



Manual de Python

Materia: Ingeniería Artificial (I)

Estudiante:

- Marquez Fidel Alexander

Carrera:

Ing. Informática

Primer Paso

a. Introducción

Python es un lenguaje de programación de propósito general muy poderoso y flexible, a la vez que sencillo y fácil de aprender.

Las principales características de Python son las siguientes:

- Es multiparadigma, ya que soporta la programación imperativa, programación orientada a objetos y funcional.
- Es multiplataforma: Se puede encontrar un intérprete de Python para los principales sistemas operativos: Windows, Linux y Mac OS. Además, se puede reutilizar el mismo código en cada una de las plataformas.
- Es dinámicamente tipado: Es decir, el tipo de las variables se decide en tiempo de ejecución.
- Es fuertemente tipado: No se puede usar una variable en un contexto fuera de su tipo. Si se quisiera, habría que hacer una conversión de tipos.
- Es interpretado: El código no se compila a lenguaje máquina.

Un poco de historia:

Guido van Rossum ideó el lenguaje Python a finales de los 80 y comenzó a implementarlo en diciembre de 1989. En febrero de 1991 publicó la primera versión pública, la versión 0.9.0. La versión 1.0 se publicó en enero de 1994, la versión 2.0 se publicó en octubre de 2000 y la versión 3.0 se publicó en diciembre de 2008..

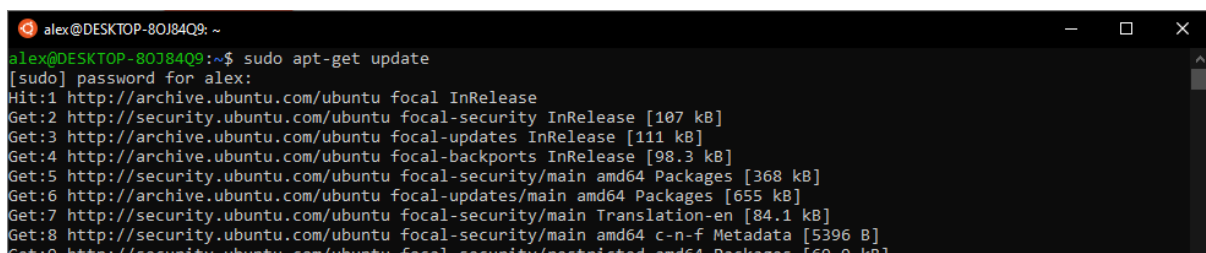
b. Instalación

El lenguaje al ser multiplataforma este puede ser instalado en cualquier sistema operativo. o al menos en los mas usados.

para poder instalar necesitamos bajar algunos paquetes. Estos los podemos encontrar en la pagina oficial <https://www.python.org/downloads/> . Además nos da una serie de pasos para poder instalar en las diferentes plataformas en las que es compatible. podemos notar en la pagina que este cuenta con varias versiones en este caso se usara la versión **3.5.2** para este manual de uso. a continuación podremos ver una breve explicación de como instalar python en diferentes sistemas operativos.

