```
# !/usr/bin/env python
# -*- coding: cp1252 -*-

'''
Pregunta 1
'''


# A)
'''
>>> Control1 = 5.5
>>> Control2 = 4.1
>>> Control3 = 6.4
>>> Promedio = control1 + control2 + control3/total
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'control1' is not defined

Acá nos arrojó un error de Nombre, es decir que una variable no fue definida previamente
intentó ser accedida. En este caso, hay que tener en cuenta que las mayúsculas son
distintivas en las variables, por lo que "control1" es distinto a "Control1".
Para arreglarlo, hay que cambiar las variables involucradas a minusculas, o la primera
letra mayuscula.

Luego, aparecerá el siguiente error:
NameError: name 'total' is not defined
debido a que la variable total no está definida en ningun lado. Para arreglarlo, podemos
definir la variable total antes de hacer el calculo final, o bien, eliminar esa referencia
y colocar directamente el valor en la división ( /3).

Finalmente, hay un error más grave aún y dificil de detectar a veces, y es que
Python respeta el orden de las operaciones, por lo que al no existir un paréntesis,
se asume que la división únicamente afectará al control3. Reescrita con
paréntesis vemos que ...

- Lo que queremos es: (control1 + control2 + control3) / 3.0
- Lo que en verdad hace es: control1 + control2 + (control3 / 3.0)

Por lo que para arreglarlo, tenemos que colocar los parentesis adecuados para realizar
bien la operación.
'''
```

```
# ------------------------------------------
# B)
'''
>>> di stancia = 20000.0                        # metros
>>> tiempo = 3000                               # segundos
>>> v = (distancia / 1000) / (tiempo / 3600)    # velocidad en km/h

El primer error, es que al escribir la primera linea, arroja el error:
SyntaxError: invalid syntax.
Esto debido a que la variable "di stancia" tiene un espacio entremedio,
lo cual no es la forma correcta de definir variables (estas no pueden contener espacios)
Para arreglarlo basta con borrar el espacio.



Otro error, y muy frecuente, es el que ocurre en este caso al calcular el resultado
final. Aparéntemente, la fórmula está correcta y de hecho, los valores numéricos
para la conversión están bien. El error radica en el uso de la división entera en Python.

Veamos que la distancia se definió como 20000.0 (Float), por lo que al dividir
por Int o Float, nos dará Float. Sin embargo, el tiempo fue definido como 3000 (Int),
por lo que al dividirlo por otro entero, Python realizará la división entera de dichos
números. En particular en este ejemplo, se cae en dicho caso, debido a que se hace la
división entre 3000 y 3600, lo cual da 0, y por consiguiente, será arrojado el error
de división por 0.

Una forma fácil de recordar dichos comportamientos, es pensar que Float domina por
sobre Int, por lo que basta que exista un Float en una operación como suma, resta,
multiplicación o división, para que el resultado sea Float.
De esta forma, es fácil ver que para que un resultado sea entero,
es que ambos operandos sean enteros.

Ahora respecto a como solucionar el problema: Existen varias alternativas, por ejemplo
definir la variable tiempo como 3000.0, o bien dividir tiempo por 3600.0 o bien
multiplicar tiempo por 1.0 antes de realizar la división.
Hay que tener cuidado con colocar lo siguiente: float(3000/3600),
ya que lo primero que se hará, es la división entera, que ya sabemos nos da 0,
y posteriormente, transforma dicho número a un Float, por lo que terminamos con un 0.0.
La moraleja entonces es asegurarse que ANTES de efectuar la división exista al menos
un Float en los operandos.
'''
```

```
# ------------------------------------------
# C)
'''
>>> mascota = 'perritos'
>>> cantidad = 10
>>> mensaje = Yo tenia + cantidad + mascota

Primero, arroja el siguiente error al intresar la tercera linea:
SyntaxError: invalid syntax

Esto, debido a que la primera parte del mensaje, Yo y tenia, es un mensaje que no fue
colocado entre comillas, por lo que python en vez de interpretarlo como texto, intenta
buscar variables con esos nombres (las cuales no existen), y ademas no hay un operador
entre ambas, por lo que arroja error de sintaxis, pues las variables se relacionan
entre si mediante operaciones. Para arreglarlo, hay que colocarle comillas a 'Yo tenia',
para que lo interprete como texto.

Luego, el segundo error que arroja es el siguiente:
TypeError: cannot concatenate 'str' and 'int' objects

Esto, debido a que no podemos concatenar directamente dos datos que sean
de tipos distintos (en este caso texto/string (la mascota) y numerico (la cantidad))
Para remediarlo, tenemos que convertir el numero a texto, con la función str()
>>> mensaje = 'Yo tenia' + str(cantidad) + mascota

Finalmente, una vez creado el mensaje, notamos que hay un pequeño error/detalle, ya
que todos los elementos del texto se encuentran juntos: 'Yo tenia10perritos', esto debido
a que si queremos agregar espacios u otros elementos, tenemos que especificarlo
explicitamente. Por lo que, para arreglarlo, haremos eso:
>>> mensaje = 'Yo tenia ' + str(cantidad) + ' ' + mascota
(el espacio vacio entre comillas, representa un espacio textual)
'''
# ------------------------------------------
# D)

'''
>>> terremoto = '1000'
>>> cantidad = '3'
>>> costo = terremoto * cantidad

Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'str'

Notemos que nos arroja error de la categoría "Error de tipo". Ésto significa que estamos
realizando una operación de elementos que no admiten los tipos entregados. En este caso,
estamos intentando multiplicar dos string, lo cual noesta definido, ya que la operación
de multiplicación espera argumentos de tipo numérico, o bien, un argumento de tipo numerico
y otro de texto.

Para arreglarlo, le podemos quitar las comillas a los numeros definidos en las variables
del principio, o bien, convertir a numero las variables terremoto y cantidad (con ayuda
de la función int()) durante el calculo de costo.
'''
```

```
# ---------------------------------------------
# E)

'''
>>> h=3
>>> def jalisco(h):
        h = h+1
        return h
>>> jalisco(5)       # cual es el resultado que entrega jalisco?
>>> h                # cual será el valor de la variable h definida al principio?

Primero notemos que la función jalisco, simplemente toma un numero h cualquiera, le suma 1,
y luego devuelve como resultado el numero h recibido, pero incrementado en 1.
Esta pregunta apunta directamente a lo que se conoce como "alcance" de una variable.
El concepto que se debe tener claro es que una función define sus propias variables,
y ellas nacen y viven únicamente dentro de dicha función. En nuestro caso, primero
definimos una variable h, que contendrá el valor 3.
Posteriormente se define una función que recibe como parámetro un valor,
que se almacenará en una variable también llamada h.
Pero aquí está lo importante, y es que ambas variables son distintas!
La variable h de la función jalisco, sólo vive dentro de su definición y por ende
es distinta a la variable h=3 definida fuera de la función.

Otra forma de verlo es que las funciones generan una burbuja totalmente independiente
del resto del programa (y por lo tanto de otras funciones también), en la que pueden
crear variables nuevas que viven únicamente dentro de dicha burbuja, y no les importa
si existe  alguna variable fuera de ella con el mismo nombre. En resumen, todas las
variables que se crean dentro de una función nacen, viven y mueren dentro de la función
(lo unico que se salva, por ahora, es el valor de retorno de la función)

Habiendo logrado entender lo anterior, veamos cuáles serán los resultados finales.

1) Primero se llama a jalisco con el valor 5, por lo que la variable h de la
   función será reemplazada por dicho valor, y no la que definimos antes como 3.
   (Recordemos la burbuja, la variable h de la función sólo actúa dentro de
    sus instrucciones de definición)

2) La función reescribe su variable h, como el valor que tiene mas uno, es decir 6,
   y finalmente retorna el h actualizado. Luego, lo que entrega jalisco, es 6.

3) Ahora bien cuando se pregunta por h, estamos fuera de la burbuja, por lo que el
   único h que conocemos, es el que definimos justo antes de definir la función.
   Luego, el valor final de dicho h sigue siendo 3, ya que no existe ninguna instrucción
   que redefina h fuera de la burbuja generada por jalisco.

'''
```

