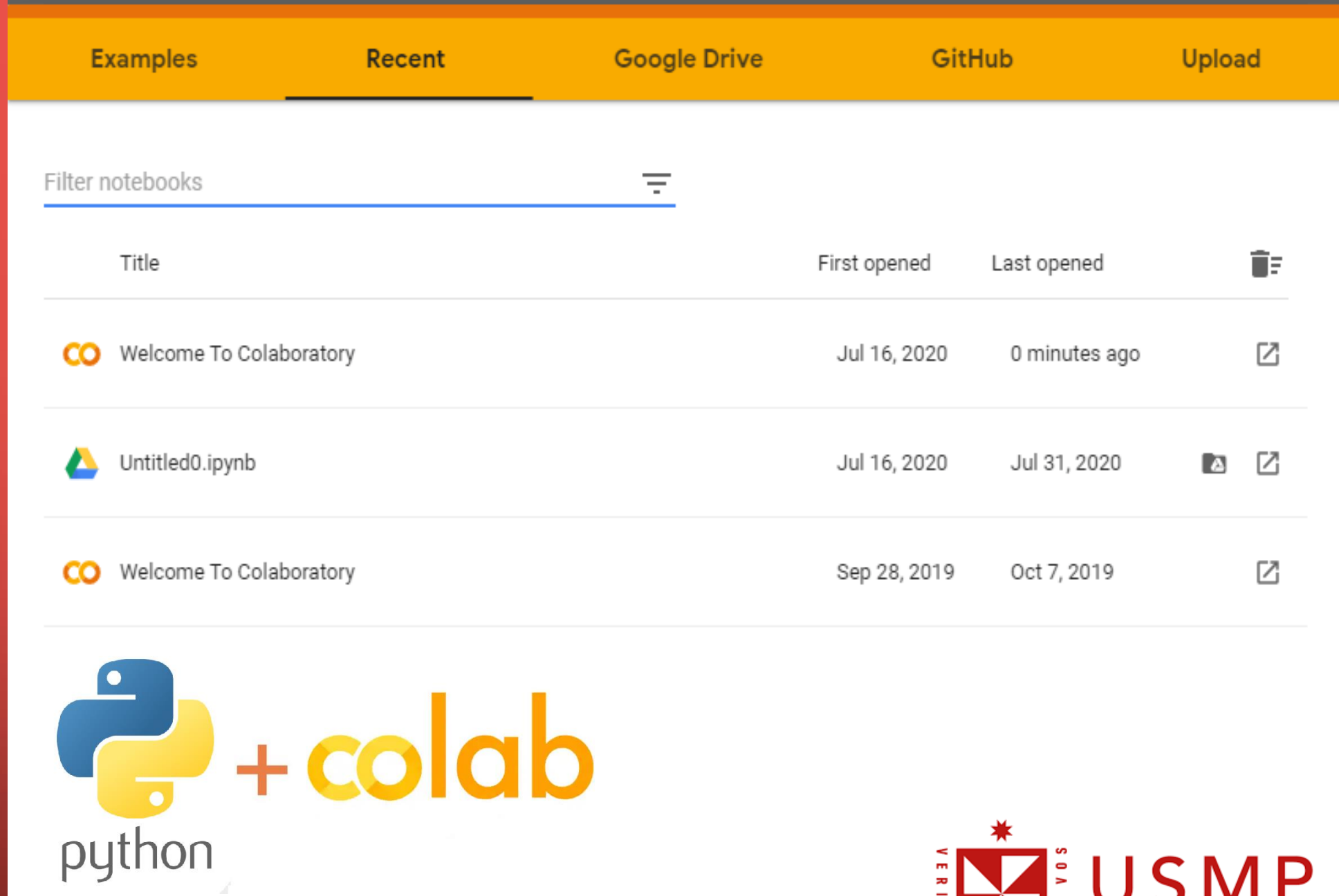
PYTHON APLICADO A LA DOCENCIA



python + colab

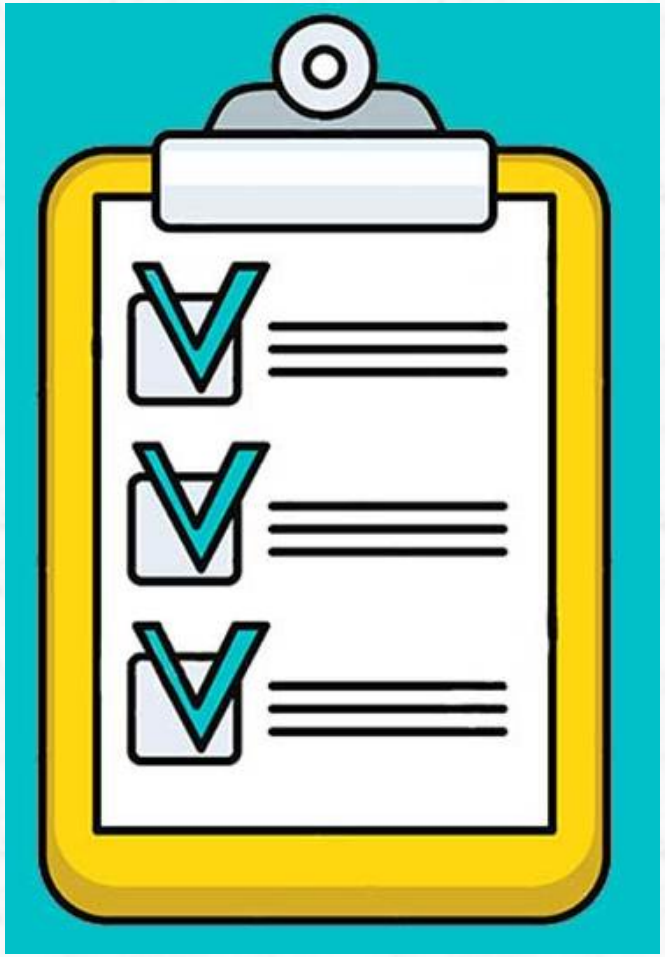
Mag. Antonio Gamero Paredes

USMP
UNIVERSIDAD DE SAN MARTÍN DE PORRES

SESIÓN 1

CONTENIDO:

- ¿Por qué Python?
- Entorno de trabajo
- Tipos y estructura de datos



LOGRO DE LA SESIÓN:

Al finalizar la sesión el participante aplica el lenguaje de programación Python teniendo en cuenta los tipos de datos existentes en el campo de la docencia.

¿POR QUÉ PYTHON?

- Es un lenguaje de programación interpretado ¡Bye compilador!

- Posee un tipado dinámico, es decir no requiere que se declare el tipo de dato de cada variable creada y además puede cambiar conforme se le vaya asignando valores.

- Recomendado para aprender a programar, sintaxis muy sencilla y legible (como si estuviéramos hablándole al ordenador).

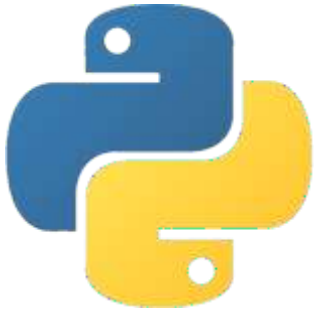
¿POR QUÉ PYTHON?

- ¡Código abierto! Completamente gratis, libre de usar y distribuir sin perder presencia en ámbitos comerciales.
- Es multiplataforma, se puede utilizar y ejecutar en Windows, Linux, Mac, etc.
- Enorme cantidad de módulos y paquetes respaldados por la comunidad.

¿POR QUÉ PYTHON?