linux

\$ sudo apt-get update

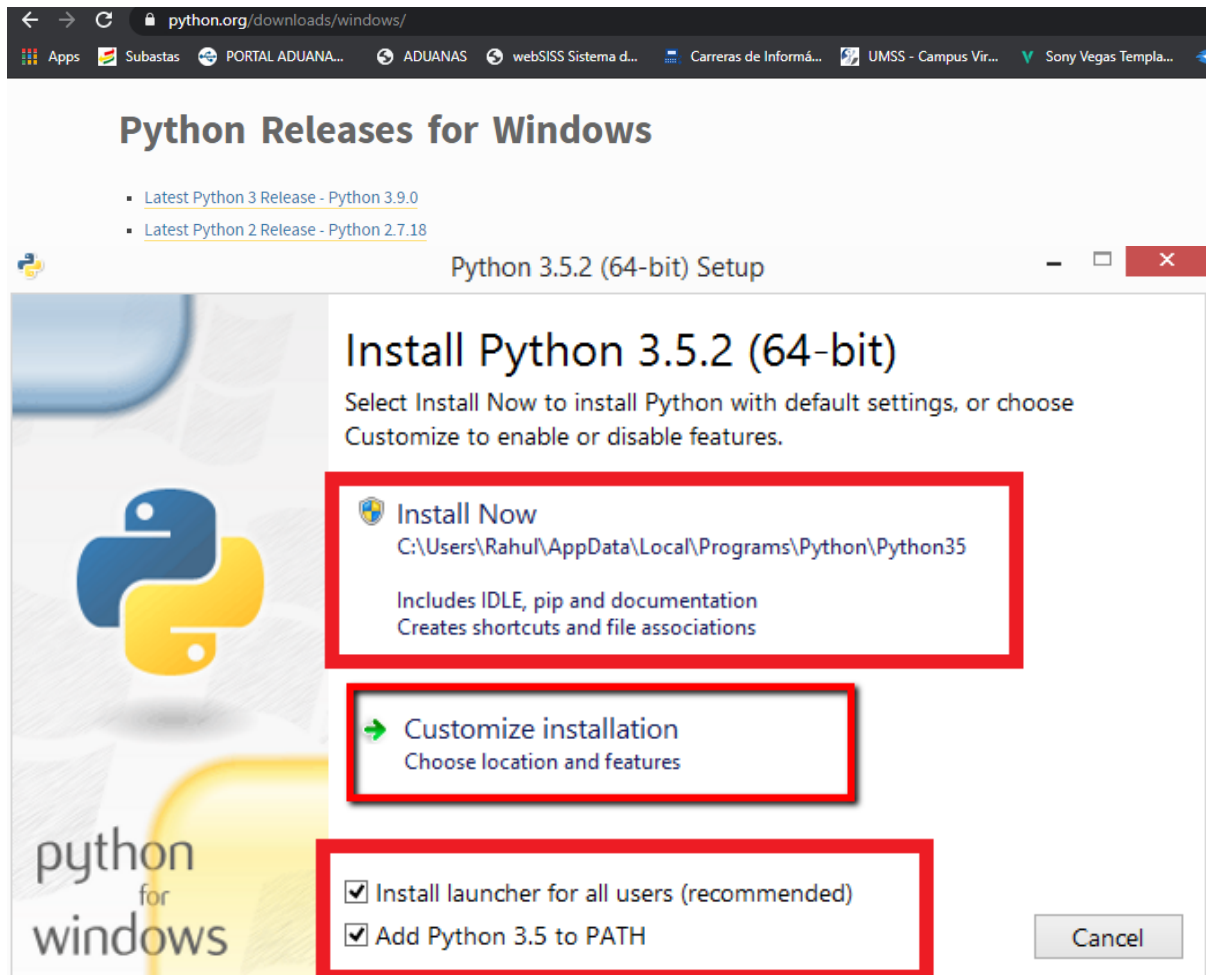


```
alex@DESKTOP-80J84Q9: ~  
alex@DESKTOP-80J84Q9:~$ sudo apt-get update  
[sudo] password for alex:  
Hit:1 http://archive.ubuntu.com/ubuntu focal InRelease  
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [107 kB]  
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [111 kB]  
Get:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease [98.3 kB]  
Get:5 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [368 kB]  
Get:6 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [655 kB]  
Get:7 http://security.ubuntu.com/ubuntu focal-security/main Translation-en [84.1 kB]  
Get:8 http://security.ubuntu.com/ubuntu focal-security/main amd64 c-n-f Metadata [5396 B]  
Get:9 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [69.9 kB]
```

\$sudo apt-get install python3.5

windows

Para windows lo que debemos hacer es descargar un archivo .exe para poder realizar una instalación mucho más fácil.



Una vez tengamos el archivo .exe podemos ejecutarlo. una vez ejecutado podemos seleccionar ADD PYTHON 3.5 TO PATH. esa opción nos agregara las variables de entorno en el sistema operativo y podremos ejecutar python desde cualquier directorio de nuestro sistema operativo.

Una vez instalado podemos ejecutar python en un terminal de windows y tendrá que lanzarnos el intérprete de python y nos dirá que versión estamos usando.

```
Command Prompt - python
C:\Users\Alex>python
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Como se puede apreciar en la imagen del terminal tenemos instalado python 3.5.2

Intérprete en línea

<https://repl.it/languages/python3>

<https://paiza.io/es/languages/python3>

https://www.onlinegdb.com/online_python_compiler

“Para continuar con el pequeño manual de python usaremos Jupyter Notebook es una aplicación web de código abierto que nos permite crear y compartir documentos que contienen código en vivo, ecuaciones, visualizaciones y texto narrativo. Los usos incluyen: limpieza y transformación de datos, simulación numérica, modelado estadístico, visualización de datos, aprendizaje automático y mucho más.”



Realizaremos la instalación de jupyter con las indicaciones de su propia pagina.

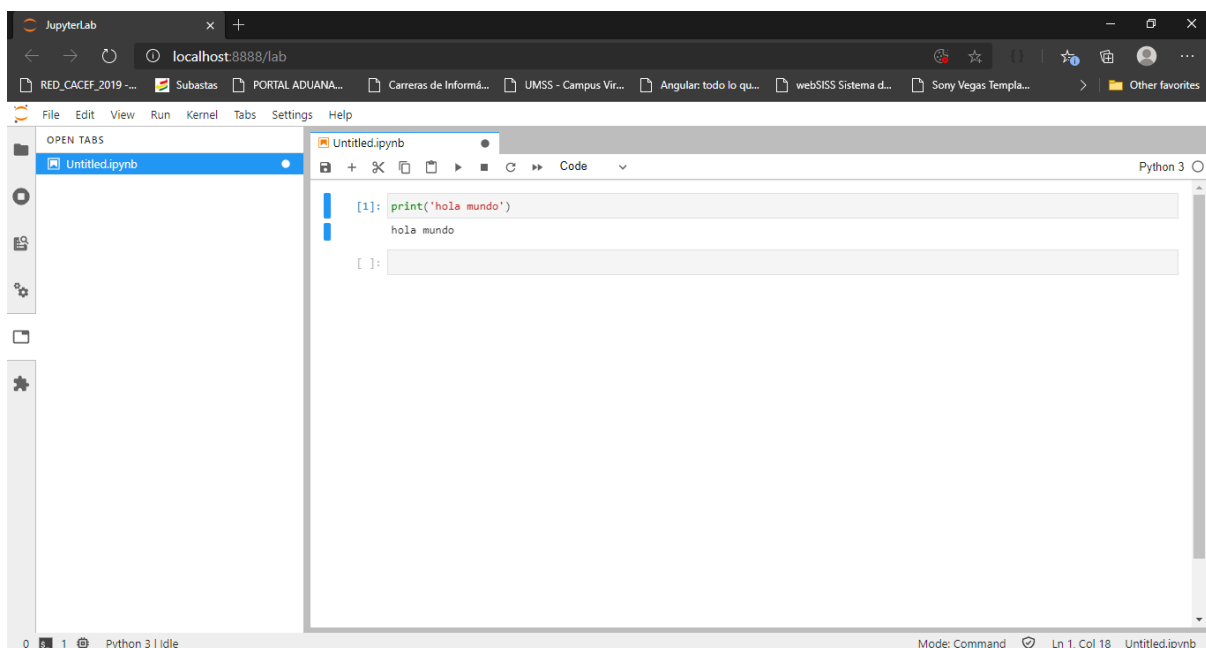
<https://jupyter.org/install>

Una vez instalado correremos el servicio con

> jupyter lab

```
Command Prompt - jupyter lab
C:\Users\Alex>jupyter lab
[I 13:52:02.895 LabApp] JupyterLab extension loaded from c:\users\alex\appdata\local\programs\python\python35-32\lib\site-packages\jupyterlab
[I 13:52:02.895 LabApp] JupyterLab application directory is c:\users\alex\appdata\local\programs\python\python35-32\share\jupyter\lab
[I 13:52:02.902 LabApp] Serving notebooks from local directory: C:\Users\Alex
[I 13:52:02.902 LabApp] Jupyter Notebook 6.1.5 is running at:
```

Una vez corriendo nuestro servicio lanzamos nuestro primer hola mundo. asi de simple con una línea de código y podemos ejecutarlo fácilmente en nuestro servicio.



Segundo Paso

Manual

“El manual se realizo en la aplicación web de júpiter. pasando todo los ejemplos y pruebas a un archivo .tex que es para archivos en latex”

Se adjunta el link del repositorio en git hub

Archivo latex .tex **MANUAL_PYTHON.tex**

Archivo **MANUAL_PYTHON.ipynb**

Archivo **MANUAL_PYTHON.PDF**

LINK:

<https://github.com/alexinf/IA-MANUAL-PYTHON.git>

MANUAL_PYTHON

November 15, 2020

[]: COMENTARIOS EN PYTHON:

[]: *#Los comentarios se puede realizar agregando un # al inicio de cada linea
""""tambien se puede comentar de esta forma
Varias lineas a la vez """*

[]: PALABRAS RESERVADAS EN PYTHON:

[1]: *#Las palabras reservadas en python pueden ser consultadas con la funcion help()
→en python
#A continuacion ejecutaremos el help("keywords") para obtener las palabras
→reservadas*

[2]: `help("keywords")`

Here is a list of the Python keywords. Enter any keyword to get more help.

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

[]: VARIABLES:

[10]: *""""En algunos lenguajes de programación, las variables se pueden entender como
→"cajas"
en las que se guardan los datos, pero en Python las variables son "etiquetas"
→que permiten
hacer referencia a los datos (que se guardan en unas "cajas" llamadas objetos).
→""""*

```
#las variables se inicializan al asignarles algun valor por ejemplo:
a = 10
b = 1.2
c = 'a'
#imprimiremos cada variable con lo siguiente
```

```
[10]: print('a: ',a , ' b: ', b, ' c: ', c)
```

```
a: 1 b: 1.2 c: a
```

```
[ ]: #Las variables deben estar "definidas" (con un valor asignado) antes
#de que puedan usarse, caso contrario pueden ocurrir errores.
```

```
[2]: print(x)
```

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-2-fc17d851ef81> in <module>
----> 1 print(x)

NameError: name 'x' is not defined
```

```
[ ]: #Un valor puede ser asignado a varias variables simultáneamente:
```

```
[4]: x = y = z = 632
print(x, ' ',y , ' ', z)
```

```
632 632 632
```

```
[ ]: OPERACIONES BASICAS:
```

```
[11]: suma = a + 2
resta = a - 8
multiplica = a * a
divide = a / 5
print('suma: ', suma)
print('resta: ', resta)
print('multiplicacion: ', multiplica)
print('divicion: ', divide)
```

```
suma: 12
resta: 2
```

```
multiplicacion: 100
divicion: 2.0
```

```
[ ]: #soporta completamente los números de punto flotante
```

```
[6]: 12*4.658*45.1
```

```
[6]: 2520.9096
```

```
[ ]: #Existe funciones de conversión
#de los nueros flotante y enteros (float(), int())
```

```
[15]: float(5)
```

```
[15]: 5.0
```

```
[16]: int(5.5)
```

```
[16]: 5
```

```
[ ]: #una funcion muy importante para saber el tipo de las variables Funcion type()
    ↳ muy util a la hora de programar para saber la
    #mutacion que pueda sufrir alguna variable en el proceso de ejecucion
    #al ser un lenguaje de programacion dinamica es muy necesario usar
    #la funcion type()
```

```
[17]: print(type(5.5))
```

```
<class 'float'>
```

```
[18]: print(type('a'))
```

```
<class 'str'>
```

```
[20]: print(type(5))
```

```
<class 'int'>
```

```
[22]: print(type([1,3,5]))
```

```
<class 'list'>
```

```
[ ]: CADENAS Y CARACTERES:
```

```
[24]: #Además de números, Python puede manipular cadenas de texto, las cuales
#pueden ser expresadas de distintas formas. Pueden estar encerradas en
#comillas simples o dobles:
```



```
[25]: "HOLA MUNDO"
```

```
[25]: 'HOLA MUNDO'
```

```
[27]: 'COMO ESTAN \'TODOS'
```

```
[27]: "COMO ESTAN 'TODOS"
```

```
[28]: '"Si," le dijo.'
```

```
[28]: '"Si," le dijo.'
```

```
[29]: "\"Si,\" le dijo."
```

```
[29]: '"Si," le dijo.'
```

```
[30]: text = "Este es un ejemplo de un parrafo que contiene\n\  
        varias lineas de texto \n\  
        Notar que los espacios en blanco al principio de la linea\  
son significantes."  
print(text)
```

```
Este es un ejemplo de un parrafo que contiene  
varias lineas de texto  
Notar que los espacios en blanco al principio de la linea son significantes.
```

```
[31]: #Las cadenas de texto pueden ser concatenadas  
#con el operador + y repetidas con *:
```

```
[48]: texto = "HOLA " + " AL MUNDO " + "DE "+"IA"
```

```
[49]: texto
```

```
[49]: 'HOLA  AL MUNDO DE IA'
```

```
[50]: '!' + texto*5 + '!'
```

```
[50]: '!HOLA  AL MUNDO DE IAHOLA  AL MUNDO DE IAHOLA  AL MUNDO DE IAHOLA  AL MUNDO DE  
IAHOLA  AL MUNDO DE IA!'
```

```
[51]: #Las cadenas de texto se pueden indexar; como en C,  
#el primer carácter de la cadena tiene el índice 0.
```

```
[52]: texto[0]
```

```
[52]: '0'
```

```
[53]: texto[0:2]
```

```
[53]: 'HO'
```

```
[54]: texto[0:4]
```

```
[54]: 'HOLA'
```

```
[56]: texto[0:8]
```

```
[56]: 'HOLA AL'
```

```
[57]: #Los índices de las rebanadas tienen valores por defecto útiles; el  
#valor por defecto para el primer índice es cero, el valor por defecto  
#para el segundo índice es la longitud de la cadena a rebanar.
```

```
[58]: texto[:4]
```

```
[58]: 'HOLA'
```

```
[62]: texto[18:]
```

```
[62]: 'IA'
```

```
[63]: #A diferencia de las cadenas de texto en C, en Python no  
#pueden ser modificadas. Intentar asignar a una posición de la  
#cadena es un error:
```

```
[64]: texto [0] = 'X'
```

TypeError

Traceback (most recent call last)

```
<ipython-input-64-23a303b3d04d> in <module>  
----> 1 texto [0] = 'X'
```

TypeError: 'str' object does not support item assignment

```
[65]: #Sin embargo, crear una nueva cadena con contenido  
#combinado es fácil y eficiente:
```

```
[66]: 'X' + texto[1:]
```

```
[66]: 'XOLA AL MUNDO DE IA'
```

```
[67]: '2020 ' + texto[0:]
```

```
[67]: '2020 HOLA AL MUNDO DE IA'
```

```
[68]: #Los índices pueden ser números negativos, para empezar a  
#contar desde la derecha. Por ejemplo:
```

```
[70]: texto[-1] #el ultimo caracter
```

```
[70]: 'A'
```

```
[71]: texto[-2] #el penultimo caracter
```

```
[71]: 'I'
```

```
[72]: #Los índices negativos fuera de rango son truncados
```

```
[75]: texto[-500:]
```

```
[75]: 'HOLA AL MUNDO DE IA'
```

```
[76]: #La función incorporada len() devuelve la longitud de una cadena  
#de texto:
```

```
[77]: len(texto)
```

```
[77]: 20
```

```
[ ]: """  
vease tambien las siguientes funciones  
typeseq: Las cadenas de texto y la cadenas de texto Unicode descritas en la_  
→siguiente sección son ejemplos de  
tipos secuencias, y soportan las operaciones comunes para esos tipos.  
string-methods: Tanto las cadenas de texto normales como las cadenas de texto_  
→Unicode soportan una gran cantidad  
de métodos para transformaciones básicas y búsqueda.  
new-string-formatting: Aquí se da información sobre formateo de cadenas de texto_  
→con str.format().  
string-formatting: Aquí se describe con más detalle las operaciones viejas para_  
→formateo usadas cuando una cadena de  
texto o una cadena Unicode están a la izquierda del operador %.  
"""
```

```
[ ]: LISTAS:
```

```
[83]: #Python tiene varios tipos de datos compuestos, usados para  
#agrupar otros valores. El más versátil es la lista.
```

```
[84]: #Es posible crear una lista que almacene solo números enteros:
```

```
[88]: lista_Numeros_Int = [0,1,2,3,4,5,6,7,8,9]
```

```
[89]: lista_Numeros
```

```
[89]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
[90]: #Es posible crear una lista que almacene solo números decimales:
```

```
[91]: lista_Numeros_Dec = [1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9]
```

```
[92]: lista_Numeros_Dec
```

```
[92]: [1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9]
```

```
[93]: #Es posible crear una lista que almacene números enteros,  
#números decimales y strings / cadenas de caracteres:
```

```
[95]: mi_lista_mix = [ 1.5, 2,"IA ESTA COOL"]
```

```
[96]: mi_lista_mix
```

```
[96]: [1.5, 2, 'IA ESTA COOL']
```

```
[99]: type(mi_lista_mix) # si imprimimos el tipo de dato podemos ver lo siguiente
```

```
[99]: list
```

```
[101]: #Es posible anidar listas  
#(crear listas que contengan otras listas), por ejemplo:
```

```
[103]: array1 = [1,2,3,4]  
array2 = ['x',array1,'y','z']  
array2
```

```
[103]: ['x', [1, 2, 3, 4], 'y', 'z']
```

```
[104]: array2.append('hola') #podemos insertar mas datos
```

```
[ ]: array2 # imprimimos el array
```

```
[ ]: CONDICIONALES:
```

