

# IPC2 - Unidad 1

## 0.1 Información general

- Información de los medios alternativos y oficiales para la comunicación



- Uso de la plataforma UEDI
- Información del uso de los Foros
- Información del uso de la plataforma Dtt

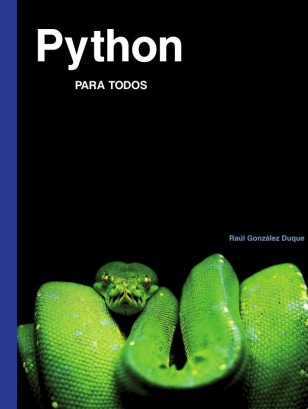
# IPC2 - Información General

- Presentar propuesta del programa de laboratorio
- Informar sobre el libro que se recomienda

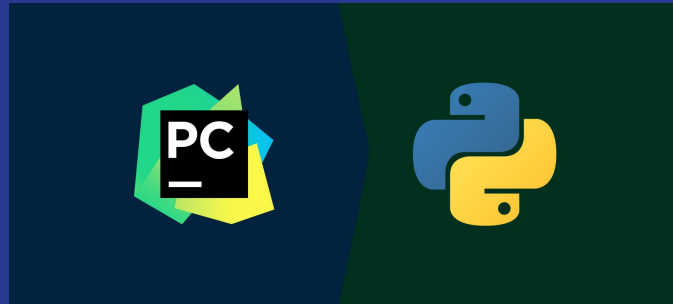
[http://do1.dr-chuck.com/pythonlearn/ES\\_es/pythonlearn.pdf](http://do1.dr-chuck.com/pythonlearn/ES_es/pythonlearn.pdf)

o'

- Información sobre las herramientas que se recomiendan



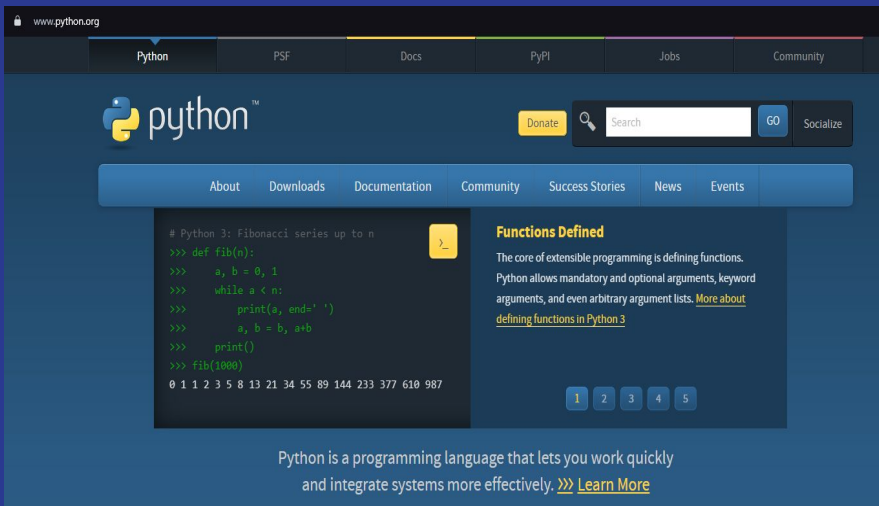
o'



# IPC2 - 1.1 Instalación de Python 3.8.1

## - Instalación en Windows

<https://www.python.org>

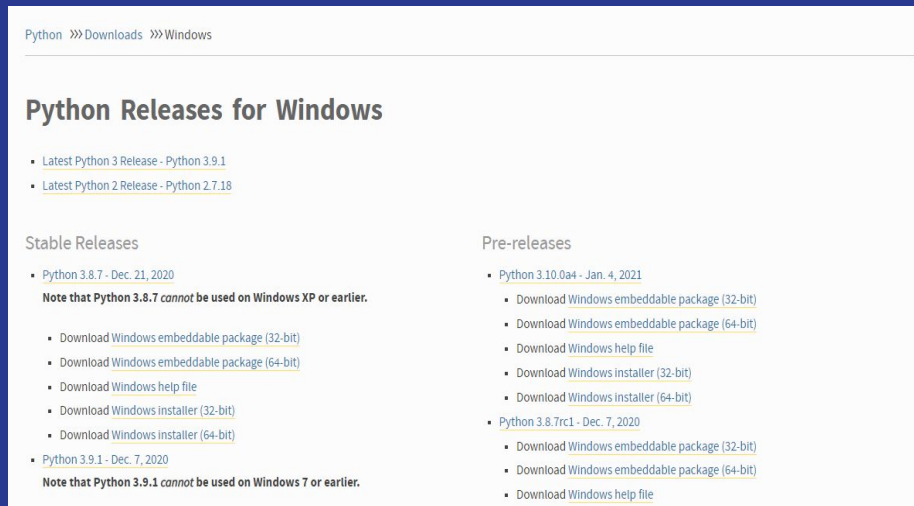


The screenshot shows the Python.org homepage. At the top, there's a navigation bar with links: Python, PSF, Docs, PyPI, Jobs, and Community. Below this is the Python logo and a search bar. A secondary navigation bar contains links: About, Downloads, Documentation, Community, Success Stories, News, and Events. The main content area features a code snippet for a Fibonacci function and a section titled "Functions Defined" explaining the core of extensible programming.

```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

**Functions Defined**  
The core of extensible programming is defining functions. Python allows mandatory and optional arguments, keyword arguments, and even arbitrary argument lists. [More about defining functions in Python 3](#)

Python is a programming language that lets you work quickly and integrate systems more effectively. [»»» Learn More](#)



The screenshot shows the "Python Releases for Windows" page. It lists the latest Python 3 release (3.9.1) and the latest Python 2 release (2.7.18). Below this, there are sections for "Stable Releases" and "Pre-releases". The "Stable Releases" section lists download links for Python 3.8.7 and Python 3.9.1, with notes about their compatibility with Windows XP and Windows 7. The "Pre-releases" section lists download links for Python 3.10.0a4 and Python 3.8.7rc1.

## Python Releases for Windows

- Latest Python 3 Release - Python 3.9.1
- Latest Python 2 Release - Python 2.7.18

### Stable Releases

- Python 3.8.7 - Dec. 21, 2020
  - Note that Python 3.8.7 cannot be used on Windows XP or earlier.**
  - Download Windows embeddable package (32-bit)
  - Download Windows embeddable package (64-bit)
  - Download Windows help file
  - Download Windows installer (32-bit)
  - Download Windows installer (64-bit)
- Python 3.9.1 - Dec. 7, 2020
  - Note that Python 3.9.1 cannot be used on Windows 7 or earlier.**
  - Download Windows embeddable package (32-bit)
  - Download Windows embeddable package (64-bit)
  - Download Windows help file

### Pre-releases

- Python 3.10.0a4 - Jan. 4, 2021
  - Download Windows embeddable package (32-bit)
  - Download Windows embeddable package (64-bit)
  - Download Windows help file
  - Download Windows installer (32-bit)
  - Download Windows installer (64-bit)
- Python 3.8.7rc1 - Dec. 7, 2020
  - Download Windows embeddable package (32-bit)
  - Download Windows embeddable package (64-bit)
  - Download Windows help file

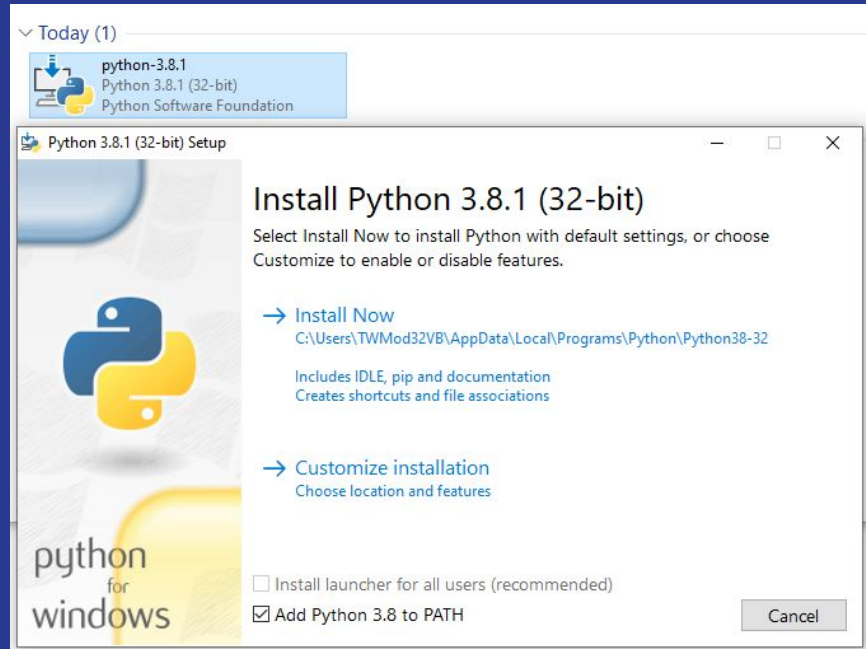
# IPC2 - 1.1 Instalación de Python 3.8.1 en Windows

- En la página de versiones se busca la que se recomienda para el curso

## Python 3.8.1 - Dec. 18, 2019

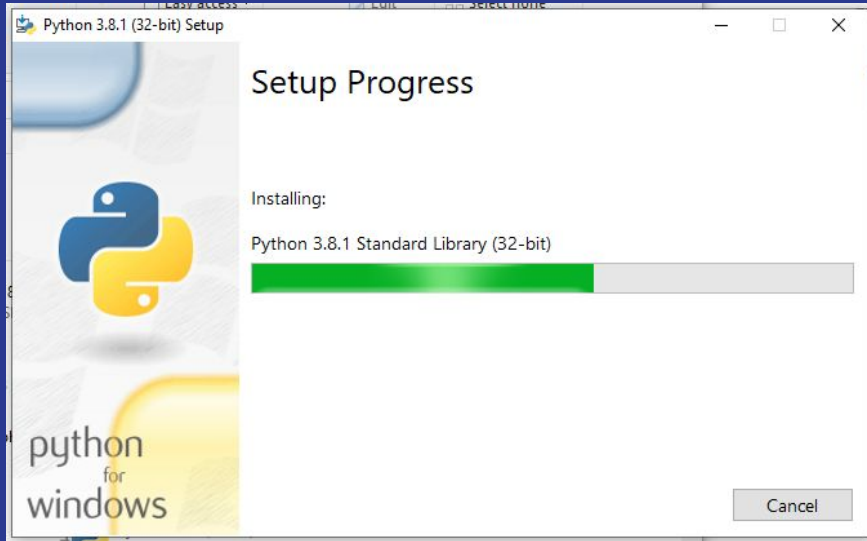
**Note that Python 3.8.1 cannot be used on Windows XP or earlier.**

- Download [Windows help file](#)
- Download [Windows x86-64 embeddable zip file](#)
- Download [Windows x86-64 executable installer](#)
- Download [Windows x86-64 web-based installer](#)
- Download [Windows x86 embeddable zip file](#)
- Download [Windows x86 executable installer](#)
- Download [Windows x86 web-based installer](#)



# IPC2 - 1.1 Instalación de Python 3.8.1 en Windows

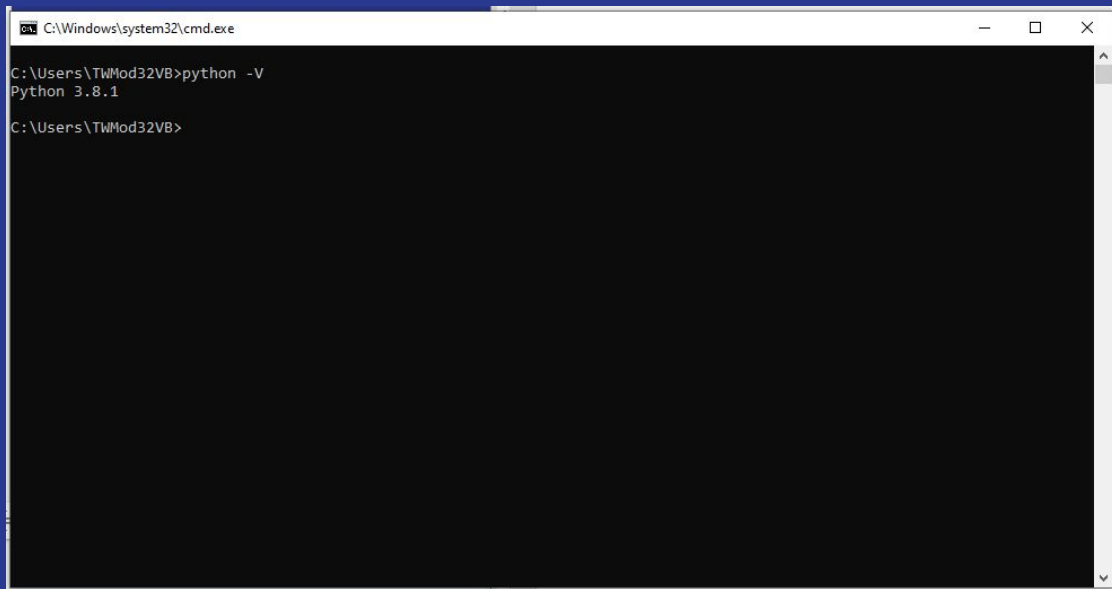
- Seleccionar la opción “Install Now” y el checkbox “Add Python 3.8 to PATH” despliega la siguiente ventana



# IPC2 - 1.1 Instalación de Python 3.8.1 en Windows

- Una vez finalizada la instalación correctamente. Se puede verificar que todo quedo bien configurado con el siguiente comando:

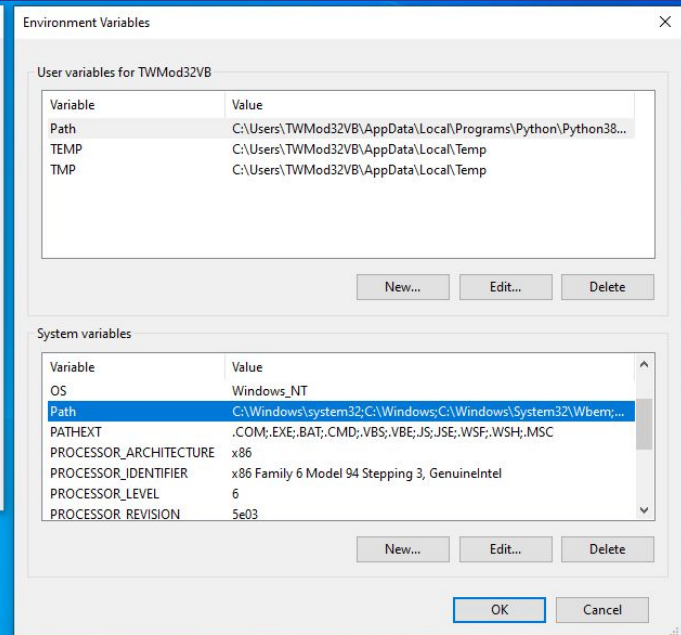
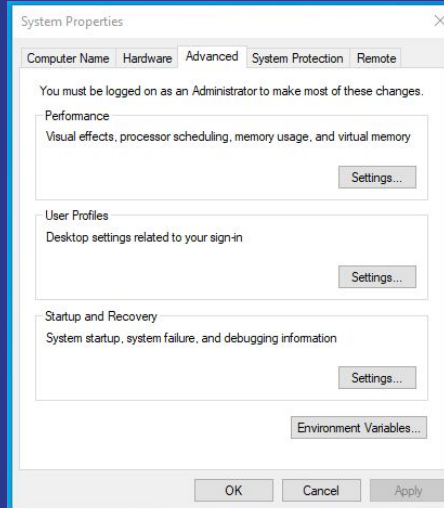
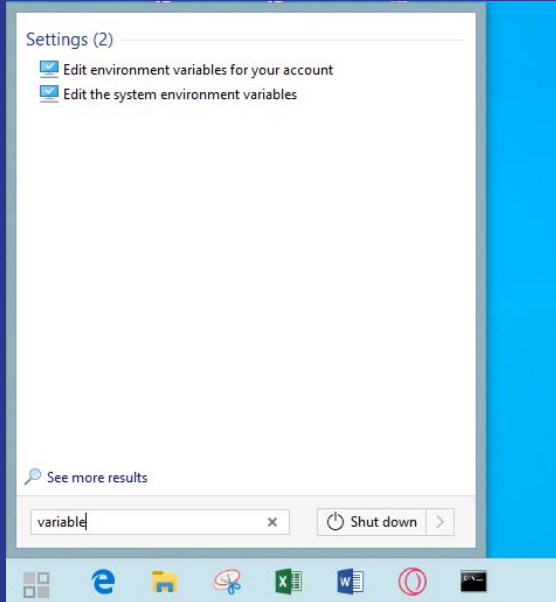
Python -V



```
C:\Windows\system32\cmd.exe
C:\Users\TWMod32VB>python -V
Python 3.8.1
C:\Users\TWMod32VB>
```

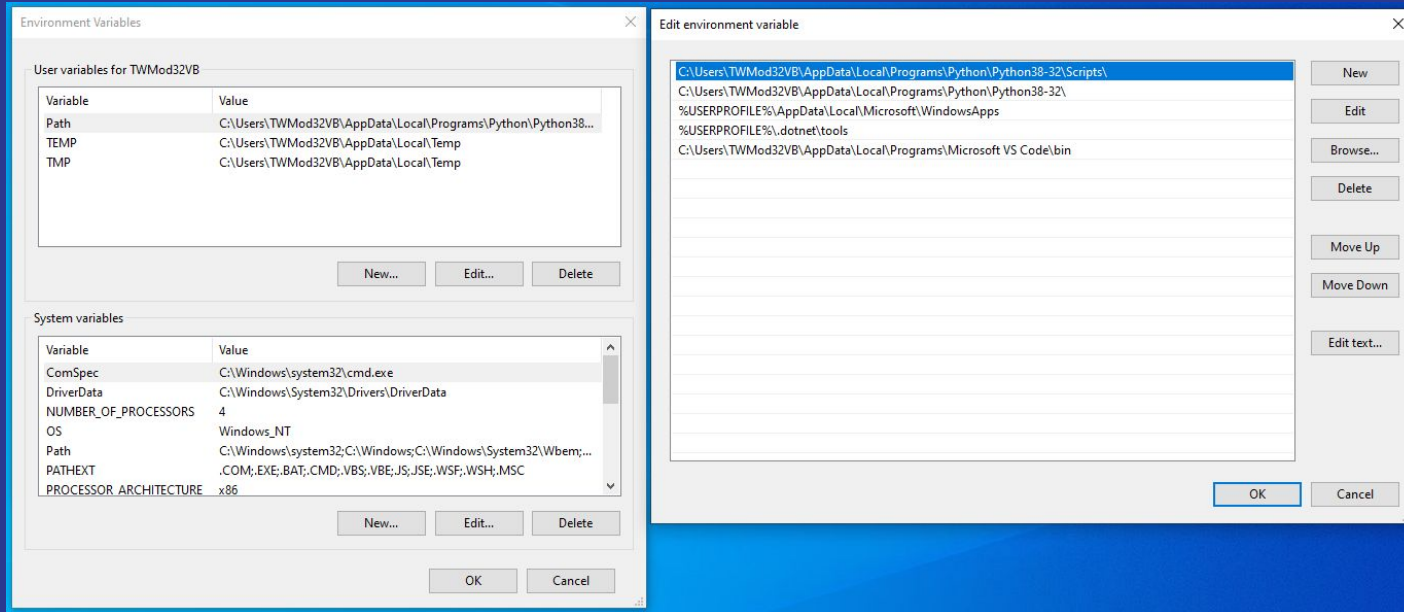
# IPC2 - 1.1 Instalación de Python 3.8.1 en Windows

- También se puede verificar que este agregado Python a la variable de entorno PATH



# IPC2 - 1.1 Instalación de Python 3.8.1 en Windows

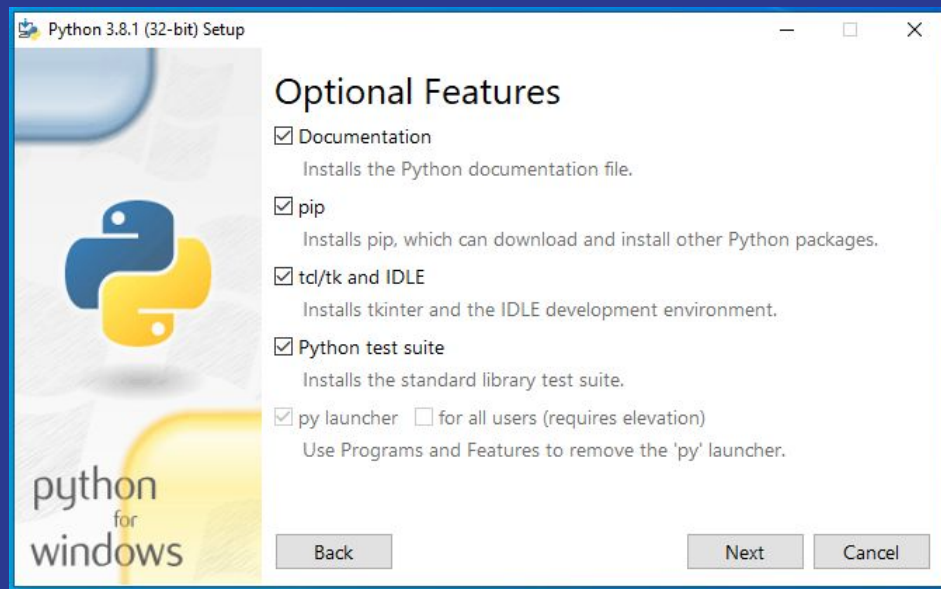
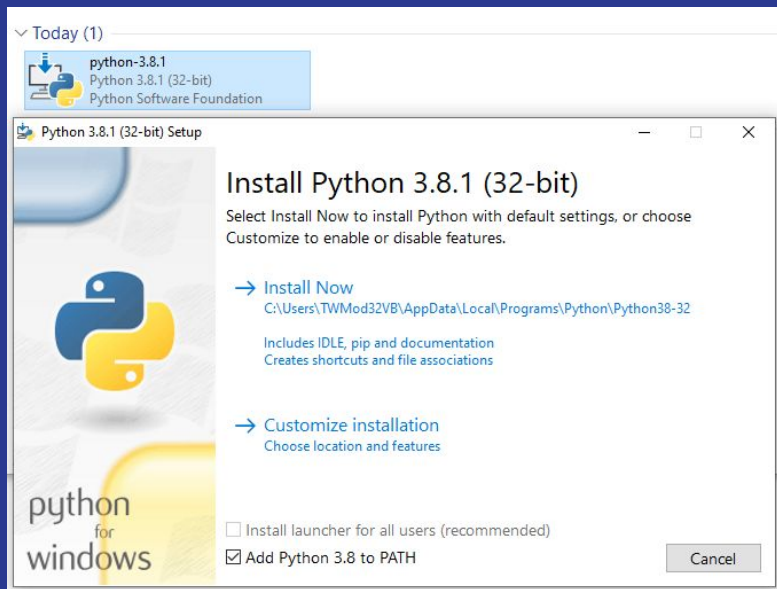
- Esto fue realizado automáticamente por el instalador debido a que fue seleccionada dicha opción, de lo contrario esta parte debe ser realizada por el usuario.





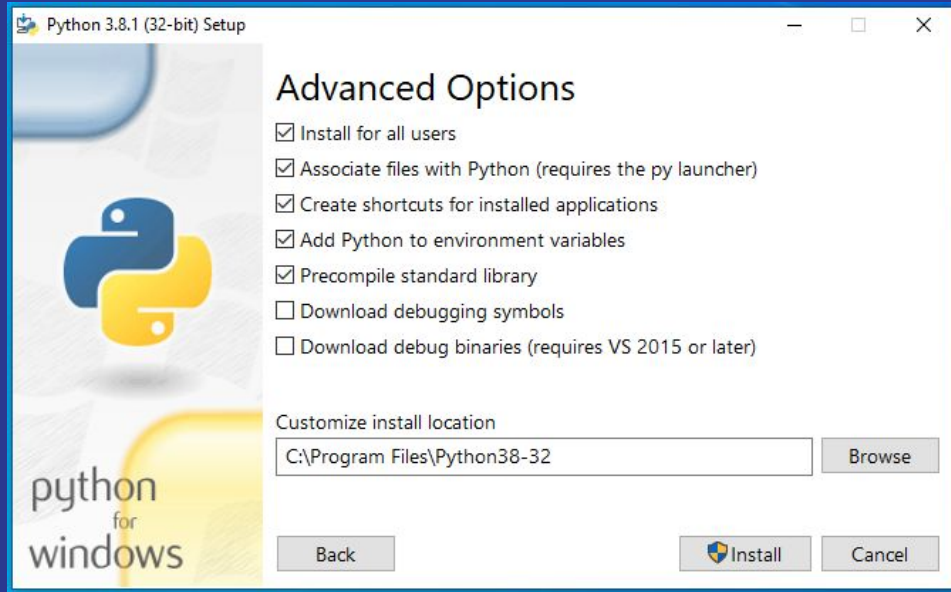
# IPC2 - 1.1 Instalación de Python 3.8.1 en Windows

- También se puede personalizar la instalación de python desde la opción “Customize instalation”



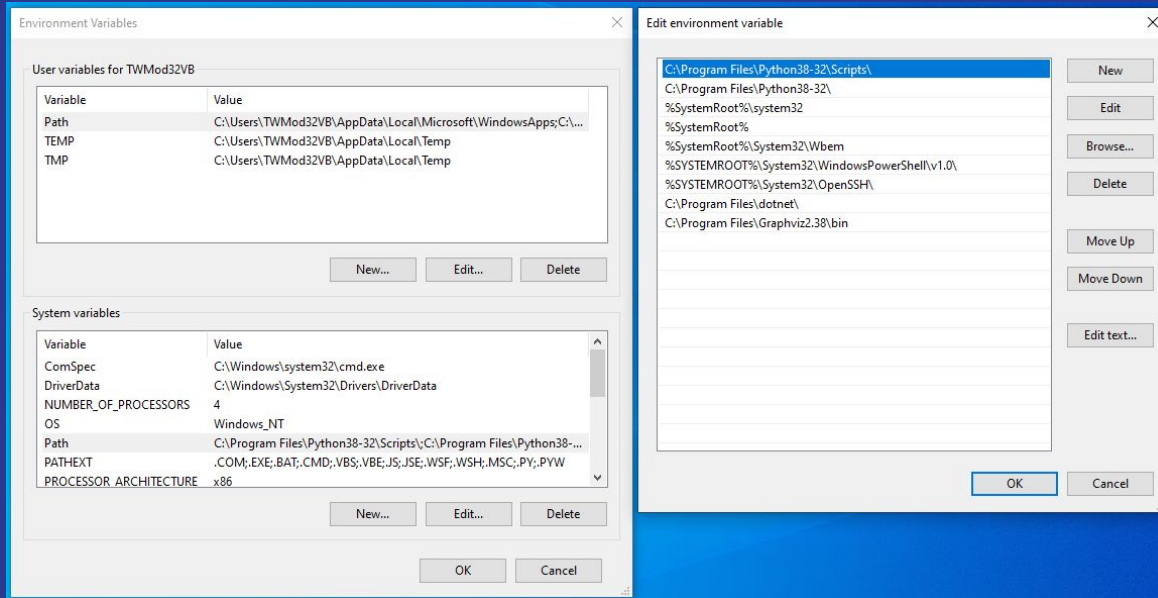
# IPC2 - 1.1 Instalación de Python 3.8.1 en Windows

- En algunos casos se aconseja modificar la ubicación de la instalación



# IPC2 - 1.1 Instalación de Python 3.8.1 en Windows

- Cuando se modifica la ubicación de instalación, los valores que python agrega a la variable de entorno PATH también cambian



# IPC2 - 1.1 Instalación de Python 3.8.1 en Windows

- Los link que provee el instalador de python son los siguientes, es aconsejable revisarlos.

Para acceder a los tutoriales:

<https://docs.python.org/3.8/tutorial/index.html>

Para acceder a la documentación oficial:

<https://docs.python.org/3.8/index.html>

El detalle completo de los cambios de la versión:

<https://docs.python.org/3.8/whatsnew/3.8.html>

# IPC2 - 1.1 Instalación de Python 3.8.1 en Windows

- Porque se usará la versión 3.8.1?

Esta versión, posee instaladores para las arquitecturas de 32 y 64 bit, a diferencia de las versiones más recientes.

Es una versión compatible con los tres sistemas operativos (Windows, Linux, Mac)

Por compatibilidad con las herramientas que se usarán en el proyecto. (Django)

# IPC2 - 1.1 Instalación de Python 3.8.1

- Instalación en Linux (Ubuntu)

<https://linuxize.com/post/how-to-install-python-3-8-on-ubuntu-18-04/>



# IPC2 - 1.1 Instalación de Python 3.8.1 en Ubuntu

- Primero se deben actualizar los repositorios de Ubuntu

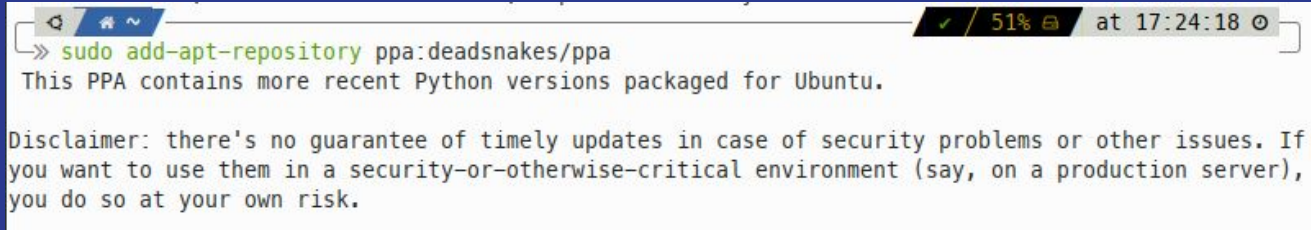
```
» sudo apt update
[sudo] contraseña para tu3:
Obj:1 http://gt.archive.ubuntu.com/ubuntu focal InRelease
Des:3 http://gt.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
```

- Se agrega uno de los paquetes que es prerequisite

```
» sudo apt install software-properties-common
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
software-properties-common ya está en su versión más reciente (0.98.9.3).
El paquete indicado a continuación se instaló de forma automática y ya no es necesario.
  libgsoap-2.8.91
Utilice «sudo apt autoremove» para eliminarlo.
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 20 no actualizados.
```

# IPC2 - 1.1 Instalación de Python 3.8.1 en Ubuntu

- Ahora se debe agregar el ppa y confirmar con la tecla [Enter] cuando lo solicite



```
>> sudo add-apt-repository ppa:deadsnakes/ppa
This PPA contains more recent Python versions packaged for Ubuntu.

Disclaimer: there's no guarantee of timely updates in case of security problems or other issues. If
you want to use them in a security-or-otherwise-critical environment (say, on a production server),
you do so at your own risk.
```

- Se procede a instalar Python



```
>> sudo apt install python3.8.1
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
```

- Finalmente se puede verificar la instalación con:

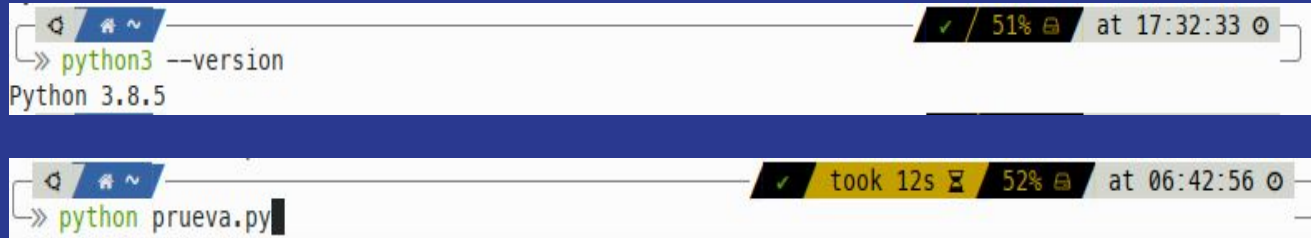


```
>> python3 --version
Python 3.8.5
```



# IPC2 - 1.1 Instalación de Python 3.8.1 en Ubuntu

- Finalmente se puede verificar la instalación con:



The image shows two terminal window screenshots. The top terminal shows the command `python3 --version` being executed, resulting in the output `Python 3.8.5`. The terminal title bar indicates a 51% completion status and a time of 17:32:33. The bottom terminal shows the command `python prueba.py` being executed. The terminal title bar indicates a 52% completion status, a duration of 12s, and a time of 06:42:56.

