Funciones

Introducción a la Programación

Hoy veremos:

1. Cómo usar funciones
2. Cómo definir nuevas funciones
3. Funciones con parámetros
4. Funciones que devuelven cosas

Funciones matemáticas

Seguramente hayan visto en matemática funciones como por ejemplo: *raiz cuadrada* o *logaritmo en base 2*.

Seguramente también hayan aprendido a evaluar expresiones como $\sqrt{\frac{\pi}{2}}$ y $\log_2(40 - 8)$

Funciones matemáticas

$\sqrt{\frac{\pi}{2}}$ $\log_2(40-8)$

Para evaluarlas, primero hay que evaluar lo que está entre paréntesis, llamado **argumento**

$\frac{\pi}{2}$ es aproximadamente $1,571$
$40-8$ es 32
Una vez evaluado esto, podemos evaluar la función misma:

la raiz cuadrada de $1,571$ es aproximadamente 1,2533
el logaritmo en base 2 de 32 es 5

Este proceso puede ser aplicado repetidas veces para evaluar expresiones más complicadas como $\log(1/\sqrt{\pi/2})$.

Primero evaluamos el argumento de la función de más adentro, después evaluamos la función, y así seguimos.

Funciones preincorporadas de Python

Python trae funciones preincorporadas para calcular casi todas las funciones matemáticas

```
raiz = math.sqrt(17.0)
angulo = 1.5
altura = math.sin(angulo)
```

Composición

Tal como con las funciones matemáticas, las funciones en Python pueden ser **compuestas**, lo cual significa que es posible usar una expresión como parte de otra.

Por ejemplo, es posible usar cualquier **expresión** como argumento de una función:

$x = cos(angulo + \frac{\pi}{2})$
(en matemática)

```
x = math.cos(angulo + math.pi/2)
```

(en Python)

También podemos tomar el resultado de una función y pasarlo como argumento de otro:

```
x = math.sqrt(abs(-5))
```

Nosotros ya usamos algunas funciones. ¿Cuáles?

Ejercicio

Escribir un programa en Python (en la computadora) que tome un número decimal x y muestre por pantalla el resultado del siguiente cálculo:

$$\sqrt{\log(|1-x|)}$$

Recuerden que para usar las funciones matemáticas de Python, hay que incluir arriba de todo:

```
import math
```

Nuestras Propias Funciones

Hasta ahora hemos visto cómo usar funciones. En computación (y nosotros también de ahora en adelante) diremos **llamar** una función cada vez que usamos una función.

Nuestras Propias Funciones

En Python, si queremos definir nuestra propia función, lo hacemos de esta forma:

```
def NOMBRE( LISTA DE PARAMETROS ):
    SENTENCIAS
```

Podemos inventar (casi) cualquier nombre que queramos para nuestra función, al igual que lo hacemos con las variables. La lista de parámetros especifica qué información hay que proveer, si es que la hay, para poder usar (o **llamar**) la nueva función.

Nuestras Propias Funciones

Las escribimos de esta forma:

```
def NOMBRE( LISTA DE PARAMETROS ):
    SENTENCIAS
```

Se puede incluir cualquier número de sentencias dentro de la función, pero todas deben escribirse con sangría a partir del margen izquierdo. Al igual que un *for* o un *while*.

Las primeras dos funciones que vamos a escribir no tienen parámetros, por lo que la sintaxis se ve así:

```
def mostrarGuion():
    print("-", end="")
```

Esta función se llama `mostrarGuion`, y los paréntesis vacíos indican que no toma parámetros. Contiene solamente una única sentencia, que imprime una cadena formada por un guión medio.

En `otro archivo` podemos llamar a esta nueva función usando una sintaxis similar a la forma en que llamamos a los comandos preincorporados de Python:

```
print("Lista de compras:")
mostrarGuion()
print("Queso")

mostrarGuion()
print("Leche")

mostrarGuion()
print("Huevos")
```

Si corremos ese programa, obtendremos en la consola:

```
Lista de compras:
-Queso
-Leche
-Huevos
```

¿Qué tal si quisiéramos mostrar una línea de 6 guiones en lugar de uno solo?

Podríamos llamar a la misma función repetidamente:

```
mostrarGuion()
mostrarGuion()
mostrarGuion()
mostrarGuion()
mostrarGuion()
mostrarGuion()
```

O podríamos escribir una nueva función, llamada `mostrarLinea`, que imprima 6 guiones:

```
def mostrarLinea():
    mostrarGuion()
    mostrarGuion()
    mostrarGuion()
    mostrarGuion()
    mostrarGuion()
    mostrarGuion()
    print()
```

O podríamos escribir una nueva función, llamada `mostrarLinea`, que imprima 6 guiones usando un ciclo:

```
def mostrarLinea():
    for i in range(6):
        mostrarGuion()
    print()
```

Nótese que...

- Se puede llamar al mismo procedimiento repetidamente. De hecho, es muy sutil hacer eso.
- Se puede hacer que una función que llame a otra función. En este caso, `mostrarLinea` llama a `mostrarGuion` y a `print` Otra vez, esto es común y muy útil.

¿Para qué sirve todo esto?

Muchas cosas, en este ejemplo vemos dos:

1. Crear una nueva función brinda una oportunidad de dar un nombre a un grupo de sentencias. Las funciones pueden simplificar un programa al esconder un cómputo complejo detrás de un comando simple, y usando una frase en castellano en lugar de código complicado.
2. Crear una nueva función puede hacer un programa más corto al eliminar código repetitivo. Por ejemplo, ¿cómo harías para imprimir 3 líneas nuevas consecutivas? Llamando a `mostrarLinea` tres veces.

Cuando hagan sus propias funciones, guardenlas en un archivo con extensión `.py`. Por ejemplo: `misFunciones.py`

Cuando las quieran usar en otro archivo, escriban arriba de todo:

```
from misFunciones import *
```

sin la extensión (.py).

Nota: Se pueden guardar muchas definiciones de funciones en un mismo archivo.

Parámetros

Algunas de las funciones preincorporadas que hemos usado tienen **parámetros**, que son valores que se le proveen para que puedan hacer su trabajo. Por ejemplo, si queremos encontrar el seno de un número, tenemos que indicar de qué número. Por ello, `sin` toma un valor como parámetro. Para imprimir una cadena, hay que proveer la cadena, y es por eso que `print` toma una cadena como parámetro.

Algunas funciones toman más de un parámetro, como `math.pow`, la cuál toma dos numeros, la base y el exponente, y devuelve el resultado de elevar la base a la potencia indicada por el exponente.

Parámetros

Cuando definamos nuestras propias funciones, la lista de parámetros indica cuántos parámetros utiliza. Por ejemplo:

```
def imprimirDosVeces(unaCadena):
    print(unaCadena)
    print(unaCadena)
```

Esta función toma un sólo parámetro, llamado `unaCadena`. Cualquiera sea ese parámetro (y en este punto no tenemos idea cuál es), es impreso en pantalla dos veces.

Parámetros

Para llamar esta función, tenemos que proveer una cadena. Por ejemplo, podríamos tener un programa como este:

```
imprimirDosVeces("No me hagas decirlo dos veces!")
```

La cadena que proporcionamos se denomina **argumento**, y decimos que el argumento es **pasado** a la función.

Alternativamente, si tuviéramos una cadena almacenada en una variable, podríamos usarla como un argumento en vez de lo anterior:

```
argumento = "Nunca digas nunca."
imprimirDosVeces (argumento)
```

Importante

El nombre de la variable que pasamos como argumento no tiene nada que ver con el nombre del parámetro.

Pueden ser el mismo o pueden ser diferentes, pero es importante darse cuenta que no son la misma cosa, simplemente sucede que tienen el mismo valor (en este caso la cadena ``Nunca digas nunca.'').

Valores de retorno

Las funciones preincorporadas que usamos, como abs y pow han reducido en valores. Llamar a cada una de estas funciones genera un nuevo valor, que usualmente asignamos a una variable o usamos como parte de una expresión.

```
el_mayor = max(3, 7, 2, 5)
x = abs(3 - 11) + 10
```

Pero hasta ahora ninguna de las fuciones que hemos escrito ha devuelto un valor. Veamos cómo hacer eso.

Valores de retorno

Veamos a la función area como ejemplo, esta función devuelve el area de un círculo de un cierto radio dado.

```
def area(radio):
    x = math.pi * radio * radio
    return(x)
```

La sentencia `return` toma un valor de retorno. Esta sentencia significa: "Salí inmediatemente de esta función, volvé al lugar donde te habían llamado y usá la siguiente expresión como un valor de retorno".

Valores de retorno

O bien:

```
def area(radio):
    return (math.pi * radio * radio)
```

Funciones puras

Una función en computación se considera *pura* si **siempre** que se la llame con parámetros adecuados retorna algún valor y nunca tiene efectos colaterales (como imprimir cosas en la pantalla).
En IP nos gustan las funciones puras.
Y además, a partir de ahora, IP es puras funciones.

Preguntas

¿?