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# P2

# A)
# invertir2: int -> int
# Funcion que invierte el orden de los digitos de un numero entero de 2 digitos
# Ejemplo: invertir2(73) entrega 37
def invertir2(n):
    decenas = n/10      # aplicamos división entera sobre el numero
    unidades = n%10     # calculamos el resto de la división por 10
    return unidades * 10 + decenas

assert invertir2(32) == 23
assert invertir2(24) == 42
```

```python
# !/usr/bin/env python
# -*- coding: utf-8 -*-

# P3: Triangulos

# A)
# semiPerimetro : num num num -> num
# Funcion que calcula el semi perimetro dados los tres lados de un triangulo a,b y c
# Ejemplo: semiPerimetro(3,4,5) entrega 6, semiPerimetro(5,12,13) entrega 15
def semiPerimetro(a,b,c):
    return (a+b+c)/2.0

#test
assert semiPerimetro(3,4,5) == 6
assert semiPerimetro(5,12,13) == 15

#---------------------------
#B)
# Heron : num num num -> num
# Funcion que calcula el area de un triangulo de lados a, b y c
# Ejemplo: Heron(3,4,5) entrega 6, Heron(5,12,13) entrega 30
def Heron(a,b,c):
    S = semiPerimetro(a,b,c)
    area = (S*(S-a)*(S-b)*(S-c))**0.5

    # Otra formula de calcularlo, es con ayuda del modulo matematico de python:
    # area = math.sqrt(S*(S-a)*(S-b)*(S-c))
    # El cual cuenta con muchas funciones, que veremos en las proximas clases,
    # luego de que aprendamos a manejar modulos en python.
    return area

#Test
assert Heron(3,4,5) == 6
assert Heron(5,12,13) == 30
```

```python
# !/usr/bin/env python
# -*- coding: utf-8 -*-

# P4: Distancias

# A)
# millasAYardas : num -> num
# Funcion que convierte una distancia de Millas a Yardas
# Ejemplo: millasAYardas(1) entrega 1760
def millasAYardas(mi):
    return mi * 1760

#test
assert millasAYardas(1) == 1760
assert millasAYardas(2.5) == 4400

#---------------------------
# B)
# YardasAMetros : num -> num
# Funcion que convierte una distancia de Yardas a Metros
# Ejemplo: YardasAMetros(1) entrega 0.9144
def YardasAMetros(y):
    return y * 0.9144

#test
assert YardasAMetros(1) == 0.9144
assert YardasAMetros(4400) == 4023.36

#---------------------------
# C)
# millasAMetros : num -> num
# Funcion que convierte una distancia de Millas a Metros
# Ejemplo: millasAMetros(1) entrega 1609.344
def millasAMetros(mi):
    yardas = millasAYardas(mi)    # primero convertimos la distancia a yardas
    return YardasAMetros(yardas)  # y luego la convertimos de yardas a metros

    # Tambien es equivalente, por composición de funciones:
    # return YardasAMetros( millasAYardas(mi))

#test
assert millasAMetros(1) == 1609.344
assert millasAMetros(2.5) == 4023.36
```