

MODULOS Y PAQUETES

MODULOS

Que es y como se usa un modulo?

Importando un modulo

Namespace

Importando elementos de un modulo

Dos namespace coexistiendo

Importando entidades especificas de un modulo (importacion selectiva)

Importando un modulo con *

Importando un modulo cambiando su alias (as)

Trabajando con Modulos Estandar

La funcion dir()

Funciones seleccionadas del modulo math

Modulo random

Funcion random()

Funcion seed()

Funciones randrange y randint

Funciones choise() y sample()

Modulo platform

La funcion platform()

Funcion machine()

Funcion processor()

Funcion system()

Funcion version()

Funciones python_implementation y python_version_tuple

Indice de modulos

PAQUETES

Tu Primer Modulo

Paso 1

Paso 2

Paso 3

Paso 4

Paso 5

Paso 6

Paso 7

Paso 8

[Paso 9](#)

[Paso 10](#)

[Tu Primer Paquete](#)

[Paso 1](#)

[Paso 2](#)

[Paso 3](#)

[Paso 4](#)

[Paso 5](#)

[Paso 6](#)

[Paso 7](#)

[Ecosistema de Python](#)

[Instalador de Paquetes de Python PIP](#)

[Dependencias](#)

[Como usar PIP](#)

[Instalar paquetes](#)

[Ejemplo de instalacion de un paquete, pygame](#)

[Uso del paquete pygame](#)

[Otras funcionalidades de la sentencia pip install](#)

[Eliminar paquetes](#)

[Lista resumida de algunas actividades principales de pip](#)

MODULOS

El código tiene tendencia a crecer, el código que no crece es inutilizable o está abandonado. Un código real se desarrolla continuamente ya que tanto las demandas del usuario como sus expectativas aumentan. Nunca hay que pensar que los programas están terminados por completo.

Un código más grande significa un mantenimiento más difícil, la búsqueda de errores siempre es más fácil cuando el código es más pequeño.

Cuando el código que creamos se espera que sea grande (la cantidad de líneas nos da una idea aunque no del todo acertada) hay que dividirlo en partes más pequeñas. Si se desea que un proyecto de software se complete con éxito se debe tener los medios que permitan:

- Dividir todas las tareas entre los desarrolladores
- Después, unir todas las partes en un todo funcional

En un determinado proyecto se puede dividir en dos partes:

- Interfaz de usuario: parte que se cominca con el usuario mediante widgets y una pantalla grafica
- La logica: la parte que se encarga de procesar los datos y produce resultados

Cada una de esas partes se divide a sus vez en otras mas pequeñas, tal proceso se denomina DESCOMPOSICION. La divicion de una pieza de software en ppartes separadas pero cooperantes es posible gracias a los MODULOS

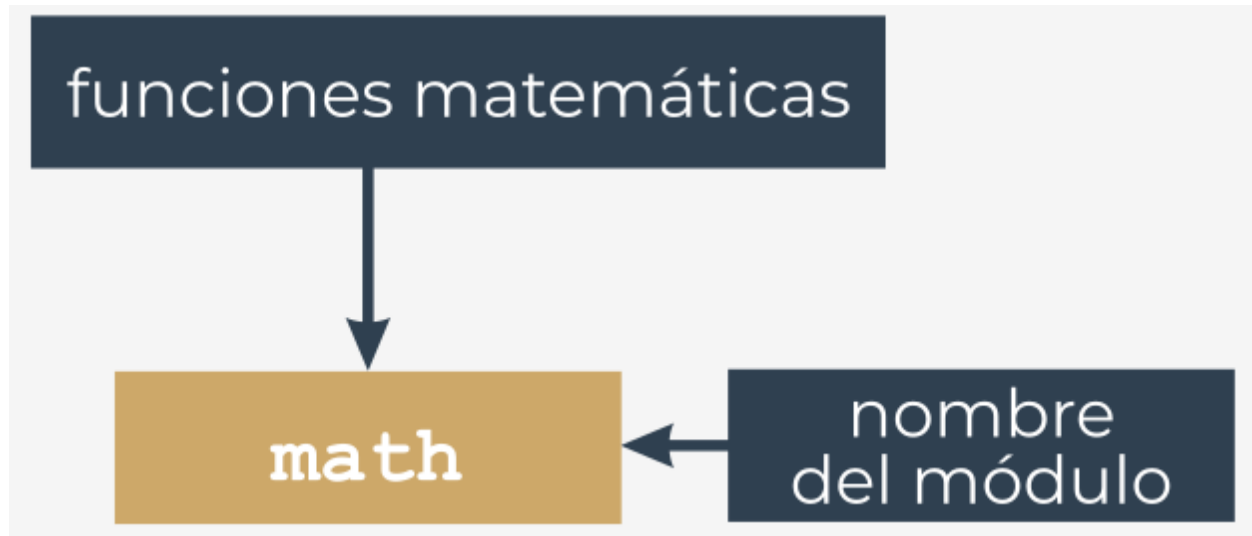
Que es y como se usa un modulo?

El tutorial de python define a un modulo como un archivo que contiene definiciones y sentencias de python q se pueden importar mas tarde y utilizar cuando sea necesario

El manejo de los modulos se podria decir que consta de dos personas distintas:

1. Usuario: suele ser el mas comun, ocurre cuando se desea utilizar un modulo ya existente escrito por otra persona o por ti mismo en algun proyecto, el programador en este caso es el usuario del modulo
2. Proveedor: es el programador que crea un nuevo modulo, ya sea para uso propio o para facilitar la vida de otros programadores, aqui el programador es el proveedor del modulo

Un modulo se identifica por su NOMBRE, si se quiere usar un modulo se necesita saber su nombre, con python (junto con las funciones) vienen intregrada una gran cantidad de modulos. Estos modulos junto con las funciones integradas forman la Biblioteca Estandar de Python, un tipo de biblioteca donde los modulos serian los libros. Cada modulo consta de entidades (funciones, variables, constantes, clases y objetos). Si se accede a un modulo en particular, se pueden usar todas sus entidades. Por ejemplo uno de los modulos mas usados es el que se llama “math” el cual contiene una coleccion de entidades q permiten implementar calculos de funciones matematicas como el `sen()` o `log()`



Importando un modulo

Para utilizar un modulo hay que importarlo (sacar el libro de la libreria), esto se realiza mediante la instruccion llamada "import".

Supongamos que necesitamos dos entidades del modulo `math`:

1. un simbolo constante que representa un valor tan preciso como sea posible usando un punto flotante doble de pi.
2. Una funcion llamada "`sen()`" que equivale a la funcion matematica seno

Ambas entidades estan disponibles a traves del modulo `math` pero la forma en que se usa depende de como se haya realizado la importacion.

La forma mas sencilla es usar la instruccion `import` de la siguiente manera:

```
import math
```

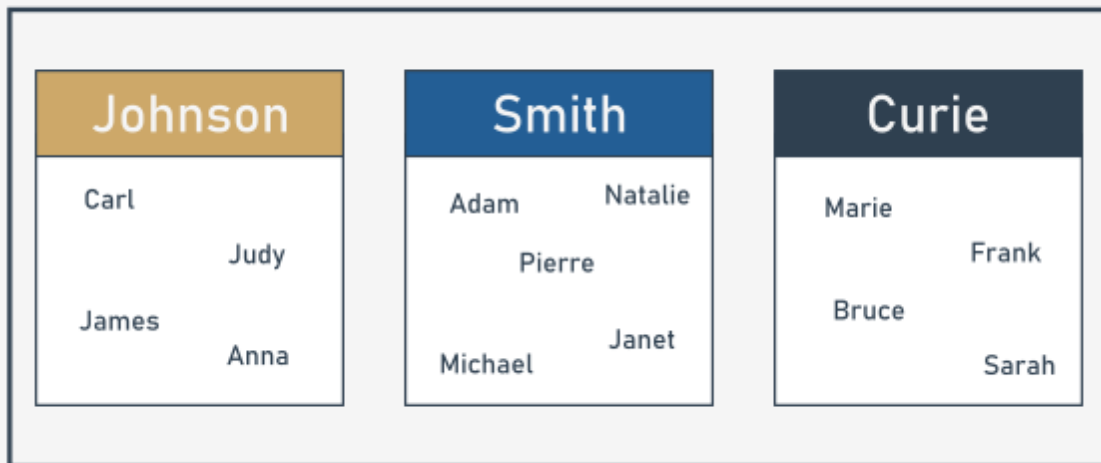
La clausula esta comprendida por la palabra reservada `import` y el nombre del modulo. Esta instruccion se puede poner en cualquier parte del codigo pero siempre antes del primer uso de cualquiera de las entidades del modulo

Si hay que importar mas de un modulo se puede hacer a traves de dos `import` o en uno solo separando el nombre de los modulos por comas

```
import math
import sys
import math, sys
```

Namespace

Un namespace es un espacio en el que existen algunos nombres y estos no entran en conflicto entre si (no hay dos objetos diferentes con el mismo nombre). Se puede decir que cada grupo social es un namespace en el cual ese grupo tiende a nombrar a sus de manera distinta, por ejemplo un padre no nombra a sus hijos con el mismo nombre



Dentro de un determinado namespace cada nombre debe permanecer unico, esto puede significar que algunos nombres pueden desaparecer cuando cualquier otra entidad de un nombre ya conocido ingresa al namespace

Si el modulo de un nombre especifico existe y es accesible python importa su contenido y se hacen conocidos todos los nombres definidos en el modulo pero no ingresa en el namespace del codigo. Esto significa que tener tus propias entidades llamadas sin o pi no seran afectadas en alguna manera por la importacion

Entonces... ¿Como accedemos al pi que viene dentro del modulo math? Para hacer esto se debe llamar el pi con el nombre del modulo original

Importante elementos de un modulo

Esta es la forma correcta en la que se habilitan los nombres de pi y sin con el nombre de su modulo de origen

```
math.pi  
math.sin
```

Esta sentencia esta compuesta de : El nombre del modulo; un punto; El nombre de la entidad que pertenece al modulo

El uso de esto es obligatorio si un modulo ha sido importando con la instruccion import, no importa si alguno de los nombres del codigo y del namespace del modulo estan en conflicto o no

Dos namespace coexistiendo

En este ejemplo se ven dos namespace, el tuyo y el del modulo, coexistiendo.

Se define nuestro propio pi y sin y luego invocamos los del modulo, estas entidades no se afectan entre si

```
import math  
  
def sin(x):  
    if 2*x==pi:  
        return 0.999999  
    else:  
        return none  
  
pi=3.14  
  
print(sin(pi/2))  
print(math.sin(math.pi/2))
```

La salida esperada es:

```
0.9999  
1.0
```

Importando entidades especificas de un modulo (importacion selectiva)

La siguiente instruccion señala con presicion que entidad/es del modulo son aceptables en el codigo:

```
from math import pi, sin
```

Esta instruccion conta de: “from”+nombre del modulo a ser selectivamente importado+ “import”+ el nombre o lista de nombres de las entidades las cuales estan siendo importadas al namespace

Las entidades listadas son las unicas q son importadas del modulo indicado, Los nombres de las entidades importadas pueden ser accedidas dentro del codigo sin especificar el nombre del modulo de origen

```
from math import pi, sin
print(sin(pi/2))
```

La salida esperada seria:

```
1.0
```

En el siguiente fragmento de codigo se muestra una importacion selectiva y a la vez luego se redefine esas entidades dentro del namespace del codigo:

```
from math import pi, sin

print (sin(pi/2))
"""La salida esperada aca seria 1.0"""

pi=3.14
def sin(x):
    if 2*x==pi:
        return 0.9999
    else:
        return None

print(sin(pi/2))
"""La salida esperada aca seria 0.9999"""
```

En las primeras lineas se hace la importacion selectiva, luego se usa las entidades importadas, luego se redefine el significado de "sin" y "pi" osea que reemplazan las definiciones originales importadas dentro del namespace del codigo

Importando un modulo con *

La forma:

```
from modulo import *
```

Permite que todas las entidades del modulo indicado sean importadas, es conveniente ya que nos libera de tener que escribir una por una; pero ,a menos que sepas todos los nombres proporcionados por el modulo, es posible que se generen conflictos de nombre

Importando un modulo cambiando su alias (as)

Si se quiere importar un modulo pero su nombre no nos conviene (porque es el mismo que una entidad ya defina) se le puede dar otro nombre renombrando .

La creacion de un alias se hace junto con la importacion del modulo y de la siguiente manera

```
import module as alias
```

Ejemplos:

```
import math as m
print(m.sin(m.pi/2))
```

Después de importar un modulo con un alias el nombre original del modulo se vuelve inaccesible y no debe ser utilizado.

El mismo renombrado puede ser usado para darle un alias a una entidad. por eje

```
from modulo import name as alias
from module import name1 as alias1, name2 as alias2
```


El nombre original de la entidad se vuelve inaccesible

Trabajando con Modulos Estandar

La funcion dir()

La funcion dir() puede revelar todos los nombres proporcionados a traves de un modulo en particular. Existe la condicion de que el modulo al cual se le aplique la funcion dir() se lo debe de haber importado completamente (import module).

La funcion devuelve una lista ordenada alfabeticamente la cual contiene todos los nombres de las entidades disponibles en el modulo. Si el modulo tiene un alias se debe usar este y no el nombre

Usar la funcion dentro de un script no es comun pero se puede hacer asi:

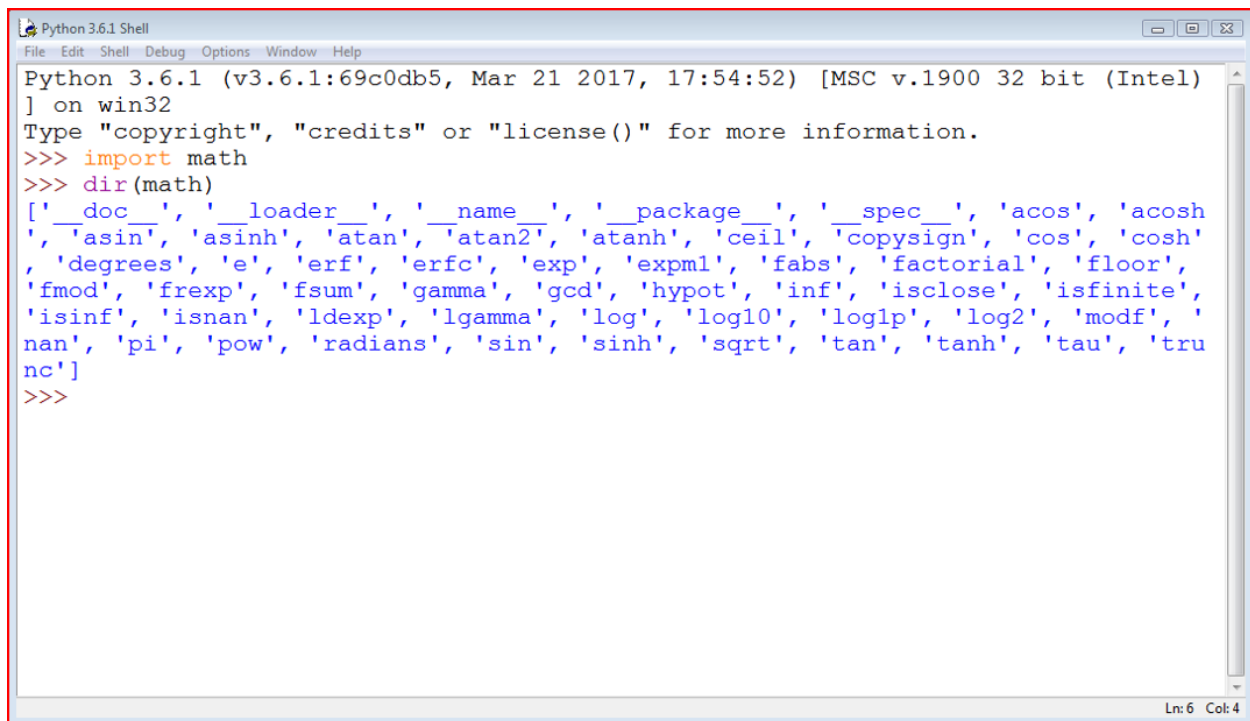
```
import math

for name in dir(math):
    print(name, end="\t")
```

Lo normal es que se desea conocer el contenido e un modulo antes de escribir y ejecutar el codigo. Se púede ejecutar la funcion directamente en la consola de python sin tener que escribir un script por separado, asi es como se puede hacer:

```
import math
dir(math)
```

Se deberia ver algo similar a esto



```
Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import math
>>> dir(math)
['_doc_', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
>>>
```

Funciones seleccionadas del modulo math

En el siguiente ejemplo se eligieron algunas entidades arbitrarias

```
from math import pi, radians, degrees, sin, cos, tan, asin

ad = 90
ar = radians(ad)
ad = degrees(ar)

print(ad == 90.)
print(ar == pi / 2.)
print(sin(ar) / cos(ar) == tan(ar))
print(asin(sin(ar)) == ar)
```

En el primer grupo estan las funciones relacionadas con trigonometria (sin, cos, tang). Para trabajar eficientemente con mediciones de angulos se usa las entidades:

- `radians(x)`: una funcion que transforma x de grados a radianes
- `degrees(x)`: una funcion que convierte x de radianes a grados

Otro grupo de funciones relacionado con la exponenciacion son:

- `e`: constante q es una aproximacion del numero de euler (e)

- `exp(x)`: encontrar el valor de e a la x
- `log(x)`: el logaritmo natural de x
- `log(x,b)`: logaritmo de x con base en b
- `log10(x)`: logaritmo decimal de x (mas preciso que `log(x,10)`)

Modulo random

Este modulo ofrece mecanismos q permiten operar con numeros pseudoaleatorios. Los numeros generados por los modulos pueden parecer aleatorios pero no hay que olvidar que todos se calculan utilizando algoritmos muy refinados, y los algoritmo no son aleatorios, son determinados y predecibles. Solo los procesos fisicos que salgan completamente de nuestro control (como la intensidad de la radiacion cosmica) pueden usarse como fuente de datos aleatorios

Un generador de numeros aleatorios toma un valor llamado “semilla”, lo trata como un valor de entrada basado en el y produce una nueva semilla. La duracion de un ciclo en el que todos los valores semilla sean unicos puede ser muy largo, pero no es infinito, tarde o temprano todos los valores iniciales comenzara a repetirse y los valores generados tambien lo haran. El factor aleatorio del proceso puede ser aumentado al establecer la semilla tomando un numero de la hora actual , esto puede garantizar que cada ejecucion del programa comience desde un valor semilla diferente.

Python realiza esta inicializacion al importar el modulo

Funcion random()

La funcion “`random()`” no debe ser confundida con el nombre dl modulo. Esta funcion produce un numero flotante x entre el rango 0.0 y 1.0 ($0.0 < x < 1$). El siguiente fragmento produce 5 valores pseudoaleatorios ya que sus valores estan determinados por el valor de la semilla actual

```
from random import random

for i in range(5):
    print(random())
```

Funcion seed()

Es capaz de establecer la semilla del generador, hay dos maneras:

- `seed()`: establece la semilla con la hora actual
- `seed(int_value)`: establece la semilla con el valor entero `int_value`

En el siguiente script el hecho de que la semilla se establece con el mismo valor la secuencia de valores generados es siempre el mismo

```
from random import random, seed

seed(0)

for i in range(5):
    print(random())
```

Funciones `randrange` y `randint`

Si se desea valores enteros aleatorios se usan las siguientes funciones:

- `randrange(fin)`
- `randrange(inicio, fin)`
- `randrange(inicio, fin, incremento)`
- `randint(izquierda,derecha)`

Las tres primeras invocaciones generan un valor entero tomado segun el rango que corresponda teniendo en cuenta la exclusion implicita del lado derecho.

La ultima genera el valor entero i el cual cae en el rango (izquierda-derecha). Por ejemplo

```
from random import randrange, randint
print(randrange(1), end=' ')
print(randrange(0, 1), end=' ')
print(randrange(0, 1, 1), end=' ')
print(randint(0, 1))
```

Esto generara una salida de tres 0 y en la ultima linea se generara un 1 o un 0

Estas funciones tienen la desventaja de que pueden producir valores repetidos incluso si el numero de invocaciones posteriores no es mayor que el rango especificado. Por

ejemplo el siguiente script es posible que genere un conjunto de numeros en el que algunos elementos no sean unicos

```
from random import randint
for i in range(10):
    print(randint(1, 10), end=',')
```

Funciones `choise()` y `sample()`

Son una solucion a escribir tu propio codigo para verificar la singularidad de los numeros sorteados

- `choise(sequencia)`: esta funcion elige un numero aleatorio de la secuencia de entrada y lo devuelve
- `sample(sequencia, elementos_a_eleguir=1)`: esta crea una lista que consta del elemento “elementos_a_seguir” (q por defecto es 1) “sorteado” de la secuencia de entrada

La funcion elige algunos de los elementos de entrada y devuelve una lista con la eleccion, los elementos de la muestra se colocan en orden aleatorio. La parte de `elementos_a_seguir` no debe ser mayor q la longitud de la secuencia de entrada

Por ejemplo:

```
from random import choise, sample
my_list=[1,2,3,4,5,6,7,8,9,10]
print(choise(my_list))
print(sample(my_list, 5))
print(sample(my_list, 10))
```

La salida del programa no es predecible

Modulo platform

A veces es necesario saber la ubicacion de tu programa dentro del entorno de la computadora. El entorno de nuestra computadora se puede entender como una piramide que consta de varias capas:



- El código en ejecución
- El entorno de ejecución del código se encuentra debajo de él
- Luego el SO, el entorno de Python proporciona funciones utilizando los servicios del sistema operativo
- Hardware: el procesador, las interfaces de red, los dispositivos de interfaz humana y toda la maquinaria para hacer funcionar la computadora

Las acciones que tiene que recorrer el programa para ejecutarse con éxito son: Tu código quiere crear un archivo, por lo que invoca una de las funciones de Python. Python acepta la orden y como que le pone el sello de aprobado. El SO comprueba si la solicitud es aceptable y válida e intenta crear el archivo. El hardware es el que activa los dispositivos de almacenamiento (disco duro, disco de estado sólido)

Algunas veces lo que queremos es saber el nombre del sistema operativo que aloja Python.

El módulo `platform` permite acceder a los datos de la plataforma subyacente, es decir, el hardware, software, SO y la versión del intérprete

La función `platform()`

Esta funcion muestra todas las capas subyacentes. Se invoca de la siguiente manera:

```
platform(aliased = False, terse = False)
```

- `alias` → cuando se establece `True` o un valor distinto a 0, puede hacer que la funcion presente los nombres de capa subyacentes alternativos en lugar de los comunes
- `terse` → cuando se establece `True` o cualquier valor distinto a 0, puede convencer a la funcion de presentar una forma mas breve del resultado si es posible

Ejecutando:

```
from platform import platform

print(platform())
print(platform(1))
print(platform(0, 1))
```

Se produce:

```
Windows-10-10.0.19044-SP0
Windows-10-10.0.19044-SP0
Windows-10
```

Funcion `machine()`

Sirve para conocer el nombre del procesador que ejecuta el sistema operativo

```
from platform import machine

print(machine())
```

Funcion `processor()`

Devuelve una cadena con el nombre del procesador si es posible

```
from platform import processor
```

```
print(processor())
```

Funcion system()

Devuelve el nombre generico del sistema operativo en una cadena

```
from platform import system  
  
print(system())
```

Funcion version()

Permite conocer la version del sistema operativo

```
from platform import version  
  
print(version())
```

Funciones python_implementation y python_version_tuple

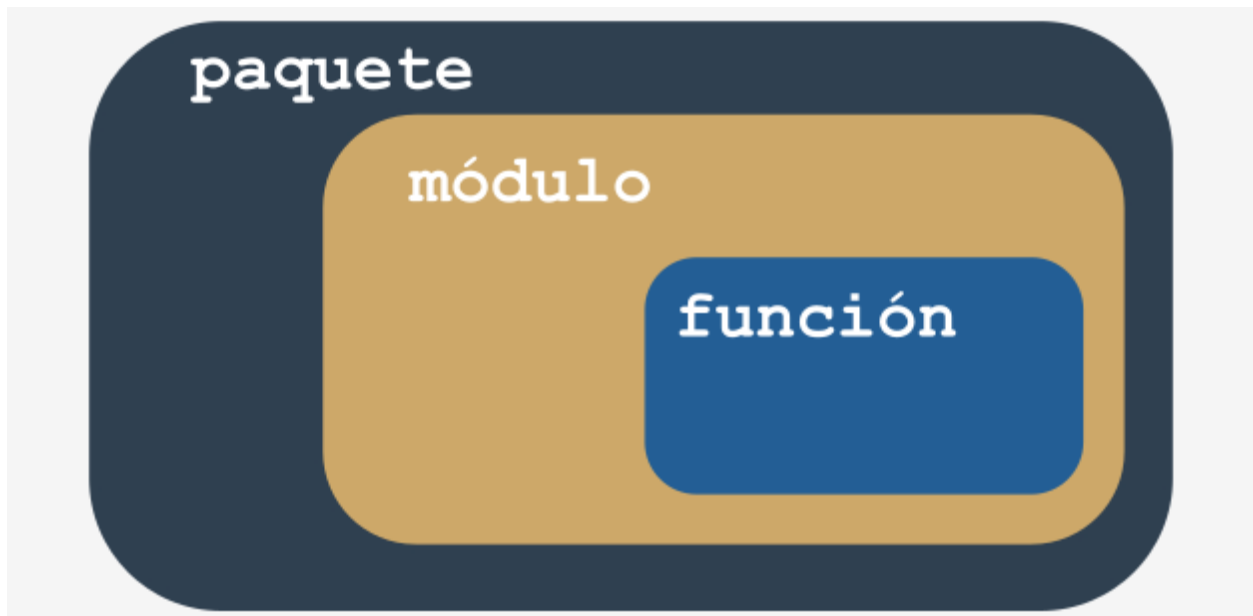
Si necesitas saber que versión de Python está ejecutando tu código, puedes verificarlo utilizando una serie de funciones dedicadas, aquí hay dos de ellas:

- `python_implementation()` → devuelve una cadena que denota la implementación de Python (espera `CPython` aquí, a menos que decidas utilizar cualquier rama de Python no canónica).
- `python_version_tuple()` → devuelve una tupla de tres elementos la cual contiene:
 - La parte **mayor** de la versión de Python.
 - La parte **menor**.
 - El número del nivel de **parche**.

Indice de modulos

Son muchisimos los modulos hechos por los programadores. Modulos muy especificos como la astrologia por ejemplo, aunque estos no vienen integrados con python

PAQUETES



Un modulo es un contenedor lleno de funciones, se pueden empaquetar tantas funciones como se desee en un modulo y distribuirlo.

Crear muchos modulos puede causar problemas de desorden, por lo que se querra agrupar estos modulos. Para esto sirven los PAQUETES, tiene un papel similar al de una carpeta o directorio

Tu Primer Modulo

Paso 1

Ahora trabajaremos localmente en nuestra maquina. Se necesitan dos archivos para realizar lo que vamos a hacer, uno de ellos sera el modulo en si y el otro contiene el codigo que utiliza el nuevo modulo

- Hay que crear un archivo del modulo que por el momento estara vacio que se llamara "module.py"

Paso 2

Crear el archivo `main.py` el cual “inicializara” o utiliza el nuevo modulo. Se debe llamar `main.py` y debe de estar en la misma carpeta que el `module.py`. Debe contener:

```
import module
```

Al inicializar el IDLE en el archivo `main.py` no pasa nada, pero python importo con exito el contenido del archivo `module.py`. En la carpeta donde se encuentran los dos archivos se agrego una carpeta llamada “pycache” en la cual hay un archivo llamado `module.cpython-xy.pyc` el cual x e y son digitos que corresponden a tu version de Python(por ejemplo el 3 y 10).

La parte posterior al primer punto dice que implementacion de python ha creado el archivo (CPython) y su numero de version. La extension `pyc` viene de las palabras Python y compilado

Cuando python importa un modulo por 1ra vez traduce el contenido a una forma compilada imposible de leer para los humanos

Paso 3

Colocamos algo en al archivo del modulo como:

```
print("Hola mundo soy un modulo")
```

Paso 4

Si ejecutamos ahora el archivo `main` observamos que si lo ejecutamos se imprime el contenido del modulo que escribimos anteriormente

Cuando un modulo es importado su contenido es ejecutado por python, se inicializa. La inicializacion se realiza solo una vez, cuando se produce la primera importacion.

Por ejemplo se tiene un modulo `modulo1` y otro llamado `modulo2` que contiene la instruccion `import modulo1`. Hay un archivo principal que contiene las intruccion `import modulo1` y `import modulo 2`, parece que `modulo1` se importa dos veces, pero solo lo hace una vez. Python recuerda los modulos importaos y omite todas las importaciones posteriores

Paso 5

Python crea una variable llamada “__name__” en la cual cada archivo fuente usa su propia versión separada de la variable, no se comparten entre módulos. Si modificamos el archivo module.py

```
print("Hola mundo soy un modulo")
print("__name__")
```

Si lo ejecutamos se debería ver:

```
Hola mundo soy un modulo
__main__
```

Si se ejecuta en main.py se ve:

```
Hola mundo soy un modulo
module
```

O sea que:

- Si se ejecuta en el archivo directamente su variable __name__ se establece a **main**
- cuando un archivo se importa como un módulo, su variable __name__ se establece al nombre del archivo

Paso 6

A diferencia de otros lenguajes en Python no hay forma de ocultar variables a los usuarios del módulo, solo se puede informar a tus usuarios que esta es tu variable y que por lo tanto no deben modificarla de ninguna manera. Esto se hace anteponiendo al nombre de la variable “_” pero esto solo es un acuerdo, los usuarios lo pueden respetar o no

Paso 7

Introduciendo código en el archivo module.py:

```
#!/usr/bin/env python3

""" module.py - Un ejemplo de un módulo en Python """

__counter = 0

def suml(the_list):
    global __counter
    __counter += 1
    the_sum = 0
    for element in the_list:
        the_sum += element
    return the_sum

def prodl(the_list):
    global __counter
    __counter += 1
    prod = 1
    for element in the_list:
        prod *= element
    return prod

if __name__ == "__main__":
    print("Yo prefiero ser un módulo, pero puedo realizar algunas pruebas por ti")
    my_list = [i+1 for i in range(5)]
    print(suml(my_list) == 15)
    print(prodl(my_list) == 120)
```

- La línea que comienza con `#!` desde el punto de vista de Python es solo un comentario ya que comienza con `#`, pero en algunos SO esta línea indica al sistema operativo como ejecutar el contenido del archivo (q programa debe ejecutarse para interpretar el texto).
- La cadena colocada antes de las instrucciones de un módulo es el DOCSTRING y debe explicar brevemente el propósito y el contenido del módulo
- Las funciones dentro del módulo `suml()` y `prodl()` pueden ser importadas
- Se usa la variable `"__name__"` para detectar cuando se ejecuta el archivo de forma independiente

Paso 8

Es posible usar el nuevo módulo, esta es una forma de hacerlo:

```
from module import suml, prodl

zeroes = [0 for i in range(5)]
ones = [1 for i in range(5)]
print(suml(zeroes))
print(prodl(ones))
```

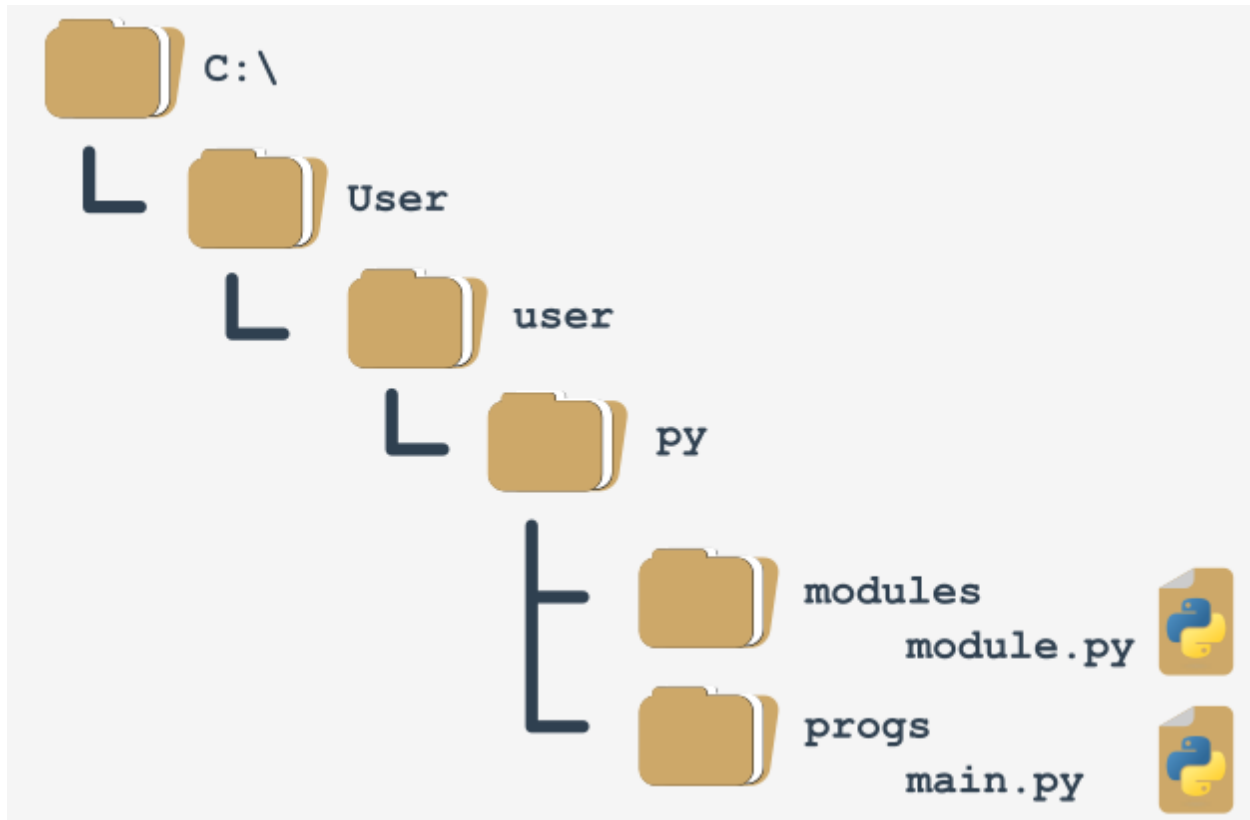
Lo cual produce la siguiente salida:

```
0
1
```

Paso 9

Asumimos que el archivo principal esta en la misma carpeta que el archivo del modulo q se va a importar. ¿Q pasa si no lo estan?

Supongamos q el SO es windows , que el script principal se encuentra en C:\Users\user\py\progs y se llama main.py, y el modulo a importar se encuentra en: C:\Users\user\py\modules



Python tiene una variable (en realidad es una lista) q almacena todas las ubicaciones que se buscan para encontrar un modulo que ha sido solicitado por la instruccion import, Python examina esas carpetas y si el modulo no se encuentra en ninguna la importacion falla.

De lo contrario, se tomara en cuenta la primera carpeta que contenga un modulo con el nombre deseado

La variable se llama path (ruta) y es accesible a travez del modulo llamado sys. Asi es como se verifica su valor

```
import sys

for p in sys.path:
    print(p)
```

Si se ejecuta en el ejemplo que vemos obtenemos:

```
C:\Users\user
C:\Users\user\AppData\Local\Programs\Python\Python36-32\python36.zip
```

```
C:\Users\user\AppData\Local\Programs\Python\Python36-32\DLLs  
C:\Users\user\AppData\Local\Programs\Python\Python36-32\lib  
C:\Users\user\AppData\Local\Programs\Python\Python36-32  
C:\Users\user\AppData\Local\Programs\Python\Python36-32\lib\site-packages
```

La carpeta en la que comienza la ejecución aparece en el primer elemento de la ruta

Paso 10

Una de las varias soluciones se ve así:

main.py

```
from sys import path  
  
path.append('..\modules')  
  
import module  
  
zeroes = [0 for i in range(5)]  
ones = [1 for i in range(5)]  
print(module.sum1(zeroes))  
print(module.prod1(ones))
```

Tu Primer Paquete

Paso 1

Suponiendo q hacemos una gran cantidad de funciones en modulos separados tal que asi:

alpha.py

```
#!/usr/bin/env python3

""" module: alpha """

def funA():
    return "Alpha"

if __name__ == "__main__":
    print("Prefiero ser un módulo.")
```

beta.py

```
def funB(): ...
```

iota.py

```
def funI(): ...
```

sigma.py

```
def funS(): ...
```

tau.py

```
def funT(): ...
```

psi.py

```
def funP(): ...
```

omega.py

```
def funO(): ...
```

Supongamos que todos los modulos tienen un aspecto similar al que tiene alpha.py q se expresa en la imagen

Paso 2

Estos modulos tienen su propia jerarquia, por lo que los modulos se agrupan. Por ejemplo segun la siguiente agrupacion:

group: extra

```
module: iota.py
```

```
def funI(): ...
```

group: good

```
module: alpha.py
```

```
def funA(): ...
```

```
module: beta.py
```

```
def funB(): ...
```

group: best

```
module: sigma.py
```

```
def funS(): ...
```

```
module: tau.py
```

```
def funT(): ...
```

group: ugly

```
module: psi.py
```

```
def funP(): ...
```

```
module: omega.py
```

```
def funO(): ...
```

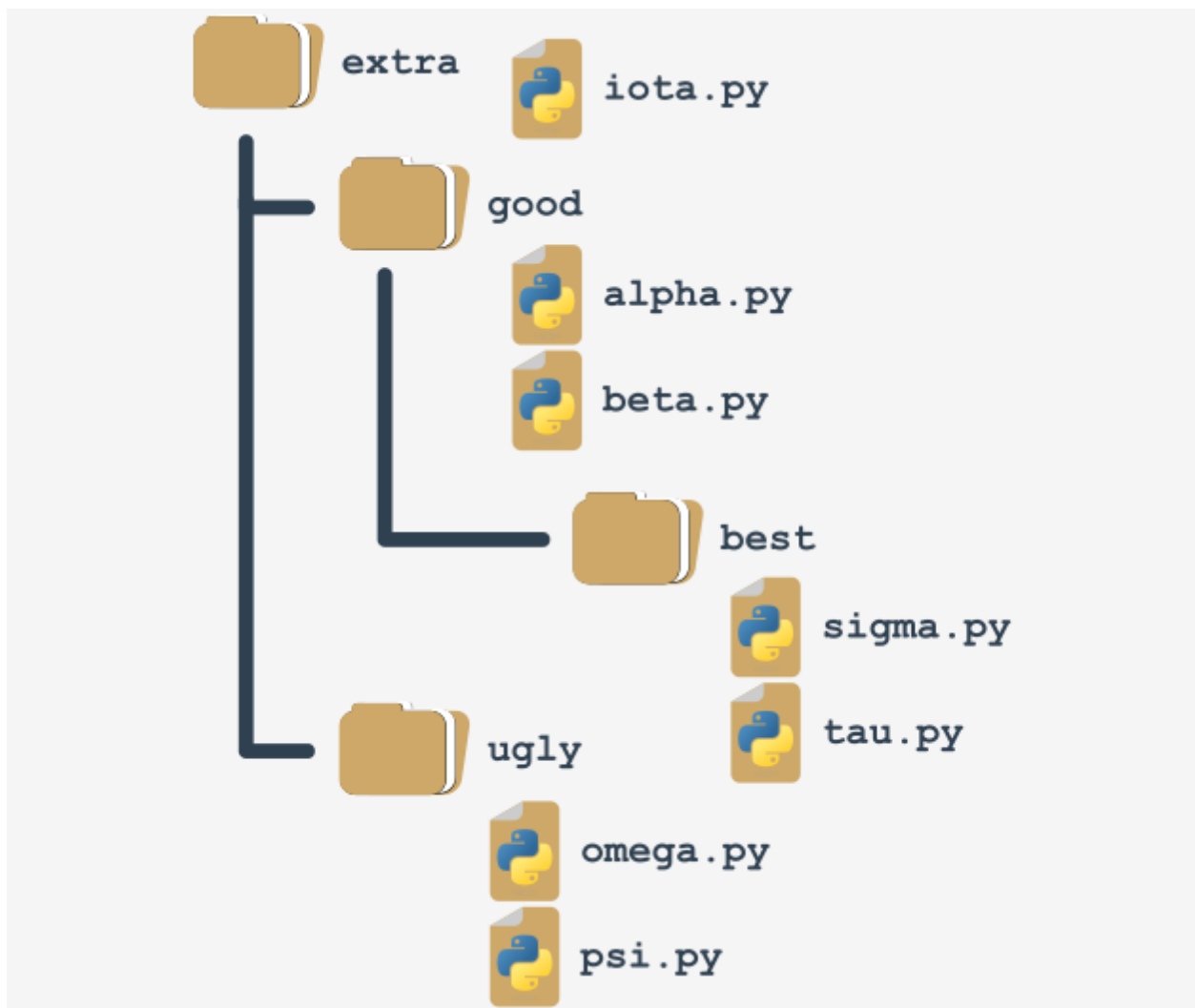
De arriba hacia abajo:

- El grupo extra contiene el modulo iota.py y dos subgrupos good y best
- El grupo good esta compuesto de los modulos alpha.py y beta.py y por el subgrupo best
- El grupo best esta compuesto por los modulos sigma.py y tau.py
- El grupo ugly esta constituido de dos modulos pai.py y omega.py

Todo esto parece la estructura de un directorio

Paso 3

Asi se veria como directorio de carpetas



Se puede pensar que “extra” como la raíz del paquete el cual impondra una regla de nomenclatura que te permita nombrar cada entidad. Por ejemplo la ubicacion de la funcion llamada funT() del paquete tau puede describirse como:

```
extra.good.best.tau.funT()
```

Paso 4

Los paquetes , como los modulos, pueden requerir inicializacion. La inicializacion de un modulo es a traves de un codigo ubicado dentro del archivo del modulo. Como un paquete no es un archivo lo que se hace es :

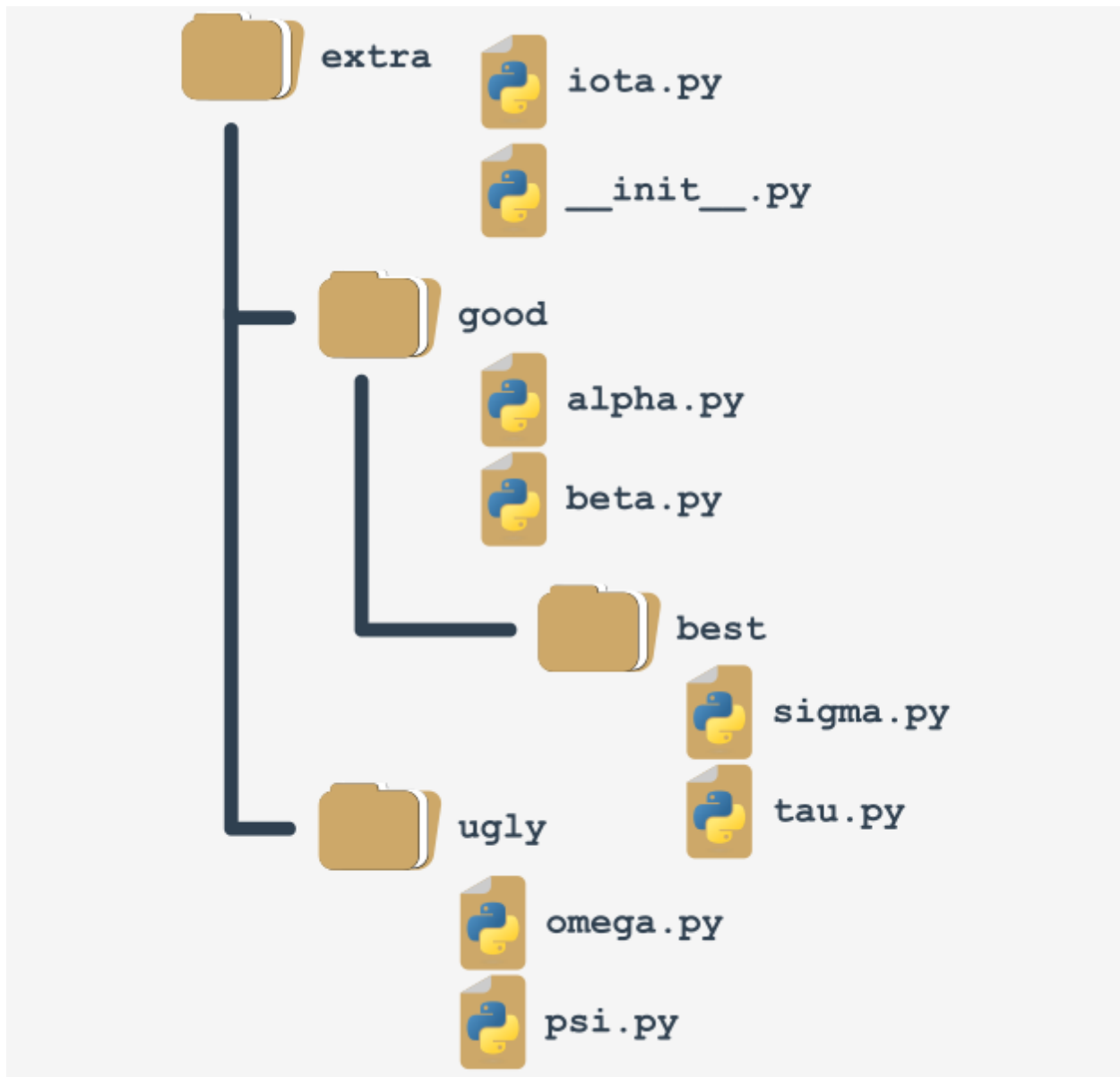
Q haya un archivo con un nombre muy exclusivo dentro de la carpeta de paquete:

init.py

El contenido del archivo se ejecuta cuando se importa cualquiera de los modulos del paquete

Paso 5

La presencia del archivo “__init.py__” compone el paquete



No solo la carpeta raíz puede contener el archivo “__init__.py”, se lo puede poner en cualquier otra subcarpeta

El paquete se puede colocar en cualquier parte para que sea accesible para Python solo hay que asegurarse que Python conozca la ubicación del paquete

Paso 6

Vamos a acceder a la función `funl()` del módulo `iota` del paquete `extra`. Así es como se hace:

```
from sys import path
path.append('../\packages')
import extra.iota
print(extra.iota.funI())
```

Se modifica la variable path para que sea accesible a Python. El import no apunta directamente al modulo, pero especifica la ruta completa desde la parte superior del paquete. El reemplazar “import extra.iota” con “import iota” provocara un error

Paso 7

Asi es como se obtiene acceso a los modulos sigma y tau

```
from sys import path

path.append('../\packages')

import extra.good.best.sigma
from extra.good.best.tau import funT

print(extra.good.best.sigma.funS())
print(funT())
```

Ecosistema de Python

Python es una herramienta interdisciplinaria, lo mas eficiente q todos los miembros de esta comunidad estan constantemente intercambiando sus codigos, asi nadie esta obligado a empezar de ero ya que es muy probable que alguien ya haya tenido que lidiar con el problema que estas lidiando ahora.

Para que esto funcione se necesita un **repositorio centralizado** de todos los paquetes de software disponibles, y una herramienta q permite a los usuarios **acceder al repositorio**.

El repositorio se llama **PyPI** (Python Package Index) y lo matiene un grupo llamado Packaging Working Group. En el se encuentran alrededor de 315mil proyectos. Es gratuito y puedes tomar un codigo y usarlo

La herramienta para hacer uso de PyPI se llama **pip**. Esta herramienta tambien es gratuita, su nombre es un acronimo recursivo (pip instala paquetes)

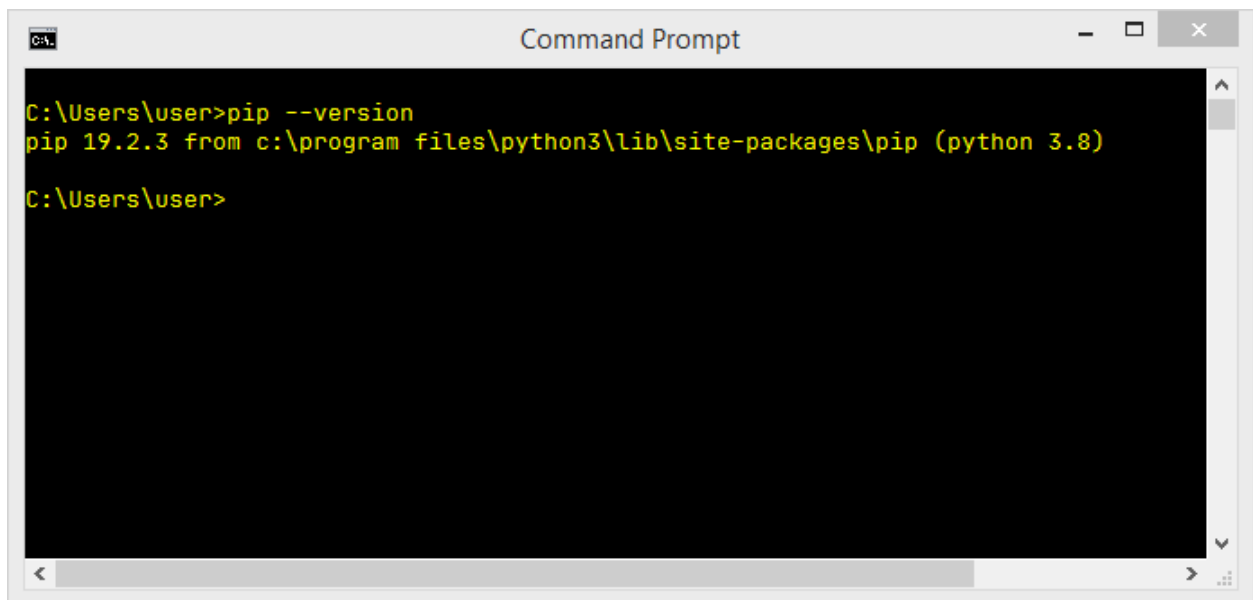
Instalador de Paquetes de Python PIP

Hay algunas instalaciones de Python que vienen con pip, otras que no.

