Usando funciones en Python

Tal y como las calculadoras científicas, Python provee un número de funciones que podemos utilizar. A estas funciones se les llama **nativas**, *built-in* en inglés. En el siguiente ejemplo, se muestra el uso de dos funciones nativas: valor absoluto y potencia.

Ejemplo 1

```
>>> abs(2)
2
>>> abs(-2)
2
>>> pow(2, 3)
8
>>> pow(3, 2)
9
>>>
```

En las primeras líneas del ejemplo, la función `abs()` es aplicada al valor 2 y -2. Esto se logra indicando los valores en los paréntesis que acompañan al nombre de la función. Formalmente, a esos valores se les llama **parámetros actuales**, o argumentos, de la función y es sobre estos parámetros actuales que la función se aplica. Notemos que la función `pow()` requiere dos parámetros: la base y el exponente. Para que funcione, debemos indicar ambos argumentos separados con una coma.

En la clase anterior conocimos que para Python los números pueden ser de tres tipos, enteros **(int)**, enteros largos **(long)**, y números no enteros **(float)**. Sin embargo, a través de funciones podemos cambiar fácilmente el tipo de número con el que Python trabaja, utilizando las funciones presentadas en la siguiente tabla:

Nombre	Descripción
`int(x)`	Recibe un número y lo convierte a un entero, si el número no puede ser contenido en 4 bits, se guarda como `long`, en caso de recibir un `float`, el resultado se trunca, no se aproxima.
`long(x)`	Recibe un número y lo convierte a un entero largo, (sin importar que pueda ser representado como `int`), en caso de recibir un `float`, el resultado se trunca, no se aproxima.
`float(x)`	Recibe un número y lo convierte a un flotante, en caso de recibir un `float`, el resultado se trunca, no se aproxima.

Pregunta 1

Ahora, desarrollemos la primera pregunta de la actividad.

Podemos extender la funcionalidad de Python agregándole **funciones importadas**. Estas no vienen nativamente con el intérprete, sino que se encuentran en bibliotecas para que sean consultadas por él. A estas bibliotecas de Python se les conoce como **módulos**. Uno de los módulos que nos será de utilidad es `math`, que contiene muchas de las funciones matemáticas y trigonométricas que usamos. La siguiente tabla lista algunas de las funciones que podemos destacar:

Nombre	Descripción
`sin(x)`	Seno de x, con x expresado en radianes
`cos(x)`	Coseno de x, con x expresado en radianes
`tan(x)`	Tangente de x, con x expresado en radianes
`exp(x)`	Número *e* elevado a x
`log(x)`	Logaritmo natural (base *e*) de x
`log10(x)`	Logaritmo en base decimal de x
`sqrt(x)`	Raíz cuadrada de x
`degrees(x)`	Convierte a grados un ángulo x expresado en radianes
`radians(x)`	Convierte a radianes un ángulo x expresado en grados

Probemos en el intérprete de Python la función `sin()`.

Ejemplo 2

```
>>> sin(0)

Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    sin(0)
NameError: name 'sin' is not defined
>>>
```

Podemos darnos cuenta que ocurrió un error. Esto fue debido a que no hemos indicado al intérprete que consulte el módulo `math`. Para esto debemos usar la sentencia `import` y en nombre del módulo.

Ejemplo 3

```
>>> import math
>>> math.sin(0)
0.0
>>> sin(0)

Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    sin(0)
NameError: name 'sin' is not defined
>>>
```

Podemos notar que a pesar de importar el módulo `math` ocurrió un error al invocar la función como lo hubiéramos hecho con las funciones nativas. Pero el error no ocurrió cuando especificamos explícitamente el nombre del módulo (separado con un punto del nombre de la función). Esto podemos corregirlo.

Ejemplo 4

```
>>> from math import sin, cos
>>> cos(0)
1.0
>>> sin(0)
0.0
>>>
```

Con lo anterior, pudimos importar las funciones seno y coseno, lo que nos permite usarlas escribiendo sólo su nombre y los parámetros actuales correspondientes.

Pregunta 2

Con esta información estamos en condiciones de desarrollar la segunda pregunta de la actividad.

Sabemos que cada día los computadores se usan para más cosas. Obviamente, es imposible que el lenguaje de programación contenga, nativamente o por medio de módulos, todas las funciones que podamos necesitar. Por esta razón, Python nos da la opción de definir nuestras propias funciones. Por ejemplo, supongamos que necesitamos calcular el valor de:

$$2^{2(x-1)^2} + 3^{2(x-1)^2} + 4^{2(x-1)^2} \quad (1)$$

Cuando x vale 1,2 y 3. Podríamos escribir las expresiones correspondientes en Python:

Ejemplo 5

```
>>> 2 ** (2 * (1 - 1) ** 2) +3 ** (2 * (1 - 1) ** 2) + 4 ** (2 * (1
- 1) ** 2)
3
>>> 2 ** (2 * (2 - 1) ** 2) +3 ** (2 * (2 - 1) ** 2) + 4 ** (2 * (2
- 1) ** 2)
29
>>> 2 ** (2 * (3 - 1) ** 2) +3 ** (2 * (3 - 1) ** 2) + 4 ** (2 * (3
- 1) ** 2)
72353
>>>
```