Porque es un lenguaje de programación que presenta:

- Facilidad de uso
- Legibilidad de código
- Gran cantidad de módulos y paquetes
- Uso mundial



No requiere que se defina el tipo de variable pero sí se deben cumplir ciertas reglas:

- Usar solo caracteres alfanuméricos y el guión bajo ‘_’
- No debe tener espacios ni empezar por un número
- Case sensitive (Edad <> edad)

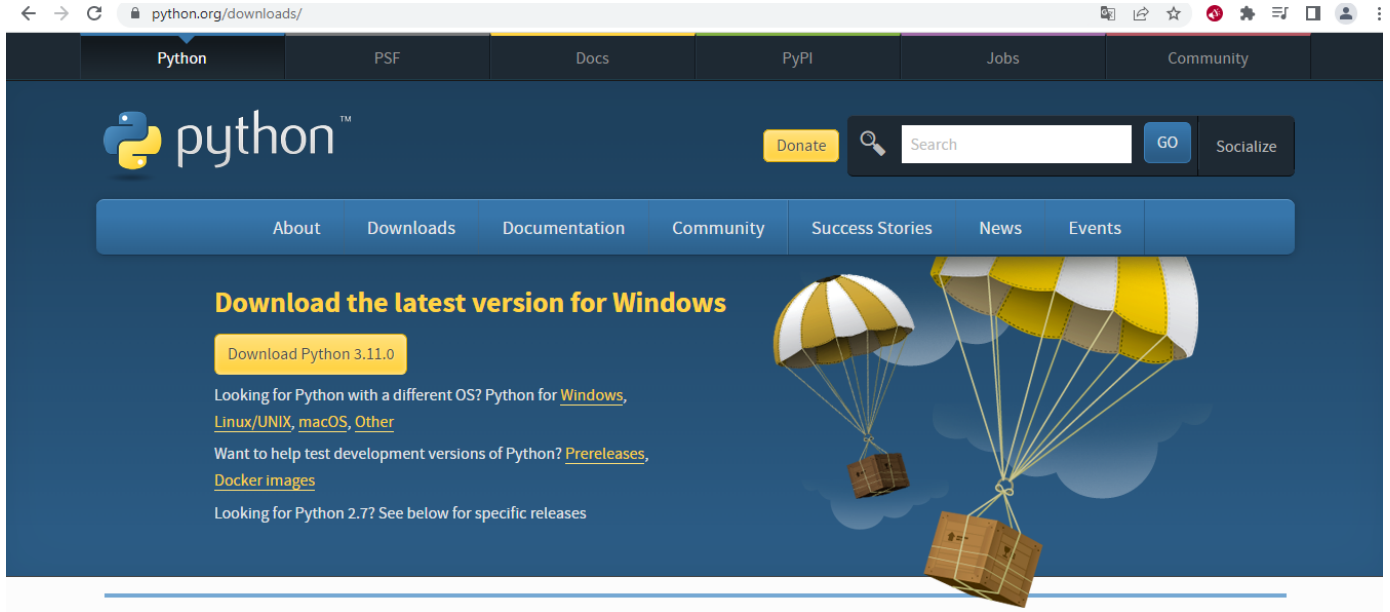
¿QUIÉNES UTILIZAN PYTHON?

- Google & YouTube.
- Instagram desarrollado en Django (framework de Python).
- Recomendación de series y películas en Netflix.
- Casi la cuarta parte de Facebook está escrito en Python.

