VIDEOJUEGOS
INTRODUCCIÓN A PYTHON

CLASE 4

Contenido

“

Este espacio curricular permite fortalecer el espíritu crítico y la actitud creativa del futuro graduado. Lo alienta a integrar los conocimientos previos, recreando la práctica en el campo real, al desarrollar un proyecto propio.

”

PRORAMACIÓN I

Uno de los primeros avances que se tiene en programación es cuando comenzamos a trabajar con funciones, el cambio de nuestro código es enorme. Hace un uso correcto de las funciones, nos permite desarrollar códigos mas legibles y mas mantenibles.

En este apartado, vamos a trabajar con Funciones y veremos las ventajas de las buenas practicas en este lenguaje de programación.

Introducción

Una función es una unidad de código, que se utiliza para una determinada tarea. Para cumplir con ello, una función permite en algunas oportunidades ingresar alguna información y, no siempre, devuelve algún dato. El concepto es muy similar a las funciones matemáticas que en algún momento usamos de la librearía math.

El provecho de las funciones se puede mostrar si primero consideramos un código, que antiguamente se llamaba "código espagueti", el uso de funciones muestra a las claras que el código se vuelve mas mantenible y mas legible-

Funciones definidas por el usuario

Como desarrolladores podemos usar las funciones que el lenguaje ofrece, o bien, desarrollar nuestras propias funciones. Cuando creamos una función debemos respetar los siguientes pasos:

1- Comenzar con def
2- Agregar el nombre de la función, utilizando el sistema camel case
3- Completar con la lista de parámetros o con () si no tiene parámetros de entrada y terminar con ":"
4- Escribir el cuerpo/bloque de la función
5- Usar la palabra reservada return para el valor de retorno si lo tuviese.

Sintaxis de la función

```
def [nombreDeLaFuncion]([lista de paranetros]):
  #codigo
  return valor_de_retorno
```

```
#funcion sin parametros
def saludarAlMundo():
    print ("Hola Mundo")
```

Algunos temas para considerar para comprender cada parte de la función:

a- Se conoce como firma de la función a la hilera de código donde se encuentra el def

   def [nombreDeLaFuncion]([lista de parámetros]):

b- La función puedo no tener parámetros
c- La función puede no retornar valores
d- El bloque de código, es decir el cuerpo de la función define el ámbito de la función.

Funciones con un parámetro

No todas las funciones reciben parámetros, pero otras pueden recibir más de uno. Los parameros se muestran en la firma de la función separados por coma.

Declaración de la función

```
#funcion con parametros
def saludar(param_name, param_saludo
    print(f"{param_saludo} {param_name}")
```

Llamada de la función

```
saludar("pablo","Buenos días")
```

En la llamada de la función podemos individualizar los valores "pablo" y "Buenos días". Estos valores reciben el nombre de argumentos. Pueden ser valores concretos o también variables.

Por su parte un parámetro no es un valor en si mismo, sino un contenedor de posición que representa el orden de argumentos (valores) que recibe de la llamada.

Funciones con varios parámetros

Las funciones no se limitan a tener un o dos parámetros, sino que pueden tener varios. LA relación entre el argumento y el parámetro es el orden de secuencia que tiene en la lista. Es decir, el primer argumento con el primero parámetro, etc.

```
#funcion con parametros
def saludar(param_name, param_saludo):
    print(f"{param_saludo} {param_name}")

saludar("pablo","Buenos días")
```

Pero, si por alguna loca razón necesito alterar el orden en la lista. Para no perder la asociación puede armar las parejas de parámetros y argumentos al momento de hacer la llamada.

```
saludar(param_saludo="Buenos días",param_name="pablo")
```

Ámbito Variable

Nuestros programas se dividen en partes de código, como dijimos antes, el bloque de una función es una parte de un código. El código que se encuentra tabulado y que representa el cuerpo de la función define un ámbito. Las variables declaradas dentro de este ámbito no existen fuera de él. Estas variables dentro de la función se conocen como variables locales.

```
def sumar(a,b):
    sumita=a+b #variable local
    return sumita


print (sumita)
```

Esta variable no existe fuera de la función

Lo que obtenemos en la consola

NameError: name 'sumita' is not defined

El alcance de la variable se refiere a las variables que un bloque de código puede ver o conocer. Una variable definida dentro de una función particular se dice que es una variable local.

Es importante mencionar, que las variables locales se destruyen al salir de la función.

Funciones de la biblioteca matemática

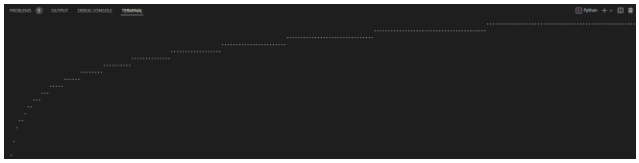
Python nos brinda una librería llamada math, es una extensión de funciones matemáticas. Esto incluye funciones trigonométricas, logarítmicas y exponenciales. También aporta funciones para el cálculo de raíz cuadrada, y valores absolutos. Para utilizar esta librería debemos importarla primero al comienzo del código fuente.

```
import math
```

Un ejemplo del uso de estas funciones se puede ver en el siguiente código:

```
import os, math
os.system("cls")
salir=False
punto_a_x=0
punto_a_y=0
contador=0
base_altura=20
while contador <=250:
        print("\033[" + str(base_altura -
punto_a_y ) + ";" + str(punto_a_x ) + "H*" )
        print("\033[" + str(base_altura -
punto_a_y ) + ";" + str(punto_a_x ) + "H." )
        contador+=1
        punto_a_x +=1
        punto_a_y =int(math.log(punto_a_x) *
3.7)
```

el resultado en consola es:



Funciones de números aleatorios

Python no facilita un juego de funciones para trabajar con número aleatorios. Esta librería se llama random, y la incluimos importando de la siguiente manera:

```
import random
```

La función random genera un número partiendo de una semilla (seed) que se calcula desde una ecuación exponencial. Los valores forman parte de una secuencia, por lo que los valores generados son siempre los mismos. Es necesario la pericia del programador para hacer de esa función un verdadero valor aleatorio.

random.randrange

Devuelve un valor aleatorio seleccionado al azar en un rango conformado por inicio y parada. Por ejemplo:

```
rnd=random.randrange(0,5)
print (rnd)
```

random.choice

devuelve un valor al azar de una lista de elementos

```
import random
palabras=["CASA","TELEFONO","CIