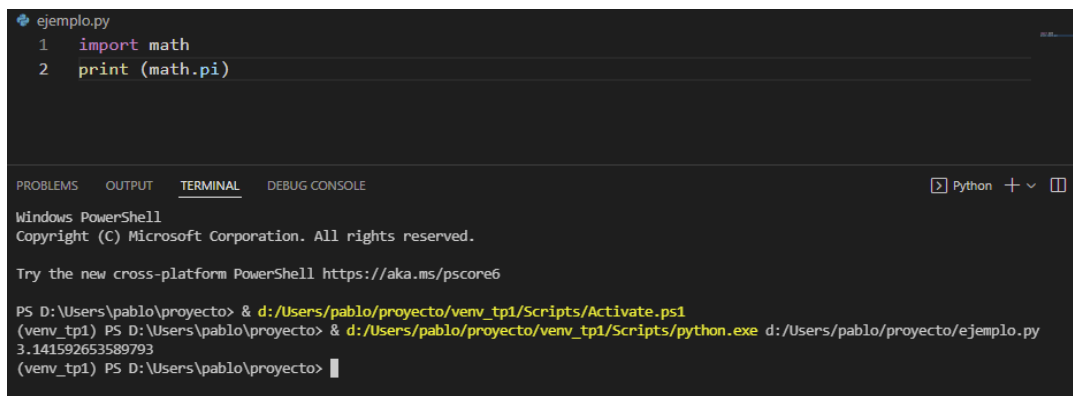
## Math

Brinda acceso a todas las funciones matemáticas, y trigonométricas. Estas

funciones no pueden ser usadas con números complejos. Como toda función, algo que veremos más adelante, devuelve un valor de retorno.

No es la idea describir cada una de las funciones, para eso te invito a visitar la página de documentación de Python en:

<https://docs.python.org/es/3/library/math.html>



```
ejemplo.py
1 import math
2 print (math.pi)

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/powershell

PS D:\Users\pablo\proyecto> & d:/Users/pablo/proyecto/venv_tp1/Scripts/Activate.ps1
(venv_tp1) PS D:\Users\pablo\proyecto> & d:/Users/pablo/proyecto/venv_tp1/Scripts/python.exe d:/Users/pablo/proyecto/ejemplo.py
3.141592653589793
(venv_tp1) PS D:\Users\pablo\proyecto>
```

Ilustración 1- Uso de la librería Math

En esa página, encontrar funciones logarítmicas, exponenciales, funciones de representación de números, trigonométricas e hiperbólicas.

Algo que, si es importante ver, es que las funciones trigonométricas no usan números enteros. Estas trabajan con radianes.

Un radian es una unidad de medida para ángulos, un sistema internacional de unidades de medida. Un radian es equivalente a

$$180^\circ/\pi$$

Python, por medio de la librería math, nos facilita la conversión de grados a radianes y, al contrario.

```
import math
angulo=90
```

```
print (f"el angulo de {angulo} grados,
equivale a {math.radians(angulo)} radianes")
```

de igual modo, podemos pasar de radianes a grados

```
import math
print (math.degrees(1.57))
```

el resultado es 89.95437383553924, pero es claro que se debe al redondeo. También la librería math me brinda funciones como math.pi que devuelve el valor pi.

## Conjunto de instrucciones

Todo lenguaje de programación debe tener, instrucciones secuenciales, de decisión y de repetición

En todos los lenguajes tenemos dos tipos de instrucciones, las simples y las compuestas. En Python, eso no es tan simple de describir. Pero



podemos decir que por un lado estas las instrucciones simples como

```
print (radians(pi))
```

Es decir, cuando se trata de solo una línea de código. Y los bloques, cuando son varias líneas de código. En Python ese bloque, se representa con los dos puntos, y las líneas que están dentro del bloque separadas del margen izquierdo con un doble tab.

En general los bloques se usan en instrucciones con las decisiones y las repeticiones, pero también se usan en las funciones o las clases.

```
if a==b:
    print ("linea 1 del bloque")
    print ("linea 2 del bloque")
```

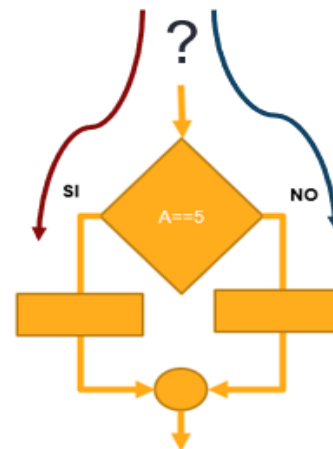
## Instrucciones de Decisión

También conocidas como sentencias condicionales, se usan en todos los programas para crear caminos alternativos en el flujo de ejecución del código. Estos condicionales nos dan la posibilidad de, dada una condición, alternar entre una fracción de código u otra. Es importante saber que, no se pueden ejecutar ambos caminos a la vez.

Estas condiciones, son como preguntas donde la respuesta es sí o no... O también. Verdadero o falso. Se usan en consecuencia los operadores lógicos y relacionales. Existen métricas que miden la complejidad del código contando la cantidad de expresiones condicionales. En el futuro, y sobre todo

cuando programamos en objetos, es preferible tratar de evitar el uso de los IF.

