

# Trabajo Práctico 0

**Materia: Algoritmos y Programación I**

**Cátedra: Ing. Pablo Guarna.**



## **Objetivo:**

Instalar el entorno necesario para poder programar en lenguaje Python y comenzar a familiarizarse con el lenguaje en cuestión.

## **Preparando el sistema:**

Lo primero que se debe hacer es instalar [Python](https://www.python.org/). Desde la cátedra damos libertad de elección para que lo instalen y lo usen de la forma que el alumno prefiera en el sistema operativo que le sienta más cómodo. Descargar la versión 3.7.3 (la más actual) ya que es la que usaremos en el curso. (<https://www.python.org/>).

Desde la cátedra recomendamos utilizar VisualStudio Code, un editor de texto que permite instalar distintos plugins (adicionales) que servirán para programar en Python. Otra opción es usar Sublime Text. Los alumnos son libres de elegir el editor que quieran, aunque recomendamos utilizar alguno de los nombrados anteriormente.

También podemos hacer uso de un [IDE](#) que nos facilitará la tarea de programar. Su nombre es PyCharm y pueden descargar la versión *community* desde este link: <https://www.jetbrains.com/pycharm/download/>. También pueden gestionar la versión *professional* con alguna acreditación de la facultad, pero para este curso es absolutamente innecesario.

Luego de haber instalado Python, podemos comprobar que lo hicimos correctamente.

## **Verificando instalación en Windows:**

Abrimos el símbolo del sistema (la “consola”), apretando Windows+R, escribiendo cmd y apretando la tecla enter.

Una vez que nos aparezca la línea de comandos, escribimos “**py --version**” y a continuación se nos indicará la versión de Python que hemos instalado previamente. Si escribimos “**py**” podemos comenzar a escribir código en la *command prompt*, y luego para salir usamos el comando “**exit()**”. En caso de tener instalada otra versión de Python, debemos utilizar “**py -3.7**”.

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 10.0.10240]
(c) 2015 Microsoft Corporation. Todos los derechos reservados.

C:\Users\nora>py --version
Python 3.6.4

C:\Users\nora>py
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hola mundo")
hola mundo
>>> exit()

C:\Users\nora>py -3.6
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()

C:\Users\nora>
```

## Verificando instalación en Linux:

En Linux podemos instalar Python directamente desde la línea de comandos. Para esto ponemos en una terminal (la abrimos con Ctrl+Alt+T):

### **sudo apt-get install python3**

En muchas distribuciones ya viene instalado, en ese caso asegurarse de que tengamos Python 3 y no Python 2 y en caso de no tener la última versión, volverlo a instalar.

Para verificar que versión de Python tenemos, dentro de la terminal ponemos el comando **“python --version”** y a continuación se nos indicará la versión de Python que hemos instalado previamente. Si escribimos **“python”** podemos comenzar a escribir código en la *command prompt*, y luego para salir usamos el comando **“exit()”**. En caso de tener instalada otra versión de Python, debemos utilizar **“python3”**.

```
uriel@uriel-Lenovo-ideapad-100-14IBD: ~
uriel@uriel-Lenovo-ideapad-100-14IBD:~$ python --version
Python 3.6.4 :: Anaconda, Inc.
uriel@uriel-Lenovo-ideapad-100-14IBD:~$ python3
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 18:10:19)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hola mundo")
hola mundo
>>> exit()
uriel@uriel-Lenovo-ideapad-100-14IBD:~$
```

**Consigna:** Del campus han descargado un paquete con tres archivos (tp0.py, test\_tp0.py, y el archivo del enunciado).

tp0.py: Este archivo contiene la firma de cinco funciones con su respectiva documentación (pre y post condiciones). Su tarea es programar la funcionalidad indicada en la documentación sobre este mismo archivo .py.

Las **pre-condiciones** son aquellas condiciones que se cumplen en un sistema antes de ejecutar determinada función: ejemplo típico de esto es que los parámetros que se le pasan a la función no pueden ser nulos.

Las **post-condiciones** son aquellas condiciones que se deben cumplir cuando se termina de ejecutar la función. En nuestro caso, indican que nos devuelve la misma y permiten detectar la funcionalidad que debemos programar.

ACLARACIÓN IMPORTANTE: No deben modificar las firmas (nombres y cantidad de parámetros que recibe) de las funciones, solamente programar lo pedido.

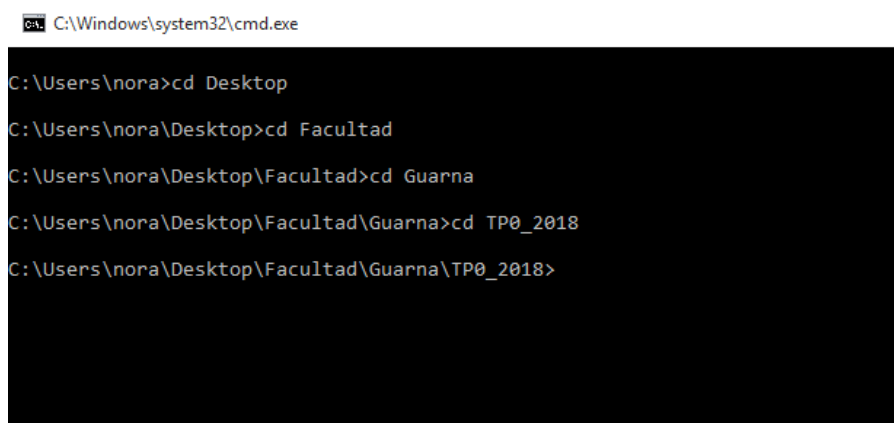
test\_tp0.py: Este archivo contiene pruebas para cada una de las cinco funciones. **La condición para aprobar el trabajo práctico es hacer que estas pruebas demuestren que la funcionalidad programada es la pedida por la documentación.** Para esto debemos correr las pruebas y verificar que todas dan OK.

### Corriendo las pruebas:

La forma de correr las pruebas varía dependiendo de si utilizamos un editor de texto (como puede ser Sublime Text) o PyCharm (recomendamos enfáticamente utilizarlo).

### Editor de texto (Windows):

Desde la cmd, utilizamos el comando “cd” hasta llegar a la carpeta donde tengamos descargados los archivos:



```
C:\Windows\system32\cmd.exe
C:\Users\nora>cd Desktop
C:\Users\nora\Desktop>cd Facultad
C:\Users\nora\Desktop\Facultad>cd Guarna
C:\Users\nora\Desktop\Facultad\Guarna>cd TP0_2018
C:\Users\nora\Desktop\Facultad\Guarna\TP0_2018>
```

Una vez dentro de la carpeta, ejecutamos las pruebas con el comando: “py test\_tp0.py”. Este comando sirve para ejecutar cualquier archivo de extensión .py.

```
C:\Windows\system32\cmd.exe

Traceback (most recent call last):
  File "test_tp0.py", line 39, in test_mayorProducto
    self.assertEqual(mayorProducto(1,2,3), 6)
AssertionError: None != 6

=====
FAIL: test_numeroEsPar (__main__.TestTP0)
-----
Traceback (most recent call last):
  File "test_tp0.py", line 33, in test_numeroEsPar
    self.assertTrue(esPar(2))
AssertionError: None is not true

=====
FAIL: test_sumarLetrasDeDistintasPalabras (__main__.TestTP0)
-----
Traceback (most recent call last):
  File "test_tp0.py", line 9, in test_sumarLetrasDeDistintasPalabras
    self.assertEqual(sumarLetras(palabra1, palabra2), 8)
AssertionError: None != 8

-----
Ran 5 tests in 0.004s

FAILED (failures=5)

C:\Users\nora\Desktop\Facultad\Guarna\TP0_2018>_
```

Notemos que debajo de todo, aparece la leyenda “FAILED (failures = 5)”, y arriba de la leyenda una descripción de todas las fallas que hubo. Una vez que hayamos programado lo pedido para cada una de las funciones, debería aparecer lo siguiente:

```
C:\Windows\system32\cmd.exe

C:\Users\nora\Desktop\Facultad\Guarna\TP0_2018>py test_tp0.py
test_tp0.py:29: DeprecationWarning: Please use assertEqual instead.
  self.assertEqual(cantidadDeSegundos(1,3,117), 3897)
.....
-----
Ran 5 tests in 0.003s

OK

C:\Users\nora\Desktop\Facultad\Guarna\TP0_2018>
```

NOTA: Pueden interpretar las fallas y leer las pruebas cuando crean que están programando algo bien pero sin embargo las pruebas no corran correctamente.

### Editor de texto (Linux):

Desde la terminal, usamos el comando “cd” para navegar hasta la carpeta donde tengamos los archivos del TP.

```

uriel@uriel-Lenovo-ideapad-100-14IBD: ~/Escritorio/Facultad/Guarna/TP0-PrimerCuatri20
uriel@uriel-Lenovo-ideapad-100-14IBD:~$ cd Escritorio/Facultad/Guarna/TP0-PrimerCuatri2018/
uriel@uriel-Lenovo-ideapad-100-14IBD:~/Escritorio/Facultad/Guarna/TP0-PrimerCuatri2018$

```

Una vez dentro de la carpeta, ejecutamos las pruebas con el comando: “python3 test\_tp0.py”. Este comando sirve para ejecutar cualquier archivo de extensión .py.

```

uriel@uriel-Lenovo-ideapad-100-14IBD: ~/Escritorio/Facultad/Guarna/TP0-PrimerCuatri20
AssertionError: None != 6
=====
FAIL: test_numeroEsPar (__main__.TestTP0)
-----
Traceback (most recent call last):
  File "test_tp0.py", line 33, in test_numeroEsPar
    self.assertTrue(esPar(2))
AssertionError: None is not true
=====
FAIL: test_sumarLetrasDeDistintasPalabras (__main__.TestTP0)
-----
Traceback (most recent call last):
  File "test_tp0.py", line 9, in test_sumarLetrasDeDistintasPalabras
    self.assertEqual(sumarLetras(palabra1, palabra2), 8)
AssertionError: None != 8
-----
Ran 5 tests in 0.001s

FAILED (failures=5)
uriel@uriel-Lenovo-ideapad-100-14IBD:~/Escritorio/Facultad/Guarna/TP0-PrimerCuatri2018$

```

Notemos que debajo de todo, aparece la leyenda “FAILED (failures = 5)”, y arriba de la leyenda una descripción de todas las fallas que hubo. Una vez que hayamos programado lo pedido para cada una de las funciones, debería aparecer lo siguiente:

```

uriel@uriel-Lenovo-ideapad-100-14IBD: ~/Escritorio/Facultad/Guarna/TP0-PrimerCuatri20
uriel@uriel-Lenovo-ideapad-100-14IBD:~/Escritorio/Facultad/Guarna/TP0-PrimerCuatri2018$ python3 test_tp0.py
test_tp0.py:29: DeprecationWarning: Please use assertEqual instead.
  self.assertEqual(cantidadDeSegundos(1,3,117), 3897)
.....
Ran 5 tests in 0.001s

OK
uriel@uriel-Lenovo-ideapad-100-14IBD:~/Escritorio/Facultad/Guarna/TP0-PrimerCuatri2018$

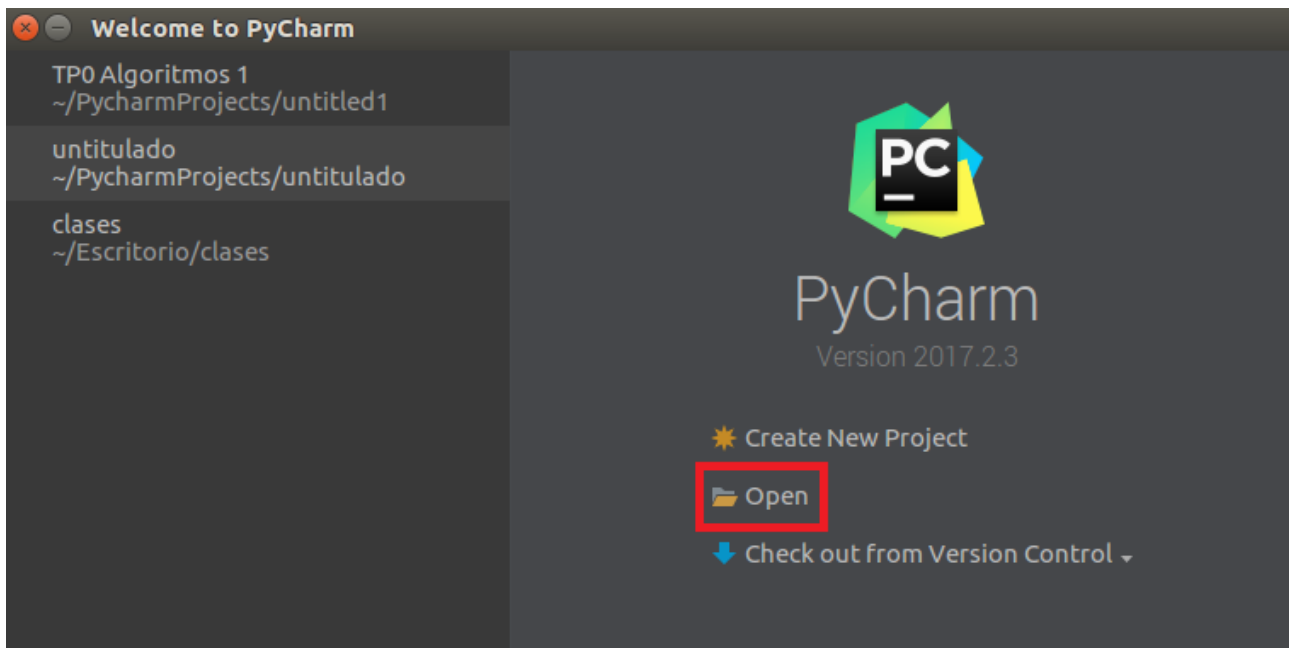
```

NOTA: Pueden interpretar las fallas y leer las pruebas cuando crean que están programando algo bien pero sin embargo las pruebas no corran correctamente.

### **PyCharm (cualquier sistema operativo):**

El IDE se encuentra tanto para Windows como para Linux. En la página hay tutoriales de como instalarlo y también de cómo cambiar la configuración.

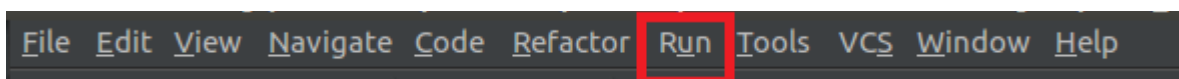
Una vez que hayamos instalado, abrimos el archivo de ejecución (.exe en Windows) y nos saldrá una ventana como ésta:



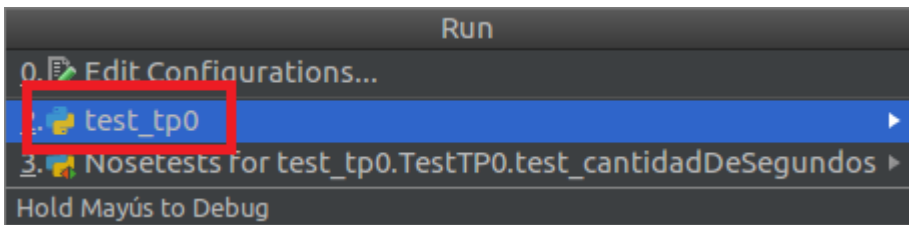
Clickeamos en “Open” y seleccionamos la carpeta donde tengamos instalados los archivos.

Sobre el panel izquierdo aparecerá la carpeta con los archivos. Para abrirlos y comenzar a editarlos basta hacer doble click sobre ellos. Recordar que solamente hay que modificar el archivo con las funciones, aunque debemos abrir el archivo de pruebas para poder correrlo.

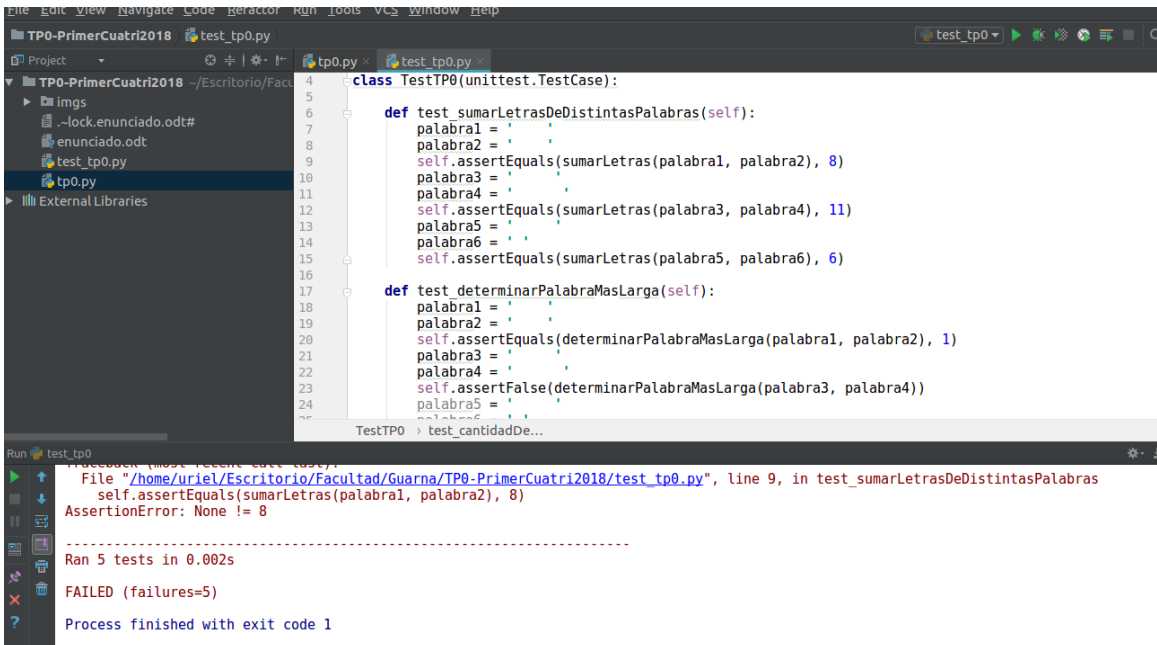
Una vez abiertos los archivos, podemos correr las pruebas. Para esto, nos paramos en el archivo de pruebas y en la barra superior clickeamos en “Run”:



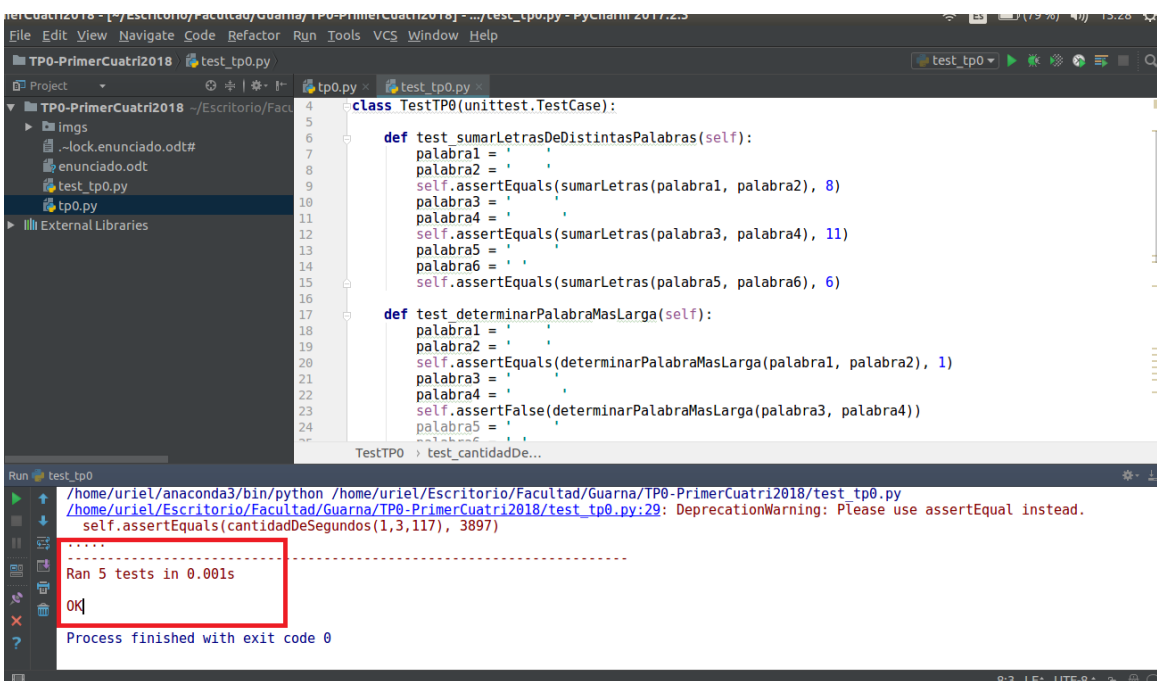
Allí hacemos click en “Run” nuevamente y nos saldrá la siguiente ventana, donde seleccionaremos el archivo de pruebas con un doble click:



¡Y listo! Ya corrimos las pruebas. Si las pruebas nos indican que aún tenemos errores en nuestro código, debería aparecernos una imagen como esta:



Notemos que debajo de todo, aparece la leyenda “FAILED (failures = 5), y arriba de la leyenda una descripción de todas las fallas que hubo. Una vez que hayamos programado lo pedido para cada una de las funciones, debería aparecer lo siguiente:





## **Condiciones de aprobación y forma de entrega:**

El TP será entregado de forma individual por todos los alumnos. No llevará nota numérica, las calificaciones posibles son aprobado o desaprobado. Para aprobar el TP es necesario hacer que las pruebas corran y den todas OK; y en caso de tener algún problema o duda se puede preguntar por el foro del campus o a algún ayudante. Se recomienda NO DEJAR EL TP PARA EL ÚLTIMO DÍA DE ENTREGA. Si bien es sencillo, tener un solo problema que no sepamos solucionar nos puede llevar a desaprobarlo.

La entrega será realizada vía campus, donde se habilitará una sección especial para que suban el TP. El tiempo será hasta el jueves 04-04-2019 hasta la medianoche. No serán aceptadas entregas fuera de término.

Se debe entregar un zip con su nombre bajo el siguiente formato “{nombre apellido} – {padron}” (por ejemplo, el nombre de un zip podría ser “Juan Perez – 100000”), que incluya:

- todos los archivos que estaban en el zip del enunciado, es decir:
  - test\_tp0.py
  - tp0.py
  - este enunciado.

Donde en tp0.py debe estar incluido el código que resuelve el tp.