```
>> python3 --version
Python 3.8.5
```

```
>> python prueba.py
```

En el caso de Ubuntu, la variable de entorno no se debe configurar. Sin embargo como se puede observar en la imagen anterior el sistema ya reconoce la palabra “python” como un comando o instrucción disponible para ejecutar los archivos de extensión “.py”

# IPC2 - 1.2 Instalación de entorno de desarrollo Atom

- Para poder instalar Atom text editor, se puede acceder al link de descarga oficial.

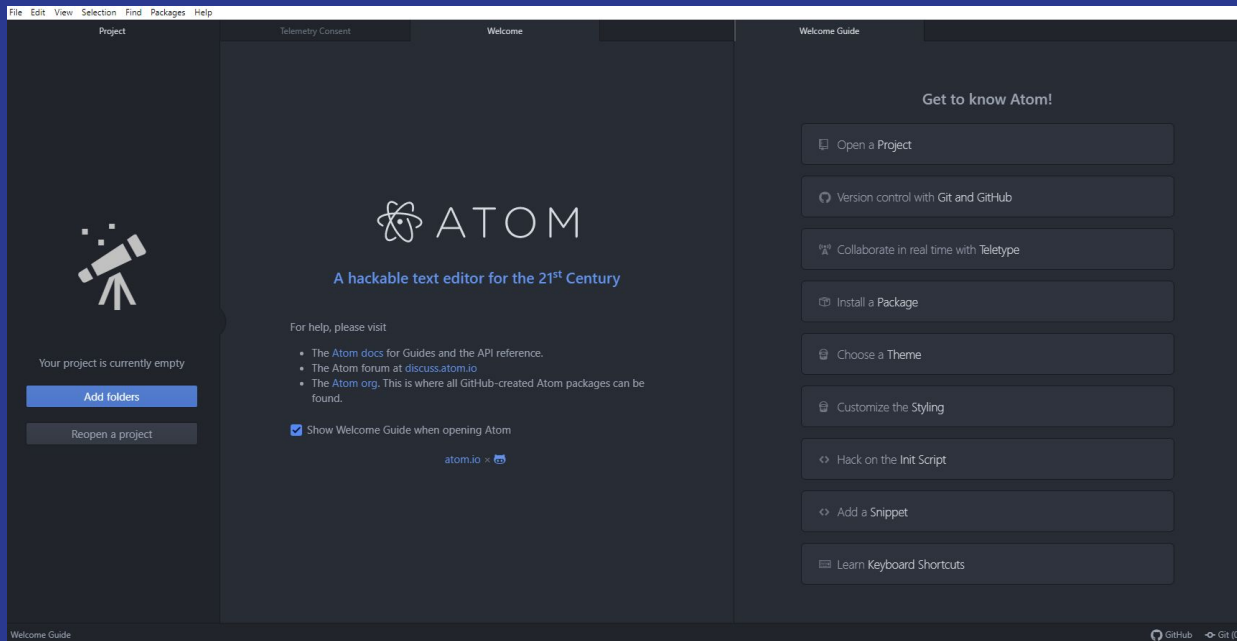
<https://atom.io>

- En Windows para instalar esta herramienta, basta con hacer doble clic sobre el archivo que se descarga. Y el instalador lo hace todo automáticamente, sin configurar nada.



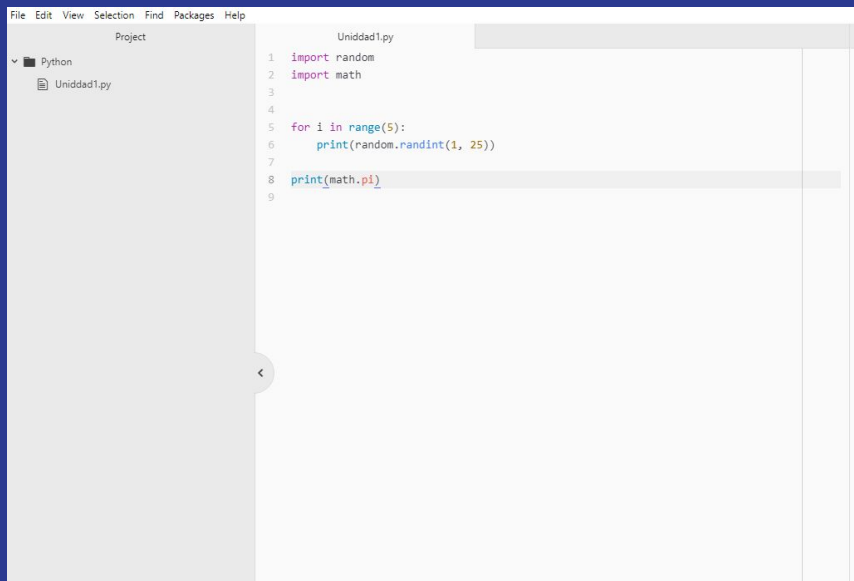
# IPC2 - 1.2 Instalación de entorno de desarrollo Atom

- Al finalizar el asistente de instalación el programa estará listo para usarse.

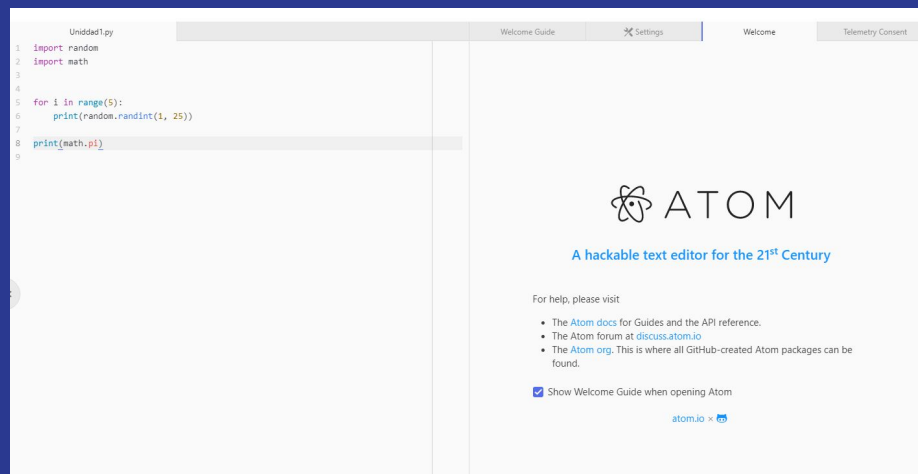


# IPC2 - 1.2 Instalación de entorno de desarrollo Atom

- Atom posee una barra derecha en la cual se visualiza el árbol de ficheros del proyecto que se está editando.

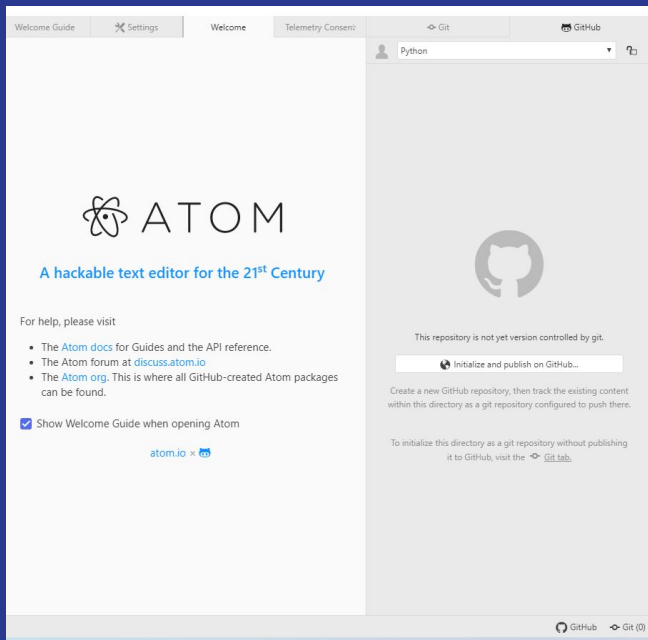


- Atom posee una sección de edición de ficheros que puede contener varias pestañas y se divide en múltiples paneles.

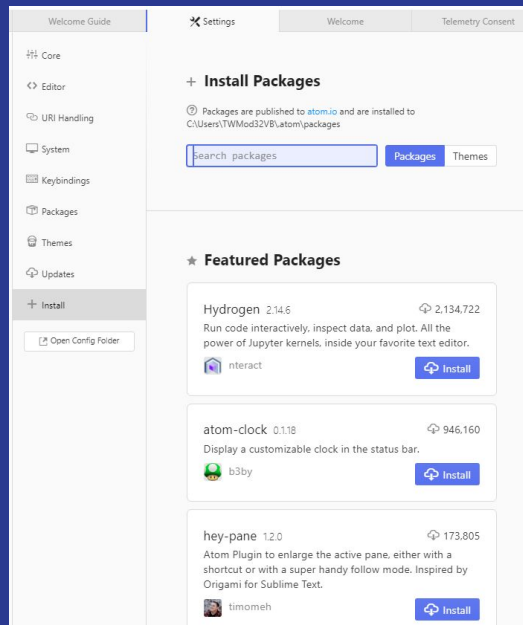


# IPC2 - 1.2 Instalación de entorno de desarrollo Atom

- En la parte izquierda posee una barra desplegable de integración con git para acceder fácilmente a las opciones de control de versiones.

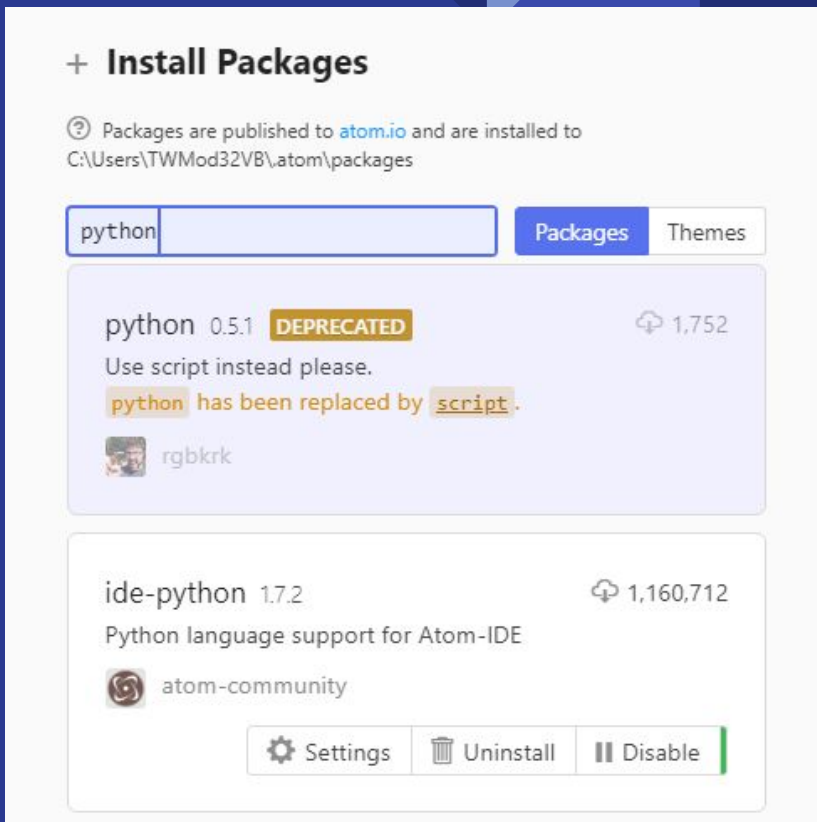


- También posee una gran variedad de paquetes que se pueden agregar, para mejorar el funcionamiento del editor. Dotando a dicha herramienta de funciones específicas.



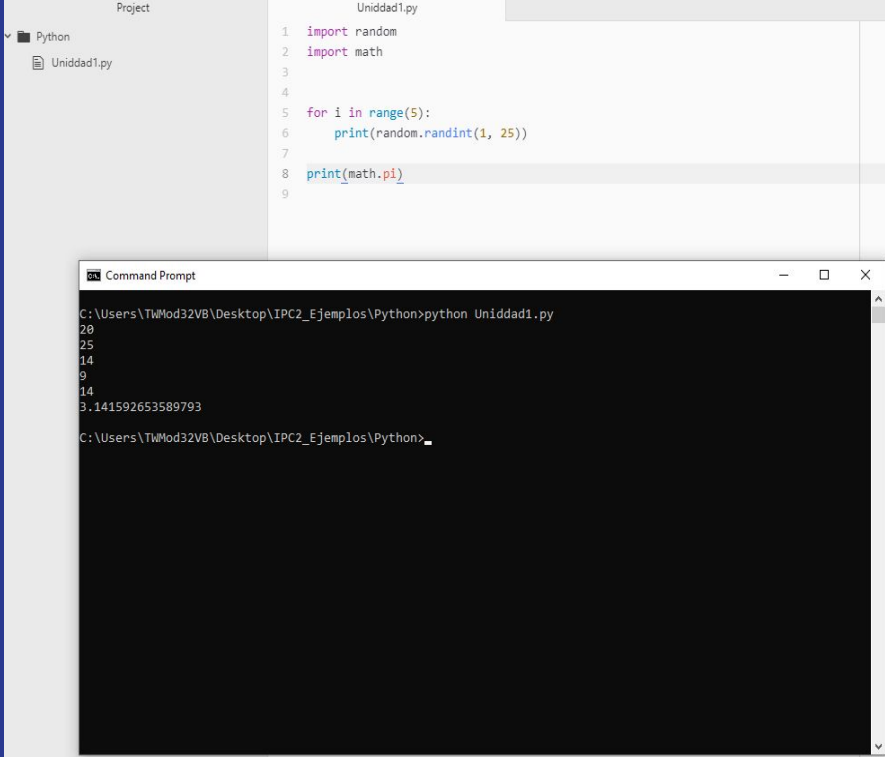
# IPC2 - 1.2 Instalación de entorno de desarrollo Atom

- Existe gran cantidad de paquetes disponibles para atom desde su “tienda” incorporada.
- Para poder ejecutar aplicaciones de python desde el propio editor, se debe agregar el paquete que agrega una terminal incorporada.
- Se recomienda instalar el paquete de ide-python, para qué atom tenga una mejor integración con python



# IPC2 - 1.2 Instalación de entorno de desarrollo Atom

- Para poder ejecutar un programa de python en Windows usando Atom.



The image shows a screenshot of the Atom text editor and a Windows Command Prompt. The Atom editor window has a sidebar on the left showing a project structure with a folder named 'Python' containing a file 'Unidad1.py'. The main editor area displays the contents of 'Unidad1.py', which contains the following Python code:

```
1 import random
2 import math
3
4
5 for i in range(5):
6     print(random.randint(1, 25))
7
8 print(math.pi)
9
```

Below the Atom editor, a Command Prompt window is open. It shows the command `C:\Users\TwMod32VB\Desktop\IPC2_Ejemplos\Python>python Unidad1.py` being executed. The output of the program is displayed in the Command Prompt:

```
20
25
14
9
14
3.141592653589793
C:\Users\TwMod32VB\Desktop\IPC2_Ejemplos\Python>
```

# IPC2 - 1.2 Instalación de entorno de desarrollo Atom

- La alternativa propuesta para Atom es PyCharm de JetBrains. Es un IDE que posee algunas características destacables:

Cuenta con las mismas ventajas de integración con git al igual que Atom.

Con el correo institucional se tiene acceso a la versión educativa de PyCharm la cual posee varios ejemplos y tutoriales de codificación.

La ejecución de las aplicaciones las puede realizar el propio IDE sin necesidad de utilizar la consola por aparte.

Posee herramientas de depuración de código, autocompletado, entre otras.





# IPC2 - PyCharm

## PyCharm

- Para la versión de 64 bit se puede descargar desde la página oficial, sin ningún problema:

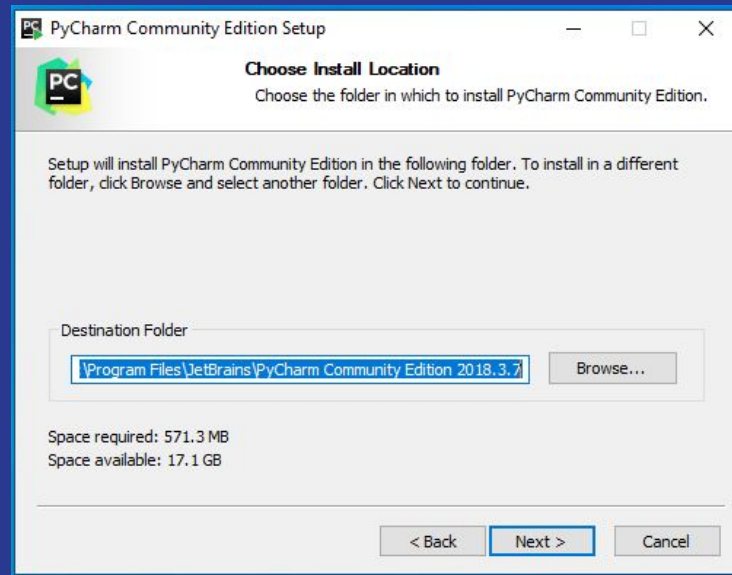
<https://www.jetbrains.com/es-es/pycharm/>

- Para la versión de 32 bits, se debe descargar una versión anterior del programa, en específico PyCharm 2018.3.7 o anteriores, disponibles en el siguiente link:

<https://www.jetbrains.com/pycharm/download/other.html>

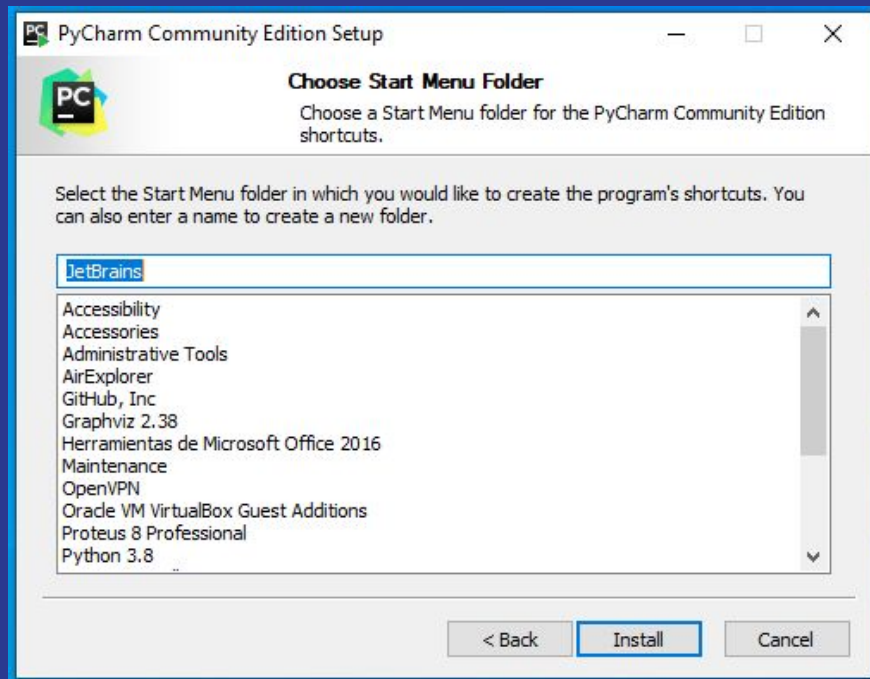
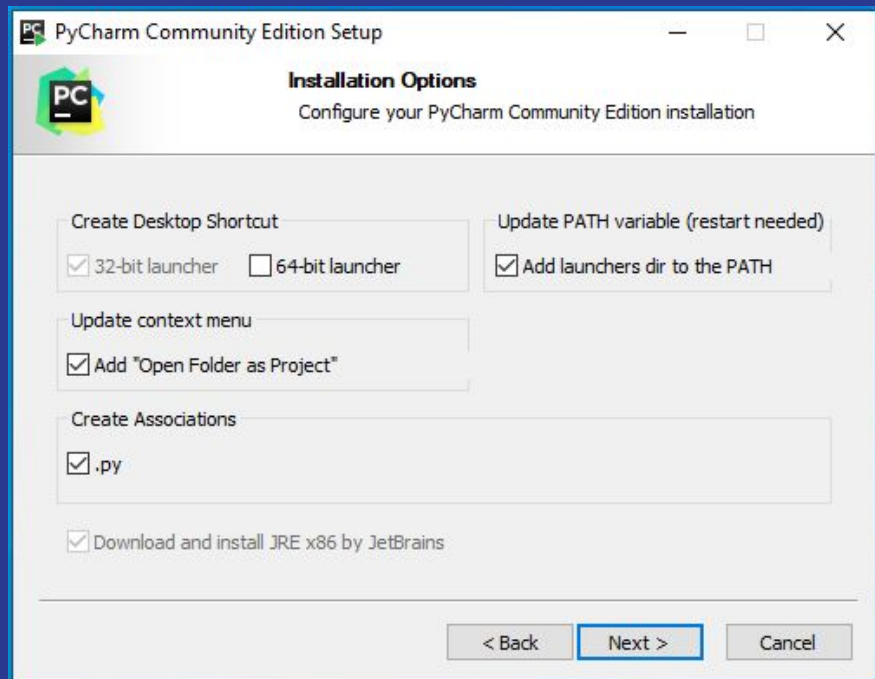
# IPC2 - PyCharm

- Para instalar esta herramienta se le da doble clic al instalador y se siguen los pasos que este indica



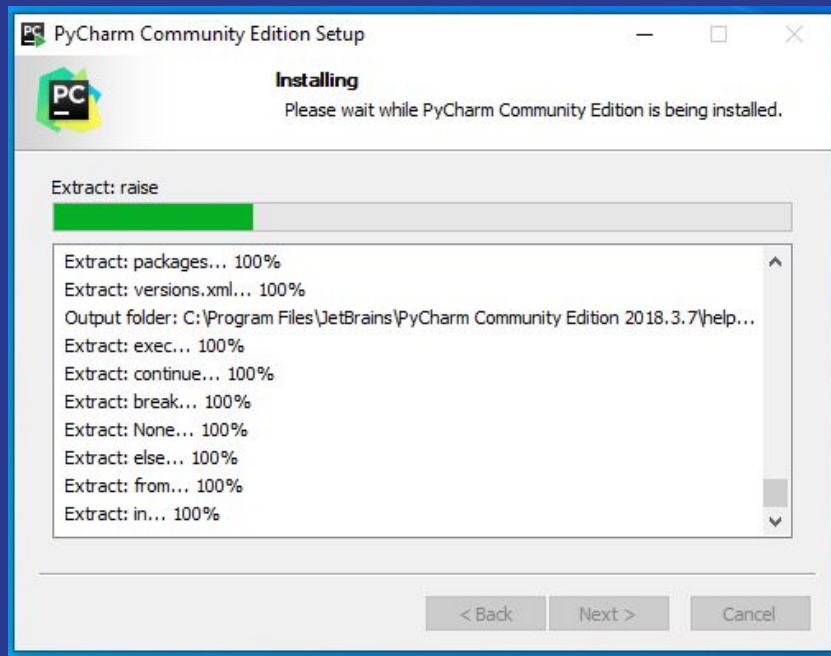
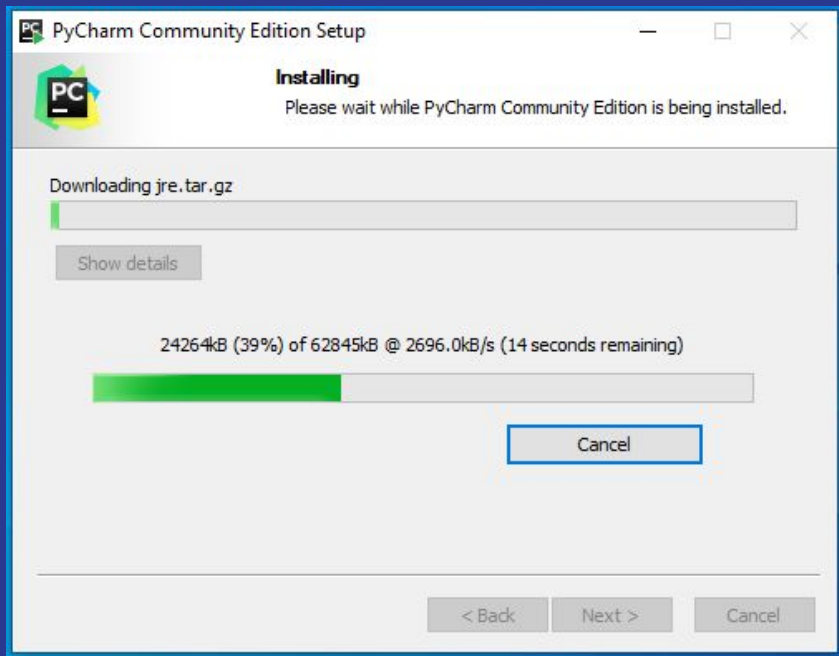
# IPC2 - PyCharm

- Se recomienda seleccionar las siguientes opciones de configuración:



# IPC2 - PyCharm

- El instalador descarga los archivos necesarios y procede con la instalación:



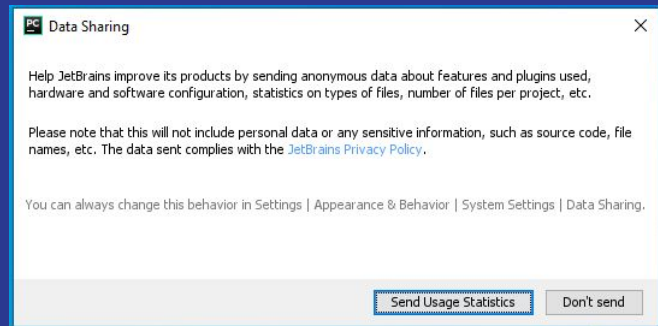
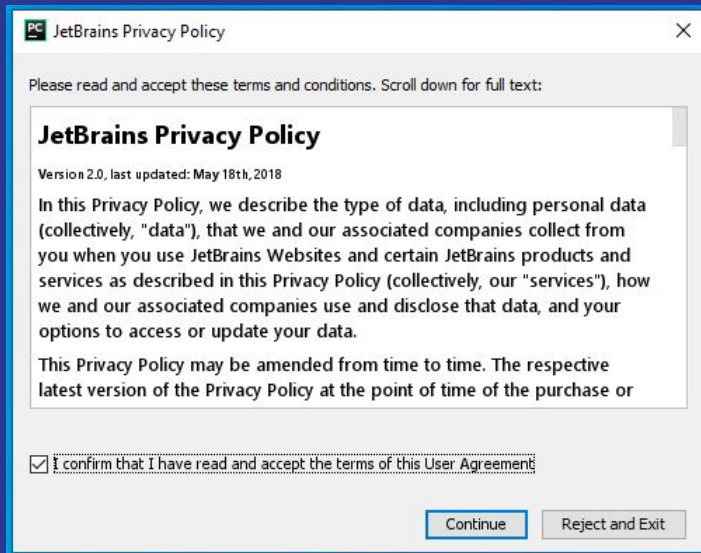
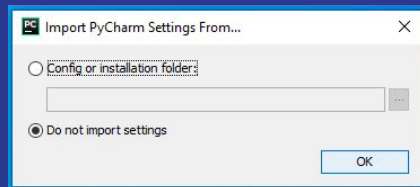
# IPC2 - PyCharm

- Al finalizar la instalación solicita el reinicio del equipo, dependiendo de la configuración que fue elegida en el asistente.



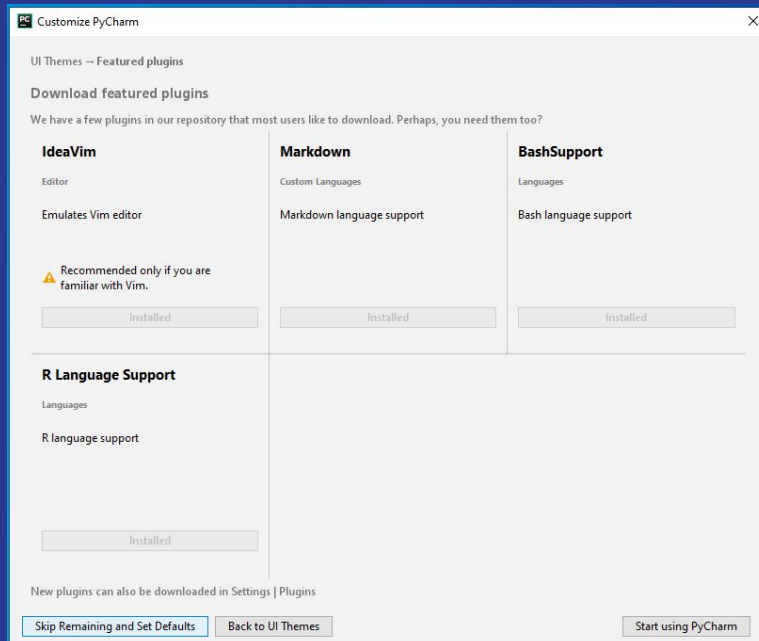
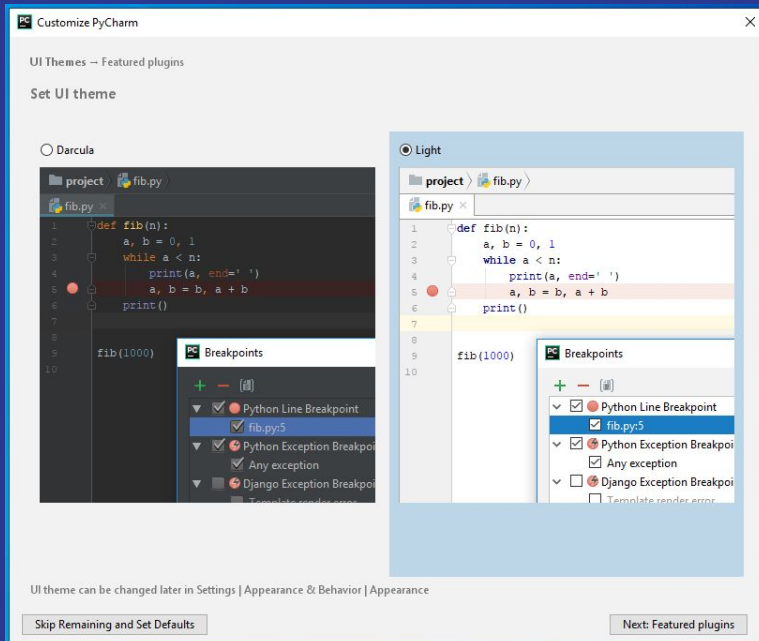