Recordando conceptos de la primera clase, al diseñar algoritmos habíamos hablado del flujo. En el grafico anterior, que representa un diagrama de flujo con un IF, las flechas



amarillas representan el flujo de la información. Las flechas rojas y azules representan el recorrido posible del flujo de información. Por qué aclaramos esto, porque al tener un “if” solo se puede ejecutar una de las dos ramas. Por lo que, el “if” determina cuál de las ramas se ejecuta y cual no.

Existen muchas formas de programar un “if”, y alternativas donde ponemos un “if” dentro de otro “if” (se llama “if” anidado). Pero todo eso, depende de la implementación que debemos programar.

Todas las instrucciones “if”, se componen de una condición, una rama para la salida de verdadero y una rama para la salida del falso.

**if [condición]:**

**Código cuando la condición da true**

**else:**

**Código cuando la condición da false**

Las condiciones

Se trata de una comparación, generalmente se usan los operadores relacionales. También una condición puede estar compuesta por más de una comparación relacionadas con los operadores lógicos.

Las condiciones, tienen que estar armadas de tal forma que deben como resultado verdadero o falso.

### Condicional solo con salida de verdadero

La instrucción **"if"** más simple, es aquella que se programa para decidir si un bloque de código se ejecuta o no. El flujo de datos sigue hasta la condición. Si la comparación en dicha condición es verdadera, el código se ejecuta. Si no, salta a la primera línea después del **"if"**.

```
✓ if a==b:
    print ("línea 1 del bloque")
    print ("línea 2 del bloque")
print ("primera línea despues del if")
```

Analicemos el código anterior, después del **"if"** tenemos la condición (a==b). si eso es verdadero, es decir a "es igual a" b. Entonces se ejecuta el bloque. ¿cuál es el bloque? Luego de la condición tenemos el carácter **":"**, eso indica que comienza el bloque. Todas las líneas de código, separadas con el doble **"tab"** es el bloque del **"if"**. El bloque entonces es el siguiente código:

```
    print ("línea 1 del bloque")
    print ("línea 2 del bloque")
```

Luego de ejecutar el bloque sigue con la siguiente línea:

```
print ("primera línea despues del if")
```

si la condición (a==b), hubiese arrojado false. El bloque no se ejecuta, y salta a la línea:

```
print ("primera línea despues del if")
```

Simple...

### Condicional con salida de verdadero y falso

Una segunda opción, es programar bloques de código tanto para una posible respuesta verdadera, como para una falsa. El **"else"**, representa la rama falsa de la condición. Podría leerse como un "sino". Es decir:

Si a==b es verdadero entonces ejecutamos el bloque verdadero. Si no, ejecutamos el bloque falso.

Es importante aclarar, que "nunca" podremos ejecutar el bloque verdadero y el falso. En todos los casos, según la condición se ejecuta uno de los dos.

```
✓ if a==b:
    print ("línea 1 del bloque verdadero")
    print ("línea 2 del bloque verdadero")
else:
    print ("línea 1 del bloque verdadero")
    print ("línea 2 del bloque verdadero")

print ("primera línea despues del if")
```

### Condicional anidado

En muchas ocasiones, podemos programar un **"if"** dentro de un **"if"**. Esto es simple, porque el bloque de código puede contener cualquier instrucción de Python. Un ejemplo serio:

```

if a==b:
    if a > b:
        print ("linea 1 del bloque verdadero")
        print ("linea 2 del bloque verdadero")
    else:
        print ("linea 1 del bloque verdadero")
        print ("linea 2 del bloque verdadero")

print ("primera linea despues del if")

```

Podemos programarlo también en el bloque de falso:

```

if a==b:
    print ("linea 1 del bloque verdadero")
    print ("linea 2 del bloque verdadero")
else:
    if a > b:
        print ("linea 1 del bloque verdadero")
        print ("linea 2 del bloque verdadero")

print ("primera linea despues del if")

```

## Elif

Cuando tenemos más de dos posibilidades, necesitamos más de dos ramas. Algo que “if” no permite, tenemos que anidar los “if” como el ejemplo anterior. Pero, Python me ofrece una opción para ello, el “elif”

Es decir, juntamos el “else” con el “if” de este código:

```

if a==b:
    print ("linea 1 del bloque verdadero")
    print ("linea 2 del bloque verdadero")
+ elif a > b:
    print ("linea 1 del bloque verdadero")
    print ("linea 2 del bloque verdadero")

print ("primera linea despues del if")

```

Y nos queda algo así:

```

if a==b:
    print ("linea 1 del bloque verdadero")
    print ("linea 2 del bloque verdadero")
elif a>b:
    print ("linea 1 del bloque verdadero")
    print ("linea 2 del bloque verdadero")

print ("primera linea despues del if")

```

## Versión compacta del condicional

Para cerrar, y solo con el afán de mostrarte que a veces las cosas se pueden complicar. Te dejo una forma compacta para programar un “if” en Python.

```

numero=5
#acá decimos que si numero es igual a 5
respuesta= "El número es 5" if (numero==5) else "El número no es 5"
print (respuesta)

```

El resultado es:

El número no es 5

## Bibliografía

- Varó, A. M., Sevilla, P. G., & Luengo, I. G. (2014). Introducción a la programación con Python 3. Recuperado de <http://dx.doi.org/10.6035/Sapientia93>



**PYTHON**