En MS Windows ya viene con Python pero si la variable PATH esta mal configurada es posible que PIP no este disponible. Para comprobar esto lo que se hace es abrir la consola windows y escribir

```
pip --version
```

Si sale algo asi es que esta todo ok



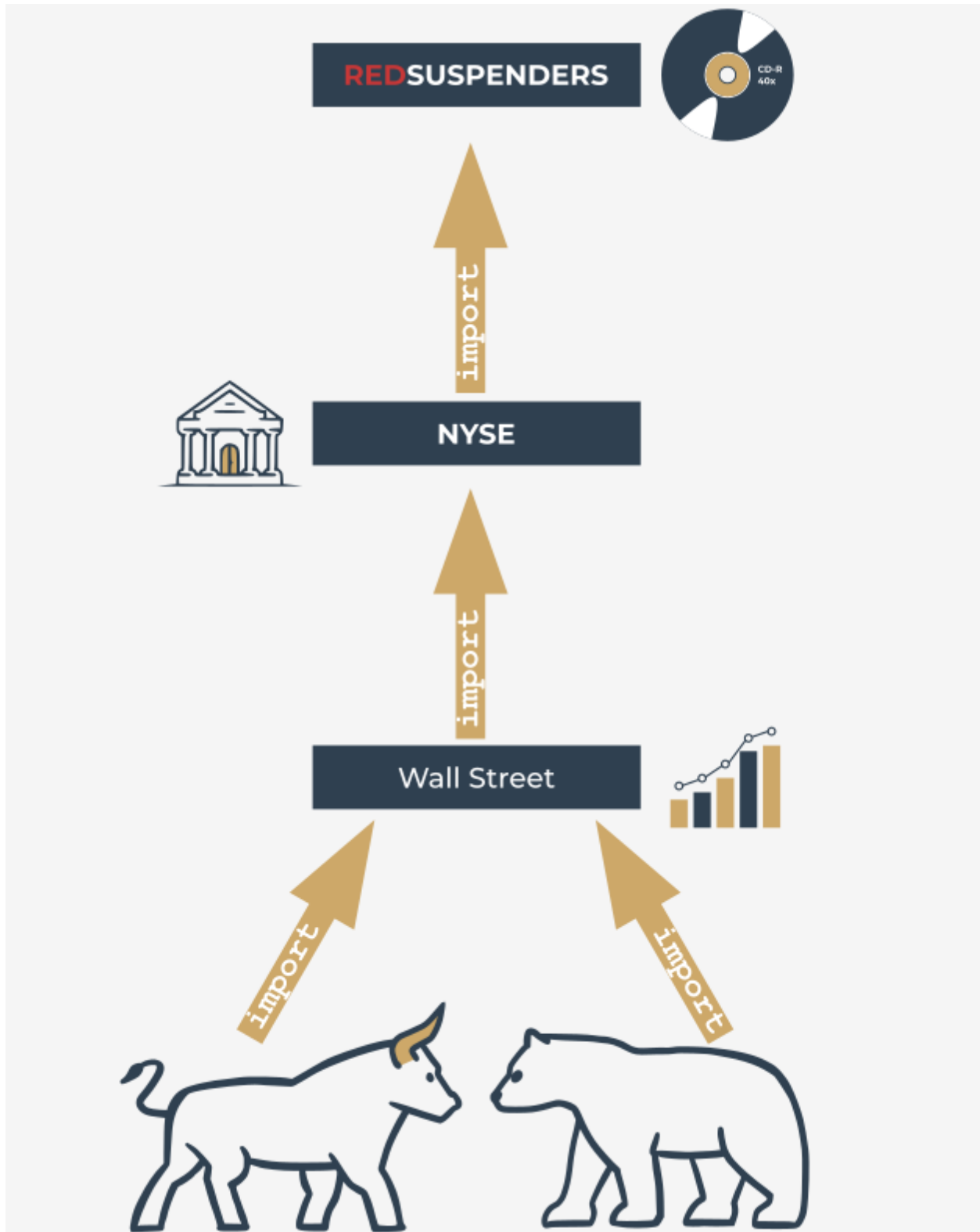
```
Command Prompt

C:\Users\user>pip --version
pip 19.2.3 from c:\program files\python3\lib\site-packages\pip (python 3.8)

C:\Users\user>
```

Dependencias

Supongamos que creamos una aplicacion llamada redsuspenders, y utilizamos algun codigo existente para lograr esto, por ejemplo un paquete llamado nyse que tiene algunas funciones y clases cruciales. A su vz el paquete nyse importa otro paquete llamdo wallstreet y asu vez este importa otros dos llamados bull y bear



La dependencia es un fenomeno que aparece cada vez que vas a utilizar un software que depende de otro software, ten en cuenta que la dependencia puede ,y

generalmente lo hace, incluir mas de un nivel de desarrollo

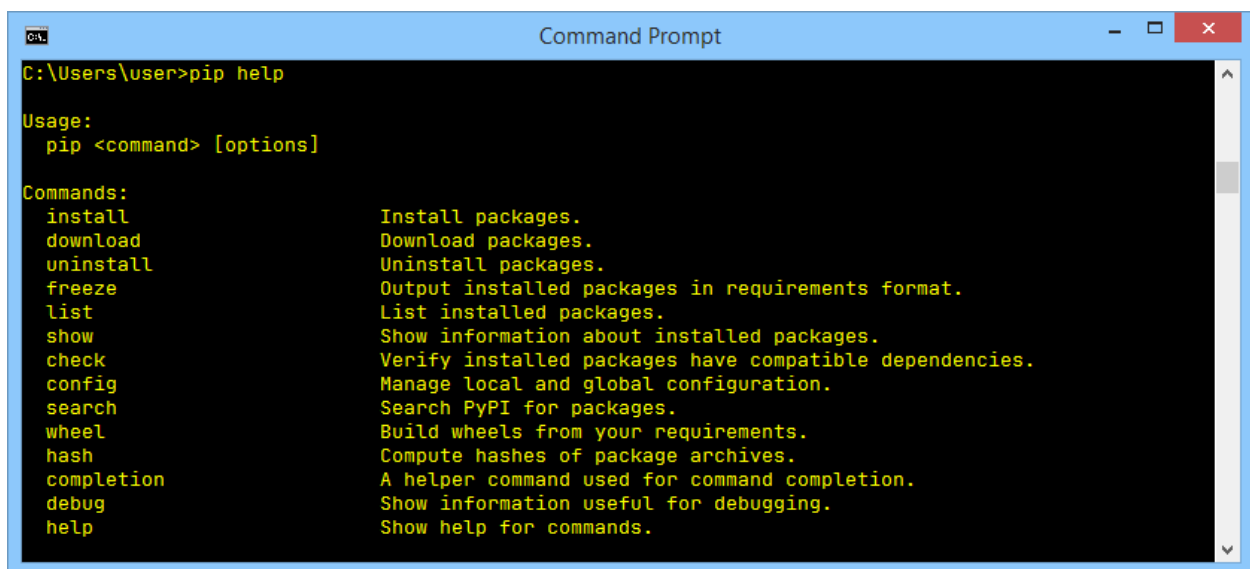
Afortunadamente PIP puede descubrir, identificar y resolver todas las dependencias

Como usar PIP

Le preguntamos a pip que puede hacer por nosotros a travez del comando

```
pip help
```

Y la respuesta sera:



```
Command Prompt

C:\Users\user>pip help

Usage:
  pip <command> [options]

Commands:
  install          Install packages.
  download         Download packages.
  uninstall        Uninstall packages.
  freeze           Output installed packages in requirements format.
  list             List installed packages.
  show             Show information about installed packages.
  check            Verify installed packages have compatible dependencies.
  config           Manage local and global configuration.
  search           Search PyPI for packages.
  wheel            Build wheels from your requirements.
  hash             Compute hashes of package archives.
  completion       A helper command used for command completion.
  debug            Show information useful for debugging.
  help             Show help for commands.
```

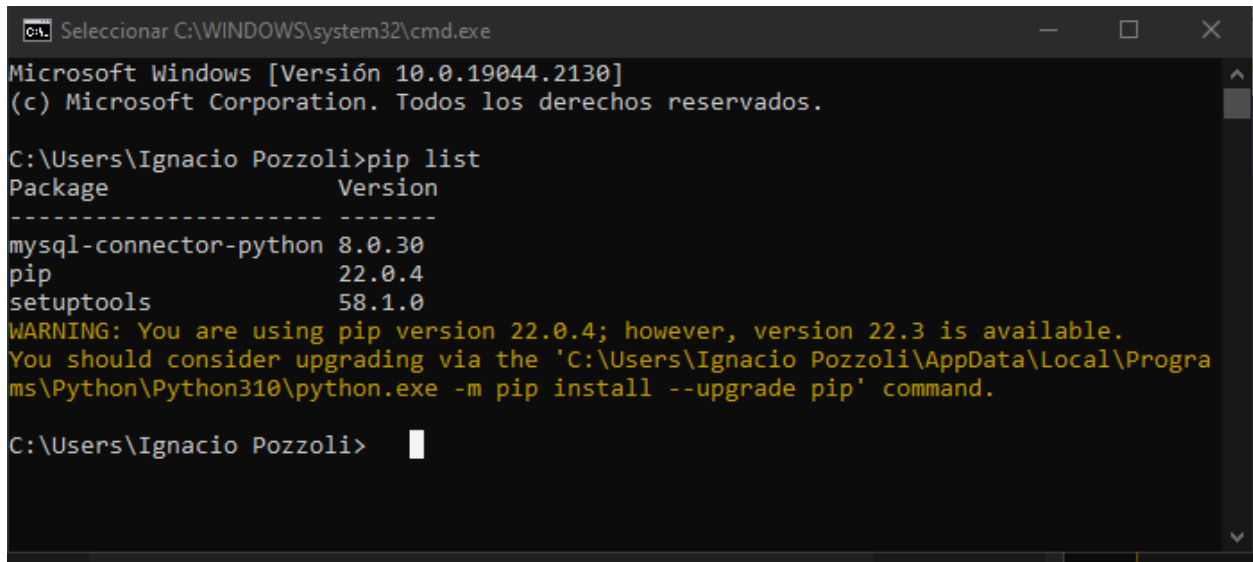
La lista resume todas las operaciones disponibles, para saber mas de una operacion puntual lo que se hace es usar el comando

```
pip help (operacion o comando)
```

- Si por ej se desea saber que paquetes de Python se han instalados se usa la operacion list, asi

```
pip list
```

El resultado dependera de cada compu, en mi caso es asi:



```
C:\Users\Ignacio Pozzoli>pip list
Package            Version
-----
mysql-connector-python 8.0.30
pip                 22.0.4
setuptools          58.1.0
WARNING: You are using pip version 22.0.4; however, version 22.3 is available.
You should consider upgrading via the 'C:\Users\Ignacio Pozzoli\AppData\Local\Programs\Python\Python310\python.exe -m pip install --upgrade pip' command.

C:\Users\Ignacio Pozzoli>
```

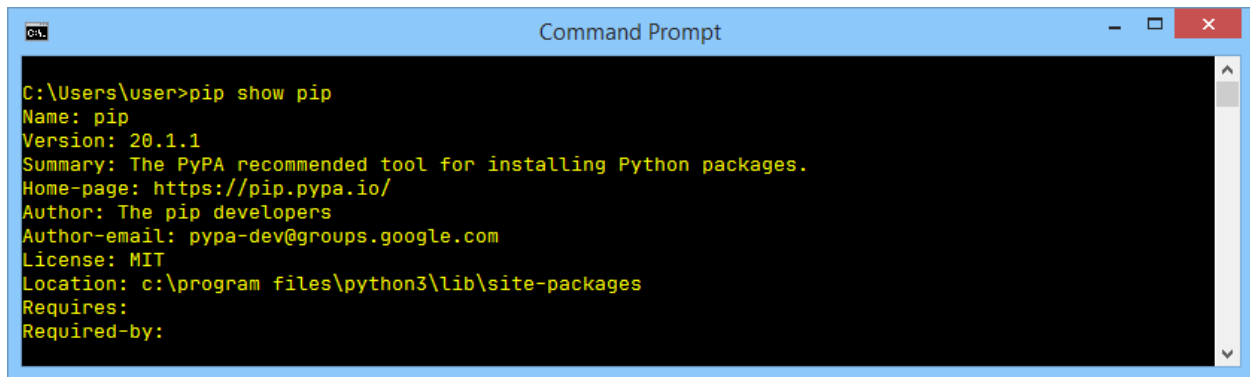
Hay dos columnas en la cual una muestra el nombre del paquete instalado y la otra muestra la version del paquete. Al final nos sale la advertencia de que hay una version de pip para actualizar

- La lista pip no es muy informativa, pero hay un comando q puede brindar mas informacion sobre cualquiera de los paquetes instalados. La sintaxis es:

```
pip show nombre_del_paquete
```

La informacion que aparece cuando ejecutamos este comando es dada por el creador del paquete.

Se mostrara algo como:



```
C:\Users\user>pip show pip
Name: pip
Version: 20.1.1
Summary: The PyPA recommended tool for installing Python packages.
Home-page: https://pip.pypa.io/
Author: The pip developers
Author-email: pypa-dev@groups.google.com
License: MIT
Location: c:\program files\python3\lib\site-packages
Requires:
Required-by:
```

En las dos líneas de la parte inferior se observa:

1. Q paquetes son necesarios para usar con éxito el paquete (Requires:)
2. Que paquetes necesitan que el paquete se utilice con éxito (required-by)

Lo importante de pip es que abre las puertas a todo el universo de software de Python. Pip no almacena todo el contenido de PyPI localmente, usa internet para consultar PyPI y descargar los datos requeridos por lo tanto es necesario una conexión a internet siempre que se solicite a pip cualquier cosa que pueda implicar interacciones directas con la infraestructura de PyPI

Un ejemplo es cuando buscamos en PyPI el paquete deseado, el comando correspondiente es:

```
pip search anystring
```

Instalar paquetes

Suponiendo que la búsqueda del paquete es exitosa y se desea instalar el paquete en la computadora, nos podemos encontrar con dos escenarios:

- Deseas instalar el nuevo paquete solo para tu cuenta: implica que ningún otro usuario de la computadora tendrá acceso a este paquete
- Deseas instalar un nuevo paquete para todo el sistema

Para diferenciar estas dos acciones pip usa una sentencia que es: “—user”. Esta sentencia indica a pip que actúe localmente en nombre de tu usuario sin privilegios de

administrador. Si no se agrega esto pip asume que sos administrador del sistema.

Ejemplo de instalacion de un paquete, pygame

Pygame es una biblioteca que permite desarrollar juegos en python. Se puede usar cualquiera de los dos siguientes comandos:

```
pip install pygame
pip install --user pygame
```

Asi pip te debera mostrar el progreso de la instalacion. Si se instala con exito se ve algo como

```
C:\Users\Ignacio Pozzoli>pip install --user pygame
Collecting pygame
  Downloading pygame-2.1.2-cp310-cp310-win_amd64.whl (8.4 MB)
    ----- 8.4/8.4 MB 693.9 kB/s eta 0:00:00
Installing collected packages: pygame
Successfully installed pygame-2.1.2
WARNING: You are using pip version 22.0.4; however, version 22.3.1 is available.
You should consider upgrading via the 'C:\Users\Ignacio Pozzoli\AppData\Local\Programs\Python\Python310\python.exe -m pip install --upgrade pip' command.
C:\Users\Ignacio Pozzoli>
```

Para verificar usamos el pip list y show

Uso del paquete pygame

Se puede acceder a pygame atravez de un programa como el siguiente:

```
import pygame

run = True
width = 400
height = 100
pygame.init()
screen = pygame.display.set_mode((width, height))
font = pygame.font.SysFont(None, 48)
text = font.render("Bienvenido a pygame", True, (255, 255, 255))
screen.blit(text, ((width - text.get_width()) // 2, (height - text.get_height()) // 2))
pygame.display.flip()
while run:
    for event in pygame.event.get():
        if event.type == pygame.QUIT\
        or event.type == pygame.MOUSEBUTTONUP\
        or event.type == pygame.KEYUP:
            run = False
```

- La línea 1: importa *pygame* y nos permite usarlo.
- La línea 3: el programa se ejecutará mientras la variable `run` sea `True`.
- Las líneas 4 y 5: determinan el tamaño de la ventana.
- La línea 6: inicializa el entorno *pygame*.
- La línea 7: prepara la ventana de la aplicación y establece su tamaño.
- La línea 8: crea un objeto que represente la fuente predeterminada

de 48 puntos.

- La línea 9: crea un objeto que represente un texto dado, el texto será suavizado (`True`) y blanco (`255, 255, 255`).
- La línea 10: inserta el texto en el búfer de pantalla (actualmente invisible).
- La línea 11: invierte los búferes de la pantalla para que el texto sea visible.
- La línea 12: el bucle principal de *pygame* comienza aquí.
- La línea 13: obtiene una lista de todos los eventos pendientes de *pygame*.
- Las líneas 14 a la 16: revisan si el usuario ha cerrado la ventana o ha hecho clic en algún lugar dentro de ella o ha pulsado alguna tecla.
- La línea 15: si es así, se deja de ejecutar el código.

Otras funcionalidades de la sentencia `pip install`

Tiene dos funcionalidades adicionales:

- Actualizar un paquete instalado localmente; si queremos asegurarnos de que estamos usando la última versión de un paquete usamos el siguiente comando:

```
pip install -U nombre_de_paquete
```

-U significa actualizar

- Es capaz de instalar una version seleccionada por el usuario manualmente con el siguiente comando:

```
pip install nombre_del_paquete==version_del_paquete
pip install pygame==1.9.2
```

Eliminar paquetes

Si queremos eliminar un paquete se usa el comando “uninstall”. La sintaxis es:

```
pip uninstall nombre_del_paquete
pip uninstall pygame
```

Lista resumida de algunas actividades principales de pip

- `pip help` *operación_o_comando* muestra una breve descripción de *pip*.
- `pip list` muestra una lista de los paquetes instalados actualmente.
- `pip show` *nombre_del_paquete* muestra información que incluyen las dependencias del paquete.
- `pip search` *cadena* busca en los directorios de PyPI para encontrar paquetes cuyos nombres contengan *cadena*.
- `pip install` *nombre* instala el paquete *nombre* en todo el sistema (espera problemas cuando no tengas privilegios de administrador).
- `pip install --user` *nombre* instala *nombre* solo para ti; ningún otro usuario de la plataforma podrá utilizarlo.
- `pip install -U` *nombre* actualiza un paquete previamente instalado.
- `pip uninstall` *nombre* desinstala un paquete previamente instalado.