PROGRAMANDO EN PYTHON

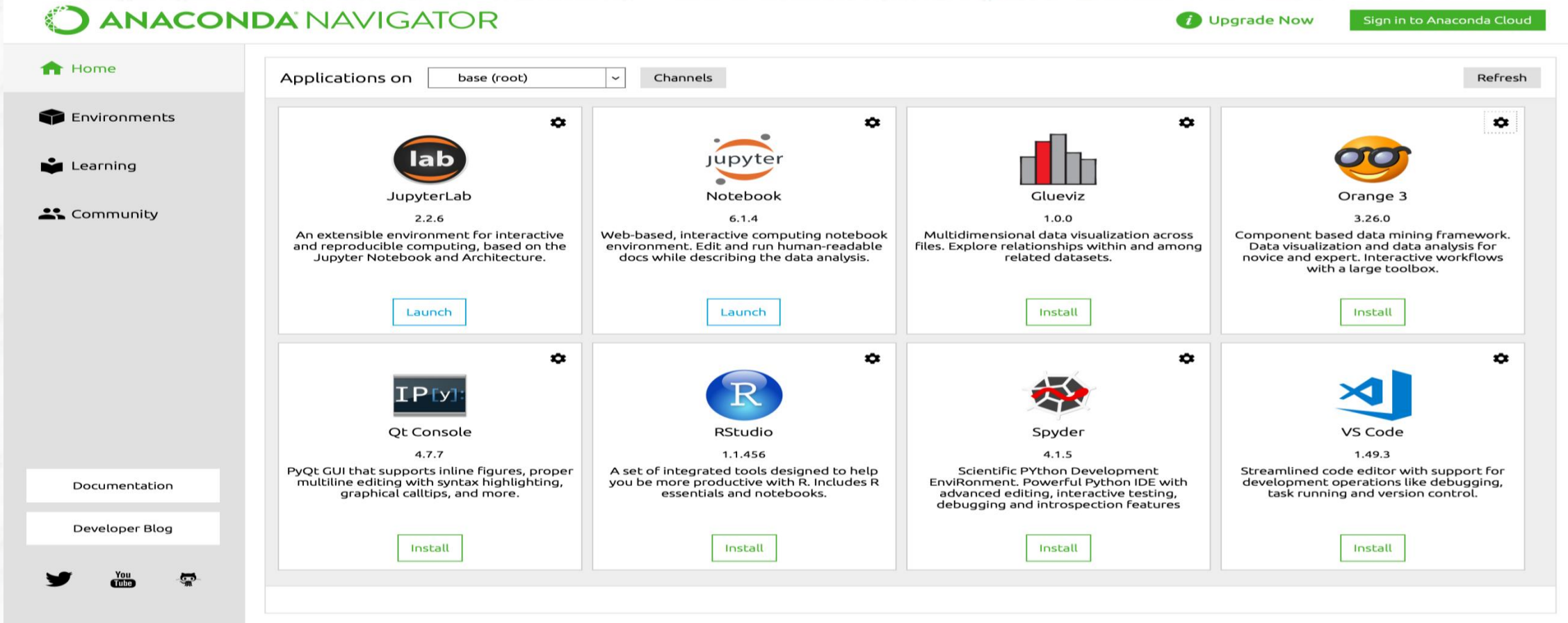
- Si estamos trabajando en una distribución de Linux, ya contamos con Python instalado localmente, en la consola se puede verificar la versión con: ***python --version.***

- De manera general, podemos descargar la versión que deseamos de Python en su página oficial: https://www.python.org/downloads/



PROGRAMANDO EN PYTHON

- Anaconda es una suite de código abierto que contiene, entre sus principales aplicaciones, Jupyter Notebook y Spyder. Cuenta con +250 librerías instaladas ideales para el desarrollo de proyectos de Ciencia de Datos, además de un gestor para actualizar o instalar librerías.



PROGRAMANDO EN PYTHON

Google Colaboratory

Es la solución de Google para el desarrollo de proyectos de Ciencia de Datos alojados completamente en la Nube, entre sus ventajas tenemos:

- ✓ Brinda una máquina virtual con 13 gb de ram y 50 gb de disco.
- ✓ Es completamente gratuito aunque pronto habrá una versión de paga.
- ✓ Se sincroniza con nuestros archivos de Google Drive y nuestros notebooks se guardan automáticamente en una carpeta llamada Colab Notebooks.

CONSTANTES Y VARIABLES

- Las constantes carácter (string) se permiten con comillas simples (') o dobles (")
- Las variables tienen que empezar por letras o guion bajo. **Son Case Sensitive!**

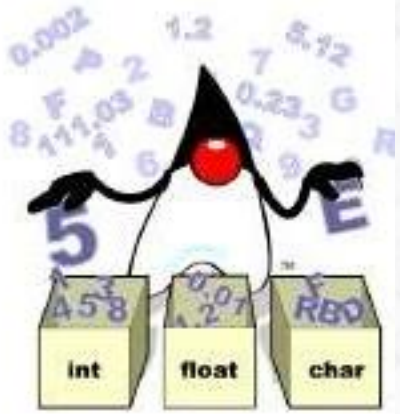
- Palabras reservadas: No se pueden utilizar como variables

```
False   class   return  is      finally
None    if      for     lambda  continue
True    def     from    while   nonlocal
and     del     global  not     with
as      elif    try     or      yield
assert  else    import  pass
break   except  in      raise
```