Nos sale bastante largo. Pero podemos notar que en cada expresión se repite `2 * (1 - 1) ** 2`, `2 * (2 - 1) ** 2` y `2 * (1 - 1) ** 2`, varias veces. ¿Podríamos evitar escribir tantas veces la misma sub-expresión? ¡Sí! Para esto, miremos la fórmula 1. Es obvio que el exponente es el mismo en cada término, y que el valor depende de una *variable matemática* x. Podemos **encapsular** el cálculo de este exponente como una función de una *variable en memoria* x:

Ejemplo 6

```
>>> def calculaExponente(x):
      valorExponente = 2 * (x - 1) ** 2
      return valorExponente

>>> calculaExponente(1)
0
>>> calculaExponente(2)
2
>>> calculaExponente(3)
8
>>> 2 ** calculaExponente(1) + 3 ** calculaExponente(1) + 4 **
calculaExponente(1)
3
>>> 2 ** calculaExponente(2) + 3 ** calculaExponente(2) + 4 **
calculaExponente(2)
29
>>> 2 ** calculaExponente(3) + 3 ** calculaExponente(3) + 4 **
calculaExponente(3)
72353
>>>
```

Podemos ver que al definir nuestra propia función `calculaExponente()`, podemos utilizarla como una función más de Python. El nombre escogido para esta función sigue la **convención** entre programadores: el nombre debe ser **indicativo** de lo que hace la función y debe partir con un **verbo**; si el nombre contiene más de una palabra, éstas deben escribirse con la **primera letra en mayúsculas**.

Fijémonos ahora en la sintaxis que utilizamos para declarar la función. Ésta se compone de dos partes: el **encabezado** de la función y el **cuerpo** de la función.

El encabezado parte con la **palabra reservada** `def`, seguida del **nombre de la función**, que debe ser un identificador válido según las reglas de Python. Este es el nombre con el que nos referiremos a la función en el resto de la sesión con el intérprete de Python. Luego, entre paréntesis, indicamos los **parámetros formales** de la función. Estos son los nombres que se utilizan al **interior** de la función para referirse a los argumentos que se le entregan al **invocarla**. El encabezado termina con un signo **dos puntos**, indicando que ahora se comienza el cuerpo de la función.

El cuerpo de la función corresponde al **bloque de sentencias** que deben ejecutarse al evaluar la función. En este caso, el bloque se compone de dos sentencias. La primera calcula el valor del exponente necesitado, utilizando una expresión que incluye el parámetro formal de la función, y lo asigna a una variable denominada `valorExponente`. La segunda, utiliza la sentencia `return` para que la función devuelva el valor de esta variable como resultado de su evaluación.

Notemos que el cuerpo de la función se escribe **indentado**[1] respecto al encabezado de la función, es decir tienen una sangría mayor desde el margen izquierdo. Es importante que recordemos que todas las sentencias de un mismo bloque tienen la misma indentación.

Es más, para marcar *sintácticamente* el fin de la declaración de una función, se deja una **línea en blanco sin indentar**.

Pregunta 3

Intentemos ahora con la pregunta 3 de la actividad.

[1] Anglicismo de uso común por gente que programa

Veamos algo más relacionado con funciones. Consideremos el siguiente código:

Ejemplo 7

```
>>> divisor = 4
>>>
>>> def formula2(base):
      divisor = 4 * base
      potencia = 4 ** base
      return potencia/divisor

>>> def formula3(base):
      potencia = 2 ** (base + 1)
      return potencia/divisor

>>> resultado1 = formula2(4)
>>> resultado2 = formula3(4)
>>> print divisor, resultado1, resultado2
```

Si nos fijamos, la variable `divisor` recibe un valor en la primera sentencia. Luego aparece recibiendo un valor en el cuerpo de la función `formula2`, y siendo usada en la línea siguiente. Finalmente es usada en el cuerpo de la `formula3`. ¿Qué valores desplegará la última sentencia?

Para responder esta pregunta, primero debemos saber que la variable `divisor` que aparece dentro del cuerpo de la función `formula2` es una **variable local**, esto significa que está **definida sólo para ese bloque**, incluso no se crea hasta que la función es invocada[2] y se elimina cuando la ejecución de la función termina. Es esta misma variable local la que es utilizada en la expresión de la sentencia de retorno.

En segundo lugar, debemos saber que la variable `divisor` que aparece en la primera línea del ejemplo 6 declara una **variable global**, lo que significa que está **definida para todas las sentencias** que siguen, **excepto** para bloques que tengan una variable local con el mismo nombre. Así, es esta variable la que se usa en la expresión de la sentencia de retorno de la función `formula3` y en la sentencia `print` de la última sentencia.

Así, el primer valor que se despliega en pantalla es un 4, asignado a la variable global `divisor`. Al ejecutar `formula2` con parámetro actual 4, la variable local `divisor` toma el valor 16, la variable local `potencia` toma el valor 256, por lo que el segundo valor desplegado es 16. Al ejecutar `formula3` con parámetro actual 4, la variable local `potencia` toma el valor 32, el que divide por el valor de la variable global `divisor`, por lo que el tercer valor desplegado es 8.

[2] Anglicismo de uso común por gente que programa

Importante

Usar variables globales en el cuerpo de una función es una **mala práctica** de programación. Lo mejor es encapsular el comportamiento completo de una función que utilice sólo parámetros formales y variables locales en su cuerpo.

Sólo usaremos variables globales para almacenar **valores constantes** que no requerirán ser cambiados en el resto del programa, como por ejemplo el valor del número π.

Pregunta 4 y 5

Con ésta información, intentemos resolver las últimas dos preguntas de la actividad de hoy