[]: Condicionales (if,elif,else)

En python puedes definir una serie de condicionales utilizando:

- if: primera condicion
- elif: resto de las condiciones. Puede haber multiples elif.
- Else: se ejecuta cuando las anteriores condiciones son falsas.

[]: Operador ternario: El orden es diferente a otros lenguajes

```
'mayor que 2' if n > 2 else 'menor o igual a 2'
```

Puede haber cero o más bloques elif, y el bloque else es opcional. La palabra

→reservada 'elif' es

una abreviación de 'else if', y es útil para evitar un sangrado excesivo. Una

→secuencia if ... elif ...

elif ... sustituye las sentencias switch o case encontradas en otros lenguajes.

```
[113]: numero = 5
if numero < 5:
    print('el numero es menor a 5')
else:
    print('el numero es mayor a 5')
```

el numero es mayor a 5

[]: ITERACIONES:

[]: Nos permiten realizar el mismo bloque repetidas veces.

[]: LA SENTENCIA FOR

La sentencia for en Python difiere un poco de lo que uno puede estar

→acostumbrado en lenguajes como

C o Pascal.

```
[114]: #recorriendo un array
```

```
[118]: arra_for = [0,1,2,3,4,5,6,7]
for x in arra_for:
    print (x )
```

0
1
2
3
4
5
6
7

```
[119]: #generando un contador con un for lo que normalmente realizamos en
       #otros lenguajes de programacion
```

```
[122]: for x in range(0,12,2): # realizamos un generador de 2 en 2 hasta 10
       print(x)
```

```
0
2
4
6
8
10
```

[]: LA SENTENCIA WHILE
Es identico a otros lenguajes de programacion tiene una condicion la cual
→ consulta para poder continuar con el ciclo.

```
[124]: #se repite un bloque mientras la condicion logica devuelva True.
```

```
[125]: c=0
       while c<5:
           print(c)
           c+=1 #contador=contador+1
```

```
0
1
2
3
4
```

```
[131]: #Break
       c=0
       while c<5:
           c+=1
           if(c==4):
               print('Se corto el ciclo')
               break
           print(c)
```

```
1
2
3
Se corto el ciclo
```

```
[138]: #Continue
       c=0
       while c<5:
```

```

c+=1
if(c==3):
    continue
print(c)

```

1
2
4
5

[]: LA SENTENCIA PASS
La sentencia `pass` no hace nada. Se puede usar cuando una sentencia es requerida, pero por la sintaxis pero el programa no requiere ninguna acción.

[3]: *#UN EJEMPLO DE PASS SERIA AL DECLARAR ALGUNA FUNCION*

```

def funcion():
    pass

print('no realiza nada')

```

no realiza nada

[]: FUNCIONES:

[]: En python la definicion de funciones se realiza mediante la instrucción `def` mas un nombre de funcion descriptivo, seguido de parentesis de apertura y cierre. La definicion de la funcion finaliza con dos puntos(:).

```

def <nombre>(parametros):
    <cuerpo>

```

[4]:

```

def funcion():
    pass

```

[]: Para llamar o activar esta funcion ponemos el nombre de la funcion seguido de parentesis

```

<funcion()>

```

[5]: *#aqui aremos un ejemplo de una funcion normal
y una funcion recursiva*

```

def fact_1(n):
    factorial_total = 1
    while n > 1:
        factorial_total *= n
        n -= 1
    return factorial_total

```

```
def fact_recurividad(n):
    if n > 1:
        return n * fact_recurividad(n - 1)
    else:
        return 1
#forma sucia -_-
print(6 * 5 * 4 * 3 * 2 * 1)
#forma funcional
a = fact_1(6)
b = fact_recurividad(6)
print (a)
print (b)
```

720

720

720

[]: Decoradores
python ademas no ofrece varias formas de usar las funciones una de ellas serian los decoradores que suelen llegar a ser bastante utiles al momento de desarrollar algun proyecto