# IPC2 - PyCharm

- En el primer uso solicita importar configuraciones, si no tiene configuración previa no hay problema. También pide aceptar políticas de privacidad. Y solicita permiso para enviar estadísticas, para el análisis de JetBrains.



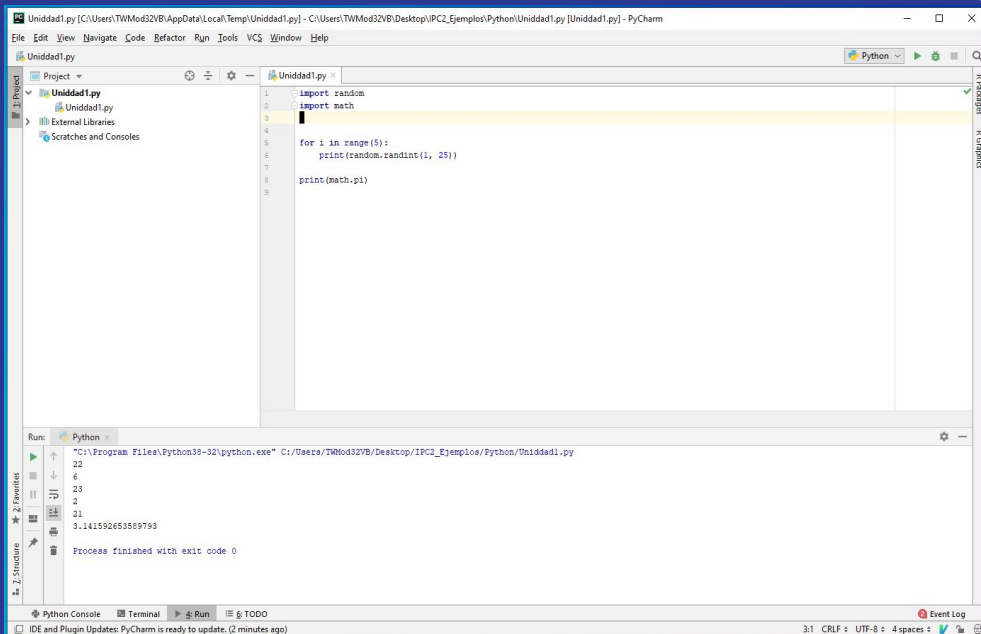
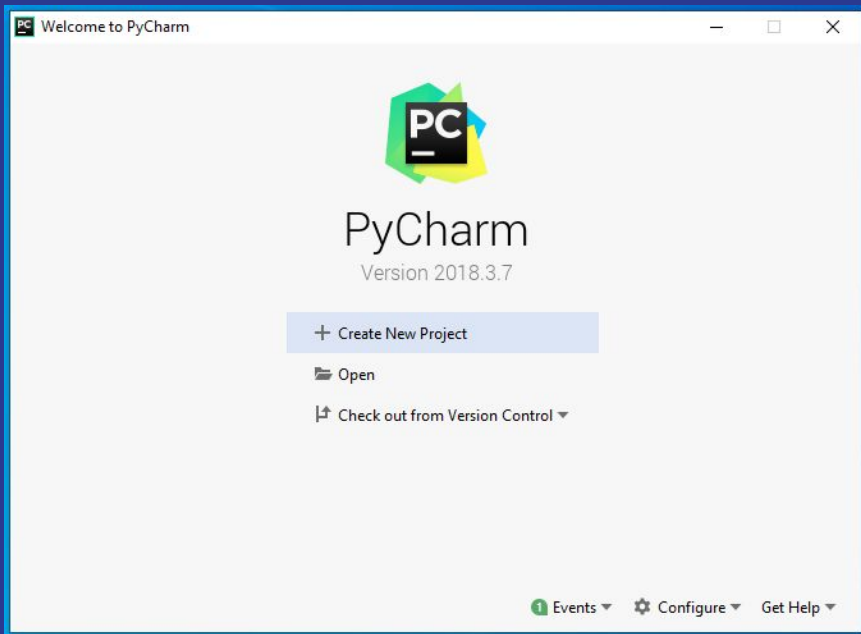
# IPC2 - PyCharm

- Inicialmente solicita seleccionar un aspecto para el IDE, y también pregunta para instalar algunos plugins recomendados.



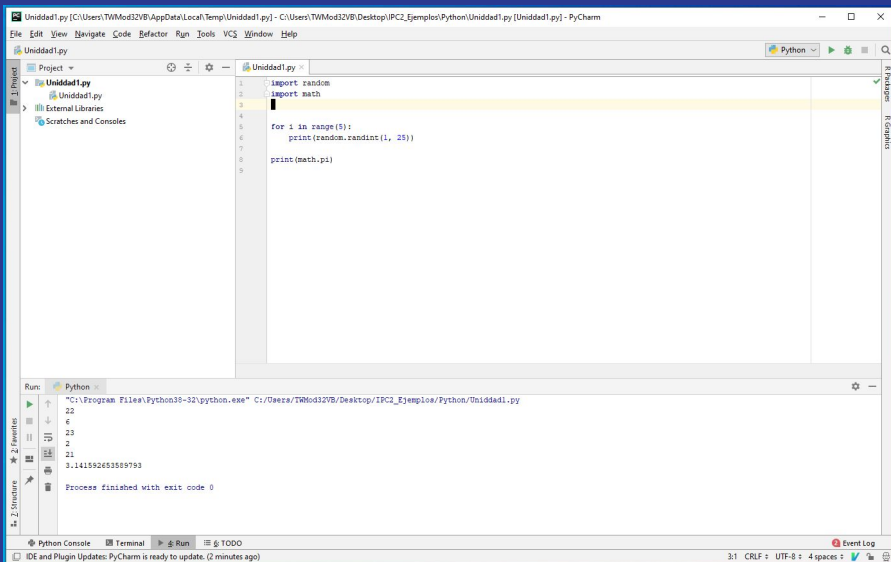
# IPC2 - PyCharm

- La pantalla inicial permite crear, abrir proyecto o clonar un proyecto a partir de una herramienta de control de versiones.





# IPC2 - PyCharm



## PyCharm

Como se puede observar en la imagen, este IDE posee de forma integrada de manera predefinida, varias secciones que permiten:

- El autocompletado
- Debugger
- Ejecución integrada
- Sugerencias de ayuda.

Todo esto incorporado en una sola ventana que provee dichas opciones de manera intuitiva.

## IPC2 - 1.3 Ejercicios para probar entorno de desarrollo y programación básica

- Qué ventajas se conocen de utilizar Python?
- Que tipo de lenguaje es Python? Interpretado
- Cómo funciona Python? utiliza módulos de código que son intercambiables en lugar de una larga lista de instrucciones que era estándar para los lenguajes de programación funcional.

# IPC2 - 1.3 Ejercicios para probar entorno de desarrollo y programación básica

## Diferencias entre la sintaxis de las distintas versiones de python

```
nombre = raw_input('Introduzca fichero:')
manejador = open(nombre, 'r')
texto = manejador.read()
palabras = texto.split()
contadores = dict()

for palabra in palabras:
    contadores[palabra] = contadores.get(palabra,0) + 1

mayorcantidad = None
mayorpalabra = None
for palabra,contador in contadores.items():
    if mayorcantidad is None or contador > mayorcantidad:
        mayorpalabra = palabra
        mayorcantidad = contador

print mayorpalabra, mayorcantidad
```

```
name = input('Enter file:')
handle = open(name, 'r')
counts = dict()

for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

bigcount = None
bigword = None
for word, count in list(counts.items()):
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

## IPC2 - 1.3 Ejercicios para probar entorno de desarrollo y programación básica

El ejemplo anterior se encuentra en el libro, la sintaxis de la parte izquierda es de una versión antigua de Python y la sintaxis de la derecha de las versiones más recientes.

El programa obtiene la palabra que más se repita en un archivo indicado por el usuario y cuenta la cantidad de veces que dicha palabra se repite, para mostrar el resultado finalmente en pantalla.

Es importante resaltar la facilidad con que Python accede a los archivos y la razón por la cual esto es posible. El acceso es posible gracias al handle que es un objeto de tipo `_io.TextIOWrapper`

# IPC2 - 1.3.1. Variables, expresiones y sentencias

Variables:

Un valor es una de las cosas básicas que utiliza un programa, como una letra o un número. Esos valores pertenecen a tipos diferentes. Una variable puede visualizarse como un espacio dentro del programa que adquiere un valor que puede cambiar (variar) durante la ejecución del programa.

Existen los tipos de variable en el lenguaje Python?

Si pero, no es necesario definir el tipo de variable cuando se escribe el código para la realizar la declaración de dicha variable. Python detecta el tipo de variable según el valor que se le asigna a la misma.

# IPC2 - 1.3.1. Variables, expresiones y sentencias

En el ejemplo se demuestra como python detecta el tipo de una variable en relación con su asignación, aunque no sea necesaria la instrucción de asignar el tipo de la variable en cada declaración. Esto es así, de acuerdo a su sintaxis.

También se puede observar que el tipo de valor asignado a una variable puede cambiar durante la ejecución sin necesidad de cambiar el tipo de variable o cambiar la variable por otra.

```
# Tipos de variables
a = 1                #Variable de tipo int
b = 'dos'           #Variable de tipo string
c = 3.14            #Variable de tipo float
d = True            #Variable de tipo booleano
e = False           #Variable de tipo booleano
f = [a,b,c,d,e,]    #Variable de tipo lista
#Imprimir los tipos de variable y su valor
print(type(a), a)
print(type(b), b)
print(type(c), c)
print(type(d), d)
print(type(e), e)
#Imprimir la lista de valores en f y el tipo de
#variable
print([x, type(x) for x in f])
#Una variable tambien puede cambiar de tipo
#durante la ejecucion
print(type(f))
g = 1
print(type(g))
g = 'nuevo valor'
print(type(g))
```

## IPC2 - 1.3.1. Variables, expresiones y sentencias

En el ejemplo anterior se menciona el cambio de tipo de variable durante la ejecución, solamente realizando una reasignación de valores.

```
print(type(f))  
g = 1  
print(type(g))  
g = 'nuevo valor'  
print(type(g))
```

Sin embargo esto no substituye el casteo lo cual se puede realizar de la siguiente forma

int (x)	Convierte x en un entero
long (x)	Convierte x en un entero largo
float (x)	Convierte x en un número de punto flotante
str (x)	Convierte x a una cadena. x puede ser del tipo float, entero o largo
hex (x)	Convierte x entero en una cadena hexadecimal
chr (x)	Convierte x entero a un caracter
ord (x)	Convierte el carácter x en un entero

## IPC2 - 1.3.1. Variables, expresiones y sentencias

Una de las características más potentes de un lenguaje de programación es la capacidad de manipular variables. Una variable es un nombre que se refiere a un valor.

Una sentencia de asignación crea variables nuevas y les da valores:

```
a = 1           #Variable de tipo int
b = 'dos'       #Variable de tipo string
c = 3.14        #Variable de tipo float
d = True        #Variable de tipo booleano
e = False       #Variable de tipo booleano
f = [a,b,c,d,e,] #Variable de tipo lista
```

El tipo de una variable es el tipo del valor al que se refiere.



## IPC2 - 1.3.1. Variables, expresiones y sentencias

El nombre con el que se puede crear una variable en python está restringido por una serie de reglas: debe empezar con una letra y puede contener números. Cualquier otra combinación de caracteres no está permitida. Si estas reglas no se cumplen, python devuelve un error sintáctico.

El libro presenta los siguientes ejemplos de error al nombrar una variable:

```
>>> 76trombones = 'gran desfile'
SyntaxError: invalid syntax
>>> more@ = 1000000
SyntaxError: invalid syntax
>>> class = 'Teorema avanzado de Zymurgy'
SyntaxError: invalid syntax
```

## IPC2 - 1.3.1. Variables, expresiones y sentencias

Se aclara que las palabras clave no pueden ser utilizadas como variables. En el ejemplo anterior se muestra que no se aceptan números al inicio del nombre para una variable, tampoco se aceptan caracteres diferentes de letras y números, y el error de utilizar las palabras clave como un nombre de variable.

Python reserva 33 palabras claves para su propio uso:

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

# IPC2 - 1.3.1. Variables, expresiones y sentencias

Una sentencia es una unidad de código que el intérprete de Python puede ejecutar.

Un script normalmente contiene una secuencia de sentencias. Si hay más de una sentencia, los resultados aparecen de uno en uno según se van ejecutando las sentencias.

Los operadores son símbolos especiales que representan cálculos, como la suma o la multiplicación. Los valores a los cuales se aplican esos operadores reciben el nombre de operandos.

Ha habido un cambio en el operador de división entre Python 2.x y Python 3.x. En Python 3.x, el resultado de esta división es un resultado de punto flotante:

```
>>> minute = 59
>>> minute/60
0.9833333333333333
```

El operador de división en Python 2.0 dividiría dos enteros y trunca el resultado a un entero:

```
>>> minute = 59
>>> minute/60
0
```

Para obtener la misma respuesta en Python 3.0 use división dividida (`//` integer).

```
>>> minute = 59
>>> minute//60
0
```

## IPC2 - 1.3.1. Variables, expresiones y sentencias

Una expresión es una combinación de valores, variables y operadores. Un valor por sí mismo se considera una expresión, y también lo es una variable, así que las siguientes expresiones son todas válidas (asumiendo que la variable *x* tenga un valor asignado):

```
17
```

```
x
```

```
x + 17
```

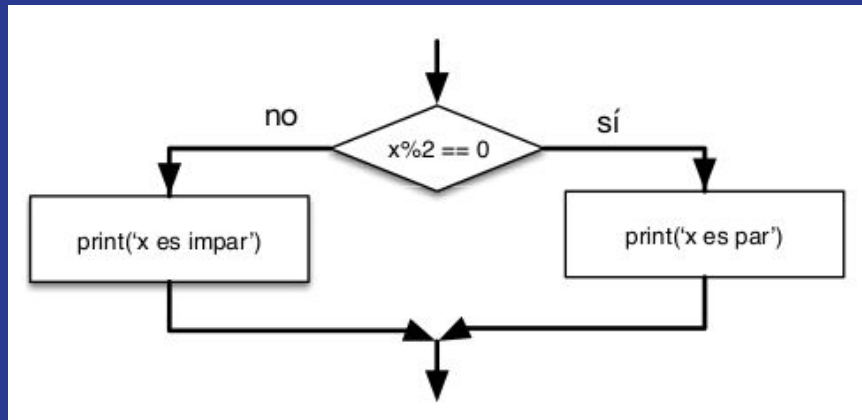
Sin embargo, en un script, ¡una expresión por sí misma no hace nada! Esto a menudo puede producir confusión.

# IPC2 - 1.3.2. Ejecución condicional

Para poder escribir programas útiles, casi siempre se necesita la capacidad de comprobar condiciones y cambiar el comportamiento del programa de acuerdo a ellas. Las sentencias condicionales nos proporcionan esa capacidad.

Una ejecución condicional es un segmento de código que se ejecuta en base al resultado de una expresión booleana.

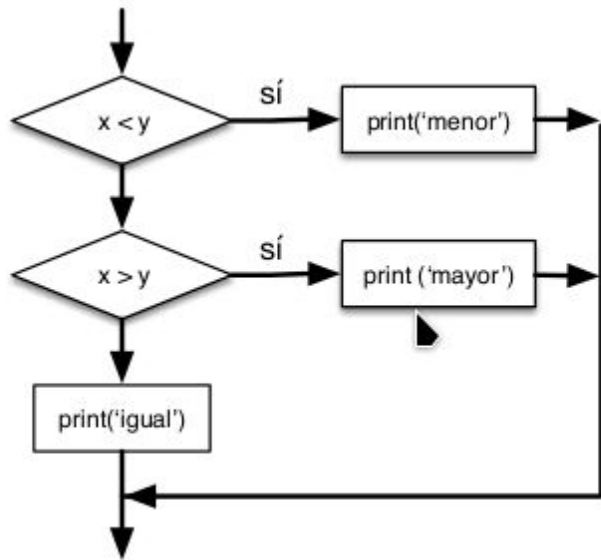
```
if x%2 == 0 :  
    print('x es par')  
else :  
    print('x es impar')
```



# IPC2 - 1.3.2. Ejecución condicional

Algunas veces hay más de dos posibilidades, de modo que es necesario más de dos ramas. Una forma de expresar una operación como ésta es usar un condicional encadenada:

```
if x < y:  
    print('x es menor que y')  
elif x > y:  
    print('x es mayor que y')  
else:  
    print('x e y son iguales')
```



# IPC2 - 1.3.3. Funciones

En el contexto de la programación, una función es una secuencia de sentencias que realizan una operación y que reciben un nombre. Cuando se define una función, se especifica el nombre y la secuencia de sentencias. Más adelante, se puede “llamar” a la función por ese nombre.

Python proporciona un número importante de funciones internas, que pueden ser usadas sin necesidad de tener que definirlas previamente. Los creadores de Python han escrito un conjunto de funciones para resolver problemas comunes y las han incluido en Python para que se puedan utilizar.

```
>>> len('Hola, mundo')  
11  
>>>
```

```
>>> int(3.99999)  
3  
>>> int(-2.3)  
-2
```

# IPC2 - 1.3.3. Funciones

También existen funciones que están disponibles después de ser importadas.

```
import random

for i in range(10):
    x = random.random()
    print(x)
```

Se pueden definir funciones propias.

```
def muestra_dos_veces(bruce):
    print(bruce)
    print(bruce)
```

```
>>> muestra_dos_veces('Spam')
Spam
Spam
>>> muestra_dos_veces(17)
17
17
```



# IPC2 - 1.3.3. Funciones

Algunas de las funciones que posee python, como las matemáticas, producen resultados; a falta de un nombre mejor, se les puede llamar funciones productivas (fruitful functions). Otras funciones, realizan una acción, pero no devuelven un valor. A esas se les puede llamar funciones estériles (void functions).

```
def muestra_dos_veces(bruce):  
    print(bruce)  
    print(bruce)
```

```
def sumados(a, b):  
    suma = a + b  
    return suma  
  
x = sumados(3, 5)  
print(x)
```

# IPC2 - 1.3.3. Funciones

Puede no estar muy claro por qué merece la pena molestarse en dividir un programa en funciones. Existen varias razones:

- El crear una función nueva presenta la oportunidad de dar nombre a un grupo de sentencias, lo cual hace al programa más fácil de leer, entender y depurar.
- Las funciones pueden hacer un programa más pequeño, al eliminar código repetido. Además, permitiendo realizar cualquier cambio en el futuro, sólo haciendo modificaciones en un único lugar.
- Dividir un programa largo en funciones permite depurar las partes de una en una y luego ensamblarlas juntas en una sola pieza.
- Las funciones bien diseñadas a menudo resultan útiles para otros programas. Una vez que están escritas pueden reutilizarse.

# IPC2 - 1.3.4. Interacción

Uno de los usos habituales de las sentencias de asignación consiste en realizar una actualización sobre una variable – en la cual el valor nuevo de esa variable depende del antiguo.

```
n = 5
while n > 0:
    print(n)
    n = n - 1
print('¡Despegue!')
```

A veces no se sabe si hay que terminar un bucle hasta que se ha recorrido la mitad del cuerpo del mismo. En ese caso se puede crear un bucle infinito a propósito y usar la sentencia `break` para salir fuera de él cuando se desee.

```
while True:
    linea = input('> ')
    if linea == 'fin':
        break
    print(linea)
print('¡Terminado!')
```

# IPC2 - 1.3.4. Interacción

Algunas veces, estando dentro de un bucle se necesita terminar con la iteración actual y saltar a la siguiente de forma inmediata. En ese caso se puede utilizar la sentencia `continue` para pasar a la siguiente iteración sin terminar la ejecución del cuerpo del bucle para la actual.

```
while True:
    linea = input('> ')
    if linea[0] == '#':
        continue
    if linea == 'fin':
        break
    print(linea)
print(';Terminado!')
```

También se puede repetir un bucle a través de un conjunto de cosas, como una lista de palabras, las líneas de un archivo, o una lista de números.

```
amigos = ['Joseph', 'Glenn', 'Sally']
for amigo in amigos:
    print('Feliz año nuevo:', amigo)
print(';Terminado!')
```

# IPC2 - 1.3.4. Interacción

Para entender mejor el funcionamiento de los bucles y las interacciones que suceden en ellos se presenta el siguiente ejemplo:

```
mayor = None
print('Antes:', mayor)
for valor in [3, 41, 12, 9, 74, 15]:
    if mayor is None or valor > mayor :
        mayor = valor
    print('Bucle:', valor, mayor)
print('Mayor:', mayor)
```

Este ejemplo presenta como durante la ejecución el programa interactúa con las variables mayor y la lista valor para lograr encontrar el número de mayor valor en la lista.

# IPC2 - 1.3.5. Strings

Una cadena es una secuencia de caracteres. Se puede acceder a los caracteres de uno en uno con el operador corchete:

```
>>> fruta = 'banana'
>>> letra = fruta[1]
```

```
>>> print(letra)
a
```

Manejar un texto como una lista de caracteres es bastante útil en la programación, por esto python interpreta un valor string como una lista de caracteres, lo cual se demuestra con el siguiente ejemplo:

```
indice = 0
while indice < len(fruta):
    letra = fruta[indice]
    print(letra)
    indice = indice + 1
```

# IPC2 - 1.3.5. Strings

Otro uso posible debido a que los string son listas de caracteres, es segmentar la cadena, esto permite analizar el texto, por ejemplo:

```
>>> s = 'Monty Python'
>>> print(s[0:5])
Monty
>>> print(s[6:12])
Python
```

Pero una diferencia en este tipo de lista, es que un string es una cadena inmutable, lo cual quiere decir que no se puede reasignar el valor a alguno de los elementos en dicha lista:

```
>>> saludo = 'Hola, mundo!'
>>> saludo[0] = 'J'
TypeError: 'str' object does not support item assignment
```

# IPC2 - 1.3.5. Strings

Se puede analizar el texto de una manera muy minuciosa. El siguiente programa cuenta el número de veces que la letra "a" aparece en una cadena:

```
palabra = 'banana'
contador = 0
for letra in palabra:
    if letra == 'a':
        contador = contador + 1
print(contador)
```

La palabra `in` es un operador booleano que toma dos cadenas y regresa `True` si la primera cadena aparece como una subcadena de la segunda:

```
>>> 'a' in 'banana'
True
>>> 'semilla' in 'banana'
False
```



# IPC2 - 1.3.5. Strings

Los cadenas son un ejemplo de objetos en Python. Un objeto contiene tanto datos (el valor de la cadena misma) como métodos, los cuales son efectivamente funciones que están implementadas dentro del objeto y que están disponibles para cualquier instancia del objeto.

Python tiene una función llamada `dir` la cual lista los métodos disponibles para un objeto. La función `type` muestra el tipo de un objeto y la función `dir` muestra los métodos disponibles.

```
>>> cosa = 'Hola mundo'
>>> type(cosa)
<class 'str'>
>>> dir(cosa)
['capitalize', 'casefold', 'center', 'count', 'encode',
 'endswith', 'expandtabs', 'find', 'format', 'format_map',
 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
 'isidentifier', 'islower', 'isnumeric', 'isprintable',
 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
 'lstrip', 'maketrans', 'partition', 'replace', 'rfind',
 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
 'split', 'splitlines', 'startswith', 'strip', 'swapcase',
 'title', 'translate', 'upper', 'zfill']
>>> help(str.capitalize)
Help on method_descriptor:

capitalize(...)
    S.capitalize() -> str

    Return a capitalized version of S, i.e. make the first character
    have upper case and the rest lower case.

>>>
```

# Programación Orientada a Objetos

# Programación Orientada a Objetos (POO)

La programación Orientada a objetos es un paradigma de programación, una manera de programar específica, donde se organiza el código en unidades denominadas clases, de las cuales se crean objetos que se relacionan entre sí para conseguir los objetivos de las aplicaciones. Los pilares de la POO son abstracción, encapsulamiento, herencia, polimorfismo.

Elementos básicos de POO:

- Clases
- Atributos
- Métodos



# Objetos y Clases

Un objeto es un simple elemento, que representa cualquier cosa en particular, no importa lo complejo que pueda ser. Los objetos se almacenan comúnmente en variables.

Una clase es la descripción de un **conjunto de objetos** similares; consta de métodos y de datos que resumen las **características comunes** de dicho conjunto. En pocas palabras, una clase es la declaración de un tipo de objeto o entiéndase como una plantilla.



# Atributos y Métodos

## Atributos

Cada objeto dispone de una serie de atributos que definen sus características individuales y le permiten diferenciarse de otros (apariencia, estado, etc).

## Método

Es una subrutina o funcionalidades que puede pertenecer a un objeto y que son definidos en una clase, y son una serie de sentencias para llevar a cabo una acción.



# Ejemplo

Objeto

p1=

Nombre: "Rebeca"  
Edad: 16  
Género: Femenino  
+ Vivir()

Objeto

p2 =

Nombre: "Alejandro"  
Edad: 21  
Género: Masculino  
+ Vivir()

Clase

Nombre:  
Edad:  
Género:  
+ Vivir()

# Herencia

Es el pilar más fuerte que asegura la reutilización de código, ya que a partir de esta característica es posible reutilizar (heredar) las características y comportamientos de una clase superior llamada clase padre, a sus clases hijas, denominadas clases derivadas.

Esto implica que una vez desarrollado el código de una clase base, su código puede ser reutilizado por las clases derivadas.



## Persona

- CUI
- Nombre
- Edad

## Profesor

(Heredadas)

- CUI
- Nombre
- Edad
- (Propias)
- Designación
- Título

## Estudiante

(Heredadas)

- CUI
- Nombre
- Edad
- (Propias)
- Carnet
- Carrera



## Abstracción

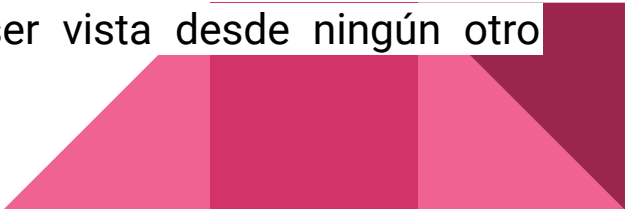
Es el pilar de la POO, que permite identificar las características y comportamientos de un objeto y con los cuales se construirá una clase. Permite reconocer los atributos y métodos de un objeto.

## Polimorfismo

A través de esta característica es posible definir varios métodos o comportamientos de un objeto bajo un mismo nombre, de forma tal que es posible modificar los parámetros del método, o reescribir su funcionamiento, o incrementar más funcionalidades a un método.

## Encapsulamiento

Es la característica de la POO que permite el ocultamiento de la complejidad del código, pertenece a la parte privada de la clase y que no puede ser vista desde ningún otro programa. Acceso a un código desde el código de otra clase.





# Ficheros

# Abrir Archivo


Cuando se desea leer o escribir en un archivo (nos referimos en el disco duro), primero debemos abrir el fichero. Al abrir el fichero nos comunicamos con el sistema operativo, que sabe donde se encuentran almacenados los datos de cada archivo.

```
file = open('ruta del archivo')
```

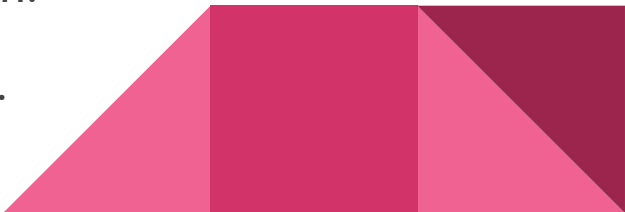
La función open puede tener varios argumentos. Los más importantes son

with open("FICHERO", mode="MOD0", encoding="CODIFICACIÓN") as fichero:

"FICHERO" es la ruta absoluta o relativa hasta el fichero. "MOD0" indica si el fichero se abre para leer, escribir o ambas cosas y si se trata de un fichero de texto o binario. El modo predeterminado es lectura en modo texto. "CODIFICACIÓN" indica el juego de caracteres del fichero: "utf-8" (UTF-8), "cp1252" (ASCII para Europa occidental).



# Modos

- r Solo lectura. El fichero solo se puede leer. Es el modo por defecto si no se indica.
  - w Sólo escritura. En el fichero solo se puede escribir. Si ya existe el fichero, machaca su contenido.
  - a Adición. En el fichero solo se puede escribir. Si ya existe el fichero, todo lo que se escriba se añadirá al final del mismo.
  - x Como 'w' pero si existe el fichero lanza una excepción.
  - r+ Lectura y escritura. El fichero se puede leer y escribir.
- 

# Leer un Archivo

Para leer un archivo, utilizamos el método 'read'. Por defecto, al abrir un archivo se abre en modo lectura 'r'.

Ejemplo:

```
archivo = open('Personas.txt')
```

```
contenido = archivo.read()
```

```
print(contenido)
```

```
archivo.close()
```



# Escribir un archivo

El método es 'write'. Pero antes que nada Para escribir en un fichero, debes abrirlo usando el modo 'w' (de write), indicandolo en el segundo parámetro.

Ejemplo:

```
f=open("curso.txt","w")
```

```
f.write("IPC2\n")
```

```
f.close()
```



# Try y Except

Al momento de manejar archivos, es importante tomar en consideración la posibilidad que un archivo no exista, para evitar que la ejecución se detenga podemos usar los comandos try y except.

Ejemplo:

try:

```
file = open(nombre_archivo)
```

except:

```
print('No se pudo abrir el fichero: ' + nombre_archivo)
```

```
exit()
```





Fin.