Universidad Nacional de General Sarmiento

Introducción a la Programación

Funciones

Funciones matemáticas

Seguramente hayan visto en matemática funciones como por ejemplo:

raiz cuadrada o logaritmo en base 2.

O también hayan aprendido a evaluar expresiones como

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

Funciones matemáticas

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

$\sqrt{\frac{\pi}{2}}$ ⟶ $\frac{\pi}{2}$ es aproximadamente **1,571** ⟶ $\sqrt{1{,}571}$

Entonces la raíz cuadrada de 1,571 es aproximadamente **1,2533**

$\log_2$ (40 - 8) ⟶ primero resuelvo (40-8) = **32**

$\log_2$ (32) = 5 o sea el logaritmo en base 2 de 32 es **5**

.

Funciones matemáticas

Este proceso puede ser aplicado repetidas veces para evaluar expresiones más complicadas como

$$\log\left(1 \,/\, \sqrt{\pi/2}\right)$$

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

Funciones 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 )
```

.

Funciones de Python

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:

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

En Python $\longrightarrow$ x = math.cos (angulo + math.pi /2)

Funciones de Python

Composición:

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

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

Nosotros ya usamos algunas funciones. ¿Cuales?

Funciones de Python

Ejercicio:

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

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

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

import math

Funciones de Python

Hasta ahora hemos visto como usar funciones.

En computación (y nosotros también de ahora en adelante) diremos **llamar** a una función cada vez que usamos una función.

Veremos como se definen las funciones en Python

Como Definir Nuevas Funciones

Así definimos nuestras funciones en Python:

```
def nombreFcion(lista de parámetros):
    sentencias
```

La primera instrucción de una función se inicia con el termino **def**, que declara que acá comienza la definición de una función.

Como Definir Nuevas Funciones

Así definimos nuestras funciones en Python:

```
def nombreFcion(lista de parámetros):
    sentencias
```

A continuación va el nombre que le daremos a la función.

Podemos inventar (casi) cualquier nombre que queramos para nuestra función, al igual que lo hacemos con las variables.

Como Definir Nuevas Funciones

Así definimos nuestras funciones en Python:

```
def nombreFcion(lista de parámetros):
    sentencias
```

La lista de parámetros va encerrada entre () y especifica que información hay que proveer, si es que la hay, para poder usar (o **llamar**) a la nueva función.

La instrucción finaliza con :

Como Definir Nuevas Funciones

Así definimos nuestras funciones en Python :

```
def nombreFcion(lista de parámetros):
    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*.

Como Definir Nuevas Funciones

Vamos a definir funciones que no tengan 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.

Como Definir Nuevas Funciones

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" )
```

Como Definir Nuevas Funciones

Si corremos ese programa, obtendremos en la consola:

Lista de compras :
- Queso
- Leche
- Huevos

Como Definir Nuevas Funciones

¿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 ()
```

Como Definir Nuevas Funciones

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

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

Como Definir Nuevas Funciones

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

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

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

Como Definir Nuevas Funciones

Que podemos observar:

- ✓ Se puede llamar al mismo procedimiento repetidamente.

  De hecho, es muy sutil hacer eso.

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

- ✓ Se puede hacer 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.

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

Como Definir Nuevas Funciones

Ventajas que brinda el uso de funciones:

1. Creando una función le damos un nombre a un grupo de sentencias, que podrían contener un cómputo complejo, y lo hacemos con un comando simple, usando una **frase en castellano** en lugar de código complicado

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

Como Definir Nuevas Funciones

Ventajas que brinda el uso de funciones:

2. Una 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

```
for i in range(3):
    mostrarLinea()
```

Como Definir Nuevas Funciones

Cuando hagan sus propias funciones, guárdenlas 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 extension (.py)

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

Funciones con 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. **sin(numero)**

Para imprimir una cadena, hay que proveer la cadena, y es por eso que print toma una cadena como parámetro. **print ("datos", valor1)**

Algunas funciones toman más de un parámetro, como math.pow, la cual toma dos números, la base y el exponente, y devuelve el resultado de elevar la base a la potencia indicada por el exponente. **math.pow(b,e)**

Funciones con 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.

Funciones con 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.

Funciones con parámetros

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 )
```

Funciones con parámetros

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

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

La función

El programa que llama a la función

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.'').

Funciones con 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 funciones que hemos escrito ha devuelto un valor. Veamos como hacer eso.

Funciones con valores de retorno

Veamos a la función area como ejemplo, esta función devuelve el área 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 inmediatamente de esta función, volvé al lugar donde te habían llamado y usá la siguiente expresión como un **valor de retorno**".

Funciones con valores de retorno

O bien.

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

El valor de retorno puede ser el resultado de un cálculo.
Python reduce la expresión a un valor, y es ese valor que retorna al programa que lo llamó.

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**.

Funciones

Preguntas