[]: *#aqui podemos ver un ejemplo*
def funcion_a(funcion_b):
 def funcion_c():
 Bloque de codigo
 funcion_b()
 return funcion_c

[]: en el ejemplo anterior podiamos ver un poco de recursividad
la recursividad puede aplicarse en este lenguaje de programacion
Python permite a una funcion llamarse a si mismo de igual forma que lo hace cuando llama a otra funcion

[8]: def fact_recurividad(n):
 if n > 1:
 return n * fact_recurividad(n - 1)
 else:
 return 1

b = fact_recurividad(6)
print(b)

720

[]: Manejo de excepciones
excepciones: errores detectados durante la ejecucion.

Como lo manejamos?

```
[15]: try:
      raise NameError('Hola')
except NameError:
    print ('Ha sucedido una excepción!')
    raise
```

Ha sucedido una excepción!

```
-----

NameError                                Traceback (most recent call last)

<ipython-input-15-d6817db9def1> in <module>
      1 try:
----> 2     raise NameError('Hola')
      3 except NameError:
      4     print ('Ha sucedido una excepción!')
      5     raise

NameError: Hola
```

[]: CLASES Y OBJETOS:

[]: Las clases introducen un poquito de sintaxis nueva, tres nuevos tipos de objetos, y algo de semántica nueva.

Casi todo en Python es un objeto, con sus propiedades y metodos.
Para crear una clase, utilizamos la palabra class.

```
[ ]: class Clase:
      <declaración-1>

      <declaración-N>
```

```
[17]: #mi primera clase
class Enteros:
    i = 12345
    def funcion(self):
        return i
```

```
[22]: x = Enteros()
      type(x)
```

```
[22]: __main__.Enteros
```

```
[ ]: Objetos clase
Para hacer referencia a atributos se usa la sintaxis estándar de todas las
    ↳referencias a atributos en
Python: objeto.nombre. Los nombres de atributo válidos son todos los nombres que
    ↳estaban en el
espacio de nombres de la clase cuando ésta se creó.
```

```
[24]: #Pondremos un ejemplo de clase Usando una de las de nuestro proyecto de
#busqueda recursiva primer el mejor.
#explicando cada funcion que pertenece a la clase.
```

```
[ ]: La clase Node tiene cinco métodos.
__init__(self, state, parent, action, path_cost) : Este método crea un nodo.
    ↳parent representa el nodo del que este es un sucesor y la action es la acción
    ↳necesaria para pasar del nodo padre a este nodo. path_cost es el costo para
    ↳llegar al nodo actual desde el nodo principal.
Los siguientes 4 métodos son funciones específicas relacionadas con el nodo.

expand(self, problem) : Este método enumera todos los nodos vecinos (accesibles
    ↳en un paso) del nodo actual.

child_node(self, problem, action) -> Dada una action, este método devuelve el
    ↳vecino inmediato que se puede alcanzar con esa acción.

solution(self) -> Esto devuelve la secuencia de acciones necesarias para llegar
    ↳a este nodo desde el nodo raíz.

path(self) : Esto devuelve una lista de todos los nodos que se encuentran en la
    ↳ruta desde la raíz hasta este nodo.
```

```
[26]: class Node:
      def __init__(self, state, parent=None, action=None, path_cost=0):
          self.state = state
          self.parent = parent
          self.action = action
          self.path_cost = path_cost
          self.f=0 #extra Variable representa el costo total
          self.depth = 0
          if parent:
              self.depth = parent.depth + 1
```

```

def __repr__(self):
    return "<Node {}>".format(self.state)

def expand(self, problem):
    return [self.child_node(problem, action)
            for action in problem.actions(self.state)]

def child_node(self, problem, action): # Creamos un nodo objeto de cada hijo.
    next_state = problem.result(self.state, action)
    new_cost = problem.path_cost(self.path_cost, self.state, action,
    ↪next_state)
    next_node = Node(next_state, self, action, new_cost )
    return next_node

def solution(self): # extrae el camino de la solución
    return [node.state for node in self.path()]

def path(self): # extrae la ruta de cualquier nodo desde el actual hasta el
    ↪origen
    node, path_back = self, []
    while node:
        path_back.append(node)
        node = node.parent
    return list(reversed(path_back))

```

[]: