Programación en Python
TEMA 2

Thonny y primeros pasos

Universidad Politécnica de Valencia

2021-2022

Índice

1. ¿Qué es Thonny?

Thonny es un entorno integrado de desarrollo (IDE) pensado para principiantes. Es sencillo de usar, consume pocos recursos y además ya incorpora el interprete de Python.

Contiene un debugger simple que permite realizar la ejecución paso a paso de un programa, viendo el contenido de las variables y como se realizan las llamadas a las funciones de manera que resulta muy útil no solo para corregir posibles errores sino también para interpretar mejor el funcionamiento del programa.

1.1. Interfaz de Thonny

La pantalla de Thonny está compuesta de varias ventanas. Esta presentación es configurable pero nosotros usaremos la mayoría de veces la disposición que se ve en la Figura 1 y que explicamos a continuación.

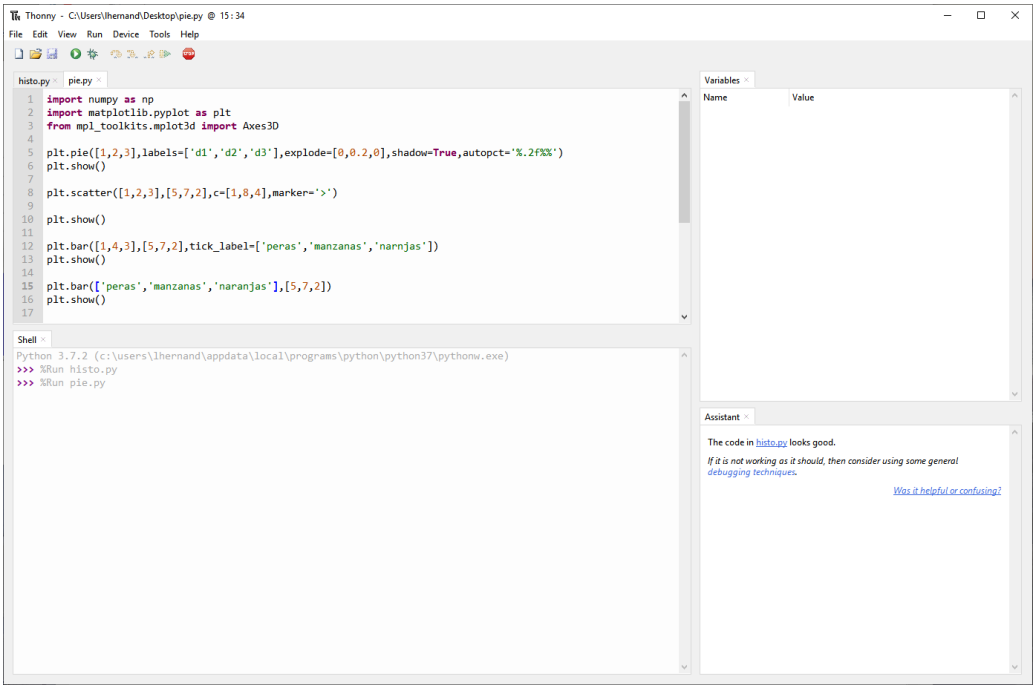

Figura 1: Interfaz de Thonny

En la parte superior encontramos el menú con las diferentes opciones que podemos usar e inmediatamente debajo la barra de herramientas que simplemente son atajos a las funciones del menú más usadas. Debajo de ella nos encontramos las ventanas principales de trabajo: En la parte izquierda encontramos el editor de código donde podremos crear y editar nuestros programas. Debajo de ella se encuentra el shell o consola de ejecución, mediante ella nos podremos comunicar con el programa, ya sea introduciendo datos o viendo los resultados que este presenta. También servirá para ejecutar de forma interactiva ordenes de Python.

En la parte superior derecha vemos la ventana de variables, que nos permitirá conocer el valor de las variables del programa en cada momento. En la parte inferior está el asistente que nos dará consejos e indicaciones sobre los errores de nuestro programa.

1.2. Creación de un nuevo programa

Los programas se almacenan en archivos. Puedes crear un nuevo archivo en cualquier momento mediante la opción del menú (File/New), el atajo de teclado Ctrl+N o el botón de un pagina en blanco de la barra de herramientas.

1.3. Guardado de programas

Para guardar el programa que acabamos de escribir, podemos usar la opción del menú (New/Save), el atajo de teclado Ctrl+S o el botón de un disquete de la barra de herramientas. Si más adelante deseamos abrir este programa, basta con usar la opción del menú (New/Open), el atajo de teclado Ctrl+O o el botón de una carpeta de la barra de herramientas.

1.4. Edición de programas

El editor de código dispone de una serie de opciones típicas de los editores de texto como son:

- Copiar (Edit/Copy) o Ctrl+c
- Pegar (Edit/Paste) o Ctrl+v
- Cortar (Edit/Cut) o Ctrl+x
- Deshacer última orden (Edit/Undo) o Ctrl+z
- Buscar y reemplazar (Edit/ Find & replace) o Ctrl+F

y otras propias para programar como:

- Comentar código (Edit/Toggle comment) o Ctrl+3
- Quitar comentario (Edit/Comment out) o Alt +3
- Auto-completado (Edit/Auto-complete) o Ctrl+Espacio

Esta ultima opción es particularmente útil para, por ejemplo, mostrar el listado de las funciones contenidas en una librería. Al pulsar esta combinación después del punto de una librería aparece un cuadro desplegable que ofrece información sobre todas las funciones y constantes matemáticas incluidas en dicha librería (ver Figura 2).

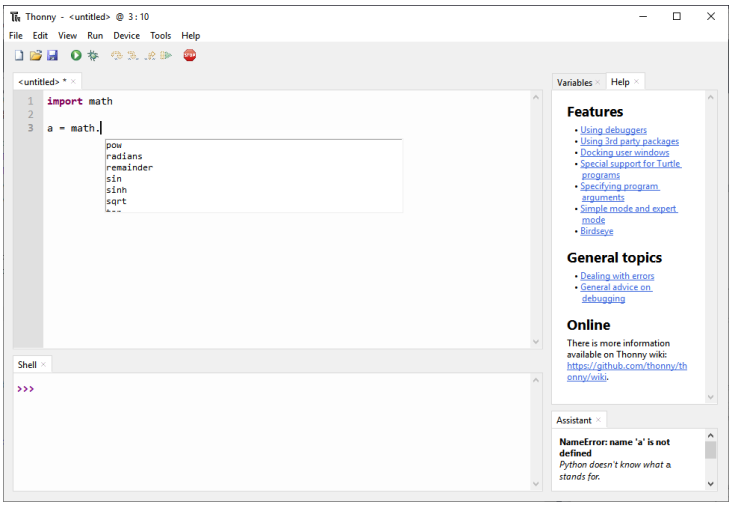

Figura 2: Uso del auto-completado

1.5. Ejecución y testing de programas

Para ejecutar un programa, podemos usar la opción del menú (Run/Run current script), el atajo de teclado F5 o el botón de una flecha verde de la barra de herramientas. Es posible que Thonny pregunte la primera vez dónde guardar el programa si no lo hemos hecho antes. Podemos terminar la ejecución del programa con Ctrl+c. Si queremos parar y restaurar el entorno borrando todas las variables que teníamos, la opción es (stop/restart backend), Ctrl+F2 o pulsando el icono de stop.

Tenemos que ejecutar nuestros programas para poder hacer tests que nos ayuda validar que el programa funciona como esperamos.

1.6. Depuración de programas

Cuando encontramos algún error en nuestro programa mientras ejecutamos tests, hay que buscar de donde viene. El depurador nos puede ayudar.

Con el depurador, en lugar de ejecutar el programa completo, podemos ejecutarlo poco a poco de forma que podamos comprobar si todo sucede como esperábamos. Esto se conoce como **depurar**, y se lleva a cabo mediante la opción del menú (Run/Debug current script (nicer)), el atajo de teclado Ctrl+F5 o el botón con la imagen de un insecto de la barra de herramientas. La ejecución del programa se detendrá entonces en la primera línea. La línea actual aparece resaltada en el editor de código.

Para avanzar en la ejecución disponemos de las siguientes opciones dentro del menú Run:

- Step Over (o F6): ejecuta la línea actual y avanza hasta la siguiente.
- Step into (o F7): entra en la llamada a la función que haya en la línea actual.
- Step out: avanza hasta el final de la función actual. Esta opción y la anterior nos serán útiles cuando veamos funciones.
- Resume (o F8): continúa, sin detenerse, la ejecución del programa hasta que termine el programa.

Durante la depuración, la herramienta de mayor utilidad que tenemos es el explorador de variables (ver Figura 2). En esta ventana se muestran las variables que se van definiendo en el programa y los valores que contienen. Esto permite comprobar si los cálculos que vamos realizando obtienen los resultados esperados.

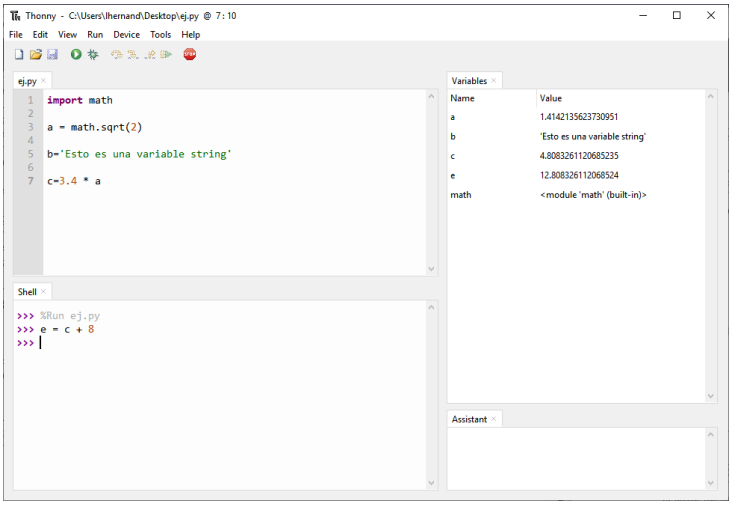

Figura 3: Explorador de variables

Los valores de las variables se mantienen en el explorador aunque hayamos terminado la ejecución o depuración del programa. Del mismo modo, también muestra las variables que hayamos definido en el

shell. Tener toda esta información puede confundir si tratamos de depurar el programa de nuevo. Antes de iniciar otra depuración podemos eliminar todas las variables ya existentes, por ello es útil la opción de Stop comentada anteriormente. También tenemos la opción de limpiar el terminal si no queremos ver todo lo que se ha mostrado hasta ahora en el shell simplemente pulsando con el botón derecho sobre el shell y eligiendo clear.

1.7. Otras opciones del menú

Además de las diversas opciones del menú de Thonny hay algunas más que resultan interesantes;

- View. Nos permite elegir que ventanas queremos que aparezcan en el entorno, así como cambiar el tamaño de fuente.
- Tools. Aquí es interesante la opción de 'Manage packages' que nos permite instalar otros paquetes que no vienen en la instalación estándar de Thonny

2. Nuestro primer programa.

El fin de un programa es que realice correctamente la tarea que se le ha asignado. Nuestros programas se escribirán en el lenguaje de programación Python y que se almacene en el disco para emplearlo más de una vez. Cuando escribimos programas es muy importante testear bien nuestro código para asegurar que hace que queremos que haga. Vamos pues a ver cuál es la dinámica para escribir y testear un programa Python en nuestro editor Thonny.

Empezaremos por crear el fichero en el que escribiremos el programa. Selecciona File/New del menú. Nos creará un nuevo archivo sin titulo. Ahora haz File/Save en el menú y guárdalo con el nombre *perimetro.py*. Escribe ahora el siguiente código:

perimetro.py

```
from math import pi

radio = 1
perimetro = 2*pi*radio
print(perimetro)
```

Este programa empieza cargando de la librería matemática de Python denominada *math* la constante numérica *pi*. Esta librería contiene la mayoría de las funciones matemáticas, trigonométricas y constantes como puede ser el seno, coseno, valor absoluto, el numero e etc...

A partir de la instrucción anterior, cualquier referencia que hagamos a la contante *pi* será el valor real 3.141592653589793.

A continuación definimos la variable *radio* y le damos el valor 1 (entero) con la instrucción de asignación (=). Una vez tenemos el valor del radio y el valor de π podemos calcular la fórmula del perímetro que es:

$$perimetro = 2 * \pi * radio$$

En nuestro caso la constante numérica π la obtenemos de la librería matemática y se llama *pi*

Por ultimo, para ver el resultado ejecuta el programa con la tecla F5 o con la flecha verde del menú de iconos de la parte superior de *Thonny*.

El resultado se muestra en la pantalla de *S*hell:

```
%Run perimetro.py

6.283185307179586
```

2.1. Entrada/salida de teclado

El programa que hemos visto adolece de un serio inconveniente: cada vez que quieras obtener resultados para unos datos diferentes deberás editar el fichero de texto que contiene el programa.

Por ejemplo, crea un nuevo programa llamado *esfera.py* con Archivo/Nuevo o Crtl+n y guárdalo en una carpeta llamada Boletin1 escribe en el editor el siguiente programa calcula el volumen de una esfera a partir de su radio, que es de un metro:

esfera.py

```
from math import pi

radio = 1
volumen = 4 / 3 * pi * radio ** 3
print (volumen)
```

Ejecútalo con la flecha verde o F5 y te dará:

```
4.1887902047863905
```

Si deseas calcular ahora el volumen de una esfera de 3 metros de radio, debes editar el fichero que contiene el programa, yendo a la tercera línea y cambiándola para que el programa pase a ser este:

esfera.py

```
from math import pi

radio=3
volumen =4 / 3 * pi * radio ** 3
print (volumen)
```

Al ejecutar nuevamente el programa obtenemos en pantalla este otro texto:

```
113.09733552923254
```

Y si ahora quieres calcular el volumen para otro radio, vuelta a empezar: ve a la tercera línea, modifica el valor del radio y ejecuta. No es el colmo de la comodidad.

2.2. Lectura de datos de teclado: input()

Vamos a aprender a hacer que nuestro programa, cuando se ejecute, pida el valor del radio para el que vamos a efectuar los cálculos sin necesidad de editar el fichero de programa. Hay una función predefinida, *input* (en inglés significa «entrada»), que hace lo siguiente: detiene la ejecución del programa y espera a que el usuario escriba un texto (el valor del radio, por ejemplo) y pulse la tecla de retorno de carro (enter/intro); en ese momento prosigue la ejecución y la función devuelve una cadena con el texto que tecleó el usuario. Si deseas que el radio sea un valor flotante (real), debes transformar la cadena devuelta por input en un dato de tipo flotante (real) llamando a la función float. La función float recibirá como argumento la cadena que devuelve input y proporcionará un número en coma flotante. (Recuerda, para cuando lo necesites, que existe otra función de conversión, int, que devuelve un entero en lugar de un flotante).

Por otra parte, input es una función y, por tanto, el uso de los paréntesis que siguen a su nombre es obligatorio, incluso cuando no tenga argumentos. Modificamos el fichero anterior para que quede de la siguiente forma:

esfera.py

```
from math import pi

cadena_leida = input()
radio = float(cadena_leida)
```

```
volumen = 4 / 3 * pi * radio ** 3
print (volumen)
```

Al ejecutar se queda esperando que se introduzca un valor en la pantalla de Shell, introduce por ejemplo un 2, obtendrás:

```
2
33.510321638291124
```

Puedes ejecutar más tests, pulsando F5 para ejecutar el programa y poniendo otros números como radio. Por ejemplo los siguientes tests:

```
>>> %Run esfera.py
0
0.0
>>> %Run esfera.py
-5
-523.5987755982989
>>> %Run esfera.py
6.00013
904.8374961225749
```

2.3. Variables

Las variables adquieren un **nombre** totalmente libre que elegimos a nuestra conveniencia. Puede ser una palabra, una simple letra o varias palabras unidas por un "_". Sólo hay que tener en cuenta que:

- no deben comenzar con un número aunque lo pueden contener.
- Python distingue entre letras minúsculas y mayúsculas
- Las palabras reservadas por el lenguaje y no pueden usarse como nombres de variables

Existen caracteres que no pueden utilizarse en el nombre pero si perseguimos la sencillez en la elección del nombre (letras y números) no necesitaremos conocer estos caracteres prohibidos.

2.4. Tipos básicos

Python tiene 4 tipos básicos de valores

- los enteros (`int`)
- los reales (`float`)
- los booleanos (`bool`)
- las cadenas o strings (`str`)

Cuando guardamos valores de estos tipos en una variable, el tipo determina el carácter de esta variable declarada (tamaño y operaciones permitidas).

Podemos convertir de un tipo a otro. Por ejemplo:

- Para enteros: int. P.e. `int(2.3)` seria igual a 2.
- Para reales: float. P.e. `float(2)` sería igual a 2.0.
- Para caracteres: str. P.e. `str(123)` sería la cadena "123"(que no el numero 123).

Por defecto la entrada de datos por teclado con la instrucción `input` lee una cadena. Si se precisa otro tipo, por ejemplo un real (float) o en entero (int), hace falta **convertir el tipo** de la cadena explícitamente al tipo deseado.

2.5. Más sobre la función print y las cadenas o strings.

Las cadenas pueden usarse también para mostrar textos por pantalla en cualquier momento a través de sentencias print. Escribe el siguiente programa:

```
from math import pi

print("Programa para el cálculo del volumen de una esfera.")
radio=float(input("Dame el radio:")
volumen =4 / 3 * pi * radio ** 3
print ('volumen:',volumen,'metros cúbicos.')
print ('Gracias por usar el programa.')
```

Como puedes observar las cadenas se pueden escribir con comilla simple o doble.

La primera aparición de `print` muestra en pantalla un mensaje que informa al usuario del propósito del programa. La segunda aparición de `print` muestra dos cosas en pantalla: el texto «Volumen:», el valor del volumen de la esfera y las unidades en las que se expresa el volumen. La función `print` puede mostrar en una misma línea más de un valor: los valores que se desee mostrar van entre paréntesis y separados por comas. Finalmente, la última aparición de `print` hace que se muestre un texto de agradecimiento y despedida.

Resultado:

```
Programa para el cálculo del volumen de una esfera.
Dame el radio:2
volumen: 33.510321638291124 metros cúbicos.
Gracias por usar el programa.
```

3. Ejercicios

Ejercicio 1. La mayor parte de los programas que implementamos hacen 3 cosas:

1. recibir datos de entrada (*input data*) (por ejemplo a través del teclado)
2. hacer algo con estos datos, por ejemplo realizar algún tipo de computación
3. mostrar el resultado como una salida (*output data*) (por ejemplo a través de la pantalla)

A veces, a la combinación teclado/pantalla se le conoce con el nombre de *consola* mediante el cuál se identifica indistintamente a ambos.

1. La entrada de datos se realiza mediante la instrucción `input` como en el siguiente ejemplo:

```
a = int(input("Introduce un numero: "))
```

Mediante esta instrucción pedimos al usuario de introducir un dato por teclado. Evidentemente, el dato introducido por teclado habrá que almacenarlo en algún sitio de la memoria, es decir, en una variable. La instrucción anterior indica que el dato introducido por consola se interpretará como un número entero (`int`) y será almacenado en la variable `a`.

2. Calcular el cuadrado del numero en `a` es un ejemplo de una computación:

```
cuadrado = a * a
```

Esta instrucción calcula el cuadrado de `a` y guarda el resultado en la variable `cuadrado`.

3. Mostrar un resultado se puede hacer con la instrucción `print`, como por ejemplo:

```
print("El cuadrado es: ", cuadrado)
```

Ejecutar este programa en la consola, podemos obtener por ejemplo:

ejemplo de ejecución

```
>>> %Run
  Introduce un numero: 5
  El cuadrado es: 25
```

Tenga en cuenta que el programa debe funcionar para cualquier número introducido por el usuario. Cuando escribimos programas con los que podemos interactuar a través de la consola, podemos testear nuestro programa ingresando datos de entrada de prueba a través del teclado y verificando la salida resultante en la pantalla. Para hacer tests con nuestro programa y ver si también funciona para otros números, podríamos ejecutar las siguientes pruebas a través de la consola:

hacer más tests

```
>>> %Run
  Introduce un numero: 0
  El cuadrado es: 0
>>> %Run
  Introduce un numero: -6
  El cuadrado es: 36
>>> %Run
  Introduce un numero: 1000000
  El cuadrado es: 1000000000000
```

Ejercicio 2. La instrucción de asignación (`=`) puede que sea la más importante, probablemente porque, en realidad, es la única instrucción que existe. Miraremos la ejecución de la siguiente instrucción de asignación:

```
  a = b + c
```

La ejecución supone lo siguiente:

1. el procesador obtiene de las variables `b` y `c` y sus contenidos;
2. el procesador realiza los cálculos, es decir, realización de la suma de los contenidos de `b` y `c`; y
3. el procesador transfiera el resultado del calculo a la memoria para almacenar lo calculado", es decir, el procesador introduce en la variable `a` el resultado de la operación.

Por lo tanto, a la hora de trabajar con la instrucción de asignación, hay que tener en cuenta lo siguiente:

1. El procesador leerá todas las variables que aparecen a la *derecha* del símbolo de asignación (`=`). Estas variables sólo serán leídas y deberán tener un valor válido, lo que significa que en algún momento se les habrá dado un valor, bien desde el propio programa o por consola.
2. A la derecha de la asignación puede aparecer una única constante (`a=2`), una única variable (`a=b`), o una expresión más compleja que involucre la utilización de varios operadores y variables.
3. A la izquierda de la asignación SÓLO aparecerá una variable, la variable que recibirá el resultado de la expresión de la derecha. Es obligatorio mantener este orden.

Ejercicio 3. Dadas dos variables `a` y `b`, realiza un programa en Python que permita al usuario introducir dos valores en las mismas, intercambie sus valores y los muestre por pantalla. La ejecución debe dar lugar a lo siguiente:

ejemplo de ejecución

```
>>> %Run
  Introduce el valor de la variable a: 4
  Introduce el valor de la variable b: 2
  El valor de a es 2
  El valor de b es 4
```

suponiendo que 4 y 2 son los valores introducidos por el usuario. Esto debe funcionar para cualquier par de valores introducidos.

Ejecuta pruebas a través de la consola y verifica la salida. ¿Tu programa funciona con números negativos? ¿Funciona con letras? ¿Funciona con números reales? ¿Las variables `a` y `b` pueden tener tipos diferentes? ¿Debería funcionar para todos estos casos?

Ejecuta más tests de tu programa para verificar su correcta funcionamiento.

Ejercicio 4. Realizar un programa en Python que recibe valores para tres variables `a`, `b` y `c`, intercambie entre sí sus valores del modo siguiente:

- `b` tome el valor de `a`,
- `c` tome el valor de `a`, y
- `a` tome el valor de `c`.

Esto hay que hacerlo SIN utilizar variables auxiliares, es decir una variable adicional de ayuda que no es `a`, `b` o `c` y utilizas para guardar uno de los valores.

Ejecute pruebas a través de la consola y verifique la salida. ¿Tu programa funciona con números negativos? ¿Funciona con letras? ¿Funciona con números reales? ¿Las variables `a`, `b` y `c` pueden tener tipos diferentes? ¿Debería su programa funcionar para todos estos casos?

Ejercicio 5. Las expresiones a la derecha de la asignación pueden ser todo lo complejas que deseemos. Implementar un programa que lea dos números reales, calcule e imprima su suma (`+`), resta (`-`), producto (`*`) y división (`/`).

ejemplo de ejecución

```
>>> %Run
  Introduce un numero real: 2.5
  Introduce otro numero real: 34.903
  La suma es: 37.403
  La resta es: -32.403
  El producto es: 87.2575
  La division es: 0.07162708076669627
```

Ejercicio 6. Implementar un programa que pida dos números al usuario, x e y y, a partir de éstos, calcule los siguientes resultados:

1. $x + y$
2. $(x + y)x$
3. $xx + yy$
4. x^{5y}
5. $3x^5 - 5x^3 + 2x - 7$
6. $yx^5 - (y + 1)x^3 + 5x - y$
7. x^{yy+2^y}

Ejercicio 7. Copiar y probar el siguiente programa:

```
a = int(input("Introduce un valor para la a = "))
a = a + 1
print("El valor de la variable a es ahora",a);
```

Como puede comprobarse, una misma variable puede aparecer tanto como operando como operador. Esta es una situación normal y muy recurrida en programación por lo que conviene acostumbrarse a ello. Sustituid ahora la instrucción *a = a + 1* por *a += 1*.

Ejercicio 8. Implementar un programa que calcule el salario bruto y neto de un empleado. El programa solicitará como datos: el numero de horas trabajadas (nh entero), el precio de la hora (ph float) y la retención aplicable en tanto por cien (r). El salario bruto (SB) y neto (SN) se calcula como:

1. $SB = nh * ph$
2. $SN = SB - (r/100 * SB)$

ejemplo de ejecución

```
>>> %Run
  Introduce el numero de horas trabajadas: 56
  Introduce el precio de la hora: 10
  Introduce la retencion aplicable en %: 25
  El salario bruto es:  560.0
  El salario neto es:  420.0
```

Para testear vuestro programa puedes probar con los siguientes casos de test:

test case ID	inputs			expected output	
	nh	*ph*	*r*	salario bruto	salario neto
1	56 horas	10 euro/hora	25 %	560 euros	420 euros
2	2.5 horas	20.4 euro/hora	25.6 %	51.0 euros	37.944 euros
3	1 hora	25 euros	0.1 %	25.0 euros	24.975 euros
4	125 hora	20 euros	0 %	2500.0 euros	2500.00 euros