TIPOS DE DATOS

1. **Booleano:** Toma el valor de Verdadero o Falso
   - b1 = True

2. **Numérico:** Podemos clasificarlos en 3 grupos:
   - Entero: -18, 1, 5, 100
   - Decimal: -7.5, 2.4, 3.252, 8.0
   - Complejo: 0.5 - 2j



3. **String:** Es un tipo de dato **inmutable**, una vez creado no se puede modificar:
   - s1 = 'Soy un String'
   - s2 = "Yo también"

TIPOS DE DATOS BÁSICOS

Objetos escalares

- Int, usado para representar números enteros, ej. 4 o 2675
- Float, usado para representar números reales, ej. 3.14 o 34.0
- Bool, usado para representar números booleanos True y False

Built-in Function type

```
In [1]: type(3)
Out[1]: int
```

```
In [3]: type(3.0)
Out[3]: float
```

```
In [4]: type(True)
Out[4]: bool
```

```
In [10]: type(type)
Out[10]: type
```

OPERACIONES CON NÚMEROS

Expresiones

Operaciones entre int y floats

- Suma → i + j
- Diferencia → i - j
- Producto → i * j
- División → i / j
- División entera → i //j
- Resto → i % j
- Potencia → i ** j

Precedencia

Parentesis
Potencia
Producto
Suma
Izquierda a derecha

¿Cuál es el resultado en Python?

```
7 - 6 + 2 ** 3 / 4 * 5
```

COMENTARIOS

El símbolo para comentar una línea es #. No necesita marca de terminación

```
In [2]: print('Hola')
        #Esto es un comentario
        #print('¿qué tal?)
        print('Adios')
```

```
Hola
Adios
```

El símbolo para comentar un bloque de líneas es tres comillas simples (‘’’)

```
In [5]: print('Hola')
        '''
        Esto es un bloque de comentarios
        Comentario linea 1
        Comentario linea 2
        '''
        print('Adios')
```

```
Hola
Adios
```

FORMATOS PARA FECHAS Y HORAS

%a	Nombre local abreviado de día de semana
%A	Nombre local completo de día de semana
%b	Nombre local abreviado de mes
%B	Nombre local completo de mes
%c	Representación local de fecha y hora
%d	Día de mes [01,31]
%H	Hora (horario 24 horas) [00,23]
%I	Hora (horario 12 horas) [01,12]
%j	Número de día del año [001,366]
%m	Mes [01,12]
%M	Minuto [00,59]

FORMATOS PARA FECHAS Y HORAS

%p	Etiqueta AM o PM
%S	Segundo
%U	Nº semana del año. Se considera al Domingo como primer día de semana [00,53]
%w	Establece el primer día de semana [0(Domingo),1(Lunes)... 6].
%W	Nº semana del año (Se considera al Lunes como primer día de semana) [00,53]
%x	Fecha local
%X	Hora local
%y	Año en formato corto [00,99]
%Y	Año en formato largo
%Z	Nombre de Zona Horaria

AYUDA EN PYTHON

```
In [*]: help()
```

```
Welcome to Python 3.7's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at https://docs.python.org/3.7/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".


help>
```

ESTRUCTURAS DE CONTROL

Condicionales (if, elif, else) e iterativas (for y while)

- Dos tipos:
  - Condicionales: nos permiten evaluar si una o más condiciones se cumplen, para decidir qué acción vamos a ejecutar.
  - Iterativas o bucles: nos permiten ejecutar un mismo código, de manera repetida, mientras se cumpla una condición.
- En Python es importante la indentación (*indentation)* o espaciado, tanto para las estructuras de control, como para la definición de clases y funciones

ESTRUCTURAS DE CONTROL CONDICIONALES

if, elif, else

- Nos permiten evaluar si una o más **condiciones se cumplen**, para decir qué acción vamos a ejecutar.
- Las estructuras de control de flujo condicionales, se definen mediante el uso de tres palabras claves reservadas, del lenguaje: **if (si), elif (sino, si) y else (sino).**
- Pueden haber cero o más **elif** (abreviatura para else if), y 0 o 1 **else**.

```
if [condicion 1]:
        [instrucciones 1]

elif [condicion 2]:
        [instrucciones 2]

else:
                . . .
```

ESTRUCTURAS DE CONTROL CONDICIONALES

```
x=5
if x<10:
    print('Menor que 10')
if x>20:
    print('Mayor que 20')
print('Programa terminado')
```

```
Menor que 10
Programa terminado
```

```
x=22
if x<10:
    print('Menor que 10')
if x>20:
    print('Mayor que 20')
print('Programa terminado')
```

```
Mayor que 20
Programa terminado
```

```
x=12
if x<10:
    print('Menor que 10')
if x>20:
    print('Mayor que 20')
print('Programa terminado')
```

```
Programa terminado
```

Para definir los bloques de sentencias en Python se utiliza la identación.
En otros lenguajes se usan {} o begin/end

ESTRUCTURAS DE CONTROL CONDICIONALES

if, elif, else

- Para describir la evaluación a realizar sobre una condición, se utilizan operadores relacionales (o de comparación):

`==` `!=` `<` `>` `<=` `>=`

- Y para evaluar más de una condición simultáneamente, se utilizan los operadores lógicos **and**, **or** y el **xor**.

```
If   edad < 18:
        print ("Menor de edad")

elif   edad == 18:
        print ("Dieciocho años cumplidos")

else:
        print ("Mayor de edad")
```

ESTRUCTURAS DE CONTROL CONDICIONALES

```
x=12
if x<10:
    print('Menor que 10')
else:
    print('Mayor que 10')
print('Programa terminado')
```

```
Mayor que 10
Programa terminado
```

El operador de comparación distinto se escribe con !=

```
5!=3, 5!=5
```

```
(True, False)
```

Escribe un programa que escriba por pantalla "No soy cinco" cuando la variable x no valga 5 y "Soy cinco" en otro caso

```
x=5
if x!=5:
    print('No soy cinco')
else:
    print('Soy cinco')
```

```
Soy cinco
```

```
x=7
if x==5:
    print('Soy cinco')
else:
    print('No soy cinco')
```

```
No soy cinco
```

El operador de comparación igual se escribe con doble igual (==)

ESTRUCTURAS DE CONTROL CONDICIONALES

if, elif, else

- La operación if - else también se puede escribir como una **operación ternaria** en una línea.

```
variable = valor1 if [condicion] else valor2
```

```
validador_mayor_edad = True if edad >= 18 else False

validador_mayor_edad

>> True
```

ESTRUCTURAS DE CONTROL CONDICIONALES

```
if condición1:
   instrucción1
elif condición2:
   instrucción2
elif condición3:
   instrucción3
...
else:
   instrucciónN
```

```
x=5

try:
    if x>5:
        print('Soy mayor que 5')
    elif x<5:
        print('Soy menor que 5')
    else:
        print('Soy igual a 5')
except:
    print('Número incorrecto')
```

```
Soy igual a 5
```

ESTRUCTURAS DE CONTROL ITERATIVAS

for

- Se utiliza para iterar sobre elementos de una secuencia (**string, tupla, lista**) u otro objeto **iterable** (se puede recorrer sus elementos uno a uno).

- Si la secuencia está vacía, el bucle no se ejecuta ninguna vez.

```
mi_string = "Juan S."

for letra in mi_string:
        print(letra)

>> 'J'
>> 'u'
>> 'a'
>> ...
```

ESTRUCTURAS DE CONTROL ITERATIVAS

for

- También se puede iterar usando la posición de cada elemento dentro de una secuencia.

```
mi_string = "Juan S."

for indice in range(len(mi_string)):

        print (indice, mi_string[indice])

>> (0, 'J')
>> (1, 'u')
>> ...
```

ESTRUCTURAS DE CONTROL ITERATIVAS

```
for index in [1,2,3]:
    print(index)
```

```
1
2
3
```

```
for index in range(4):
    print(index)
```

```
0
1
2
3
```

```
for index in range(3,6):
    print(index)
```

```
3
4
5
```

```
# Probemos ahora un bucle
for valor in range(5):
    print("Iteración número " + str(valor))
```

```
Iteración número 0
Iteración número 1
Iteración número 2
Iteración número 3
Iteración número 4
```

ESTRUCTURAS DE CONTROL ITERATIVAS

Escribe un programa que escriba por pantalla los múltiplos de 5 entre 1 y 100

```
for i in range(1,100):
    if i%5==0:
        print(i)
```

Escribe un programa que itere por los números de la siguiente lista y escriba la media de los valores que va iterando:
Lista_inicial=[0,1,2,3,4,5,6,7,8,9]

```
suma=0
num_elementos=0
for i in range(9):
    suma=suma+i
    num_elementos=num_elementos+1
    print("La media en la iteracion "+str(num_elementos)+" es "+str(suma/num_elementos))
```

ESTRUCTURAS DE CONTROL ITERATIVAS

while

- Se encarga de ejecutar una misma acción repetidamente, **mientras que** una determinada condición se cumpla; es decir, tengo el valor de verdadero o **True**.

```
while [condición]:
        [instrucción 1]
        [instrucción 2]
                . . .
        [instrucción n]
```

ESTRUCTURAS DE CONTROL ITERATIVAS

while

- Se encarga de ejecutar una misma acción repetidamente, **mientras que** una determinada condición se cumpla; es decir, tengo el valor de verdadero o **True**.

```
año = 1994
print ("Estos son los años entre el 1994 y el 2020")

while año <= 2020:
        print (año)
        año += 1

>> 2016
>> 2017
>> 2018
>> ...
```

ESTRUCTURAS DE CONTROL ITERATIVAS

```
# Bucle while

valor = 0
while (valor < 5):
    print("Iteración número " + str(valor))
    valor = valor + 1

# Fin del bucle
```

```
Iteración número 0
Iteración número 1
Iteración número 2
Iteración número 3
Iteración número 4
```

CONTROL DE ERRORES

```
x=7
if x>5:
    print('Soy mayor que 5')
else:
    print('Soy menor o igual que 5')
```

```
Soy mayor que 5
```

¿Qué ocurre si la variable de entrada X es un string?

```
x='Hola'
if x>5:
    print('Soy mayor que 5')
else:
    print('Soy menor o igual que 5')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-28-f06117991a0c> in <module>()
      1 x='Hola'
----> 2 if x>5:
      3     print('Soy mayor que 5')
      4 else:
      5     print('Soy menor o igual que 5')

TypeError: unorderable types: str() > int()
```

Recibimos un error y el programa se para.

¡No ejecuta el resto de sentencias!

CONTROL DE ERRORES

```
x='Hola'
try:
    if x>5:
        print('Soy mayor que 5')
    else:
        print('Soy menor o igual que 5')
except:
    print('Número incorrecto')
```

```
Número incorrecto
```

Es aconsejable que todos los programas que desarrollemos tengan control de errores para detectar las situaciones incorrectas que el usuario pueda generar.


gracias
merci
obrigado
danke
thank you
teşekkür ederim
sagolun
sukriya
hvala
dziękuje
arigatô
terima kasih
ngiyabonga
mochchakkeram
go raibh maith agat
bedankt
dank je
tapadh leat
takk
dakujem
kop khun krap
grazie
mauruuru
mamnun
dankon
dhanyavadagalu
tack
sobodi
djiere dieuf
merce
gracies
faafetai lava
xiexie
sulpay
kam sah hamnida
dankie
akun
nandri
gracie
barka
dhanyavad
kia ora
murakoze
najis tuke
dioloh
blagodaram
didi madloba
chokrane
taiku
obrigada
kiitos
mesi
misaotra
kaszonom
enkosi
vinaka
tanemirt
mahalo
dekuji
asante
grazzi
gratias ago
welalin
spas
tahmat
aciú
tau
tenki
bayarlalaa
matondo
chnorakaloutioun
tnugarez
mersi
manana
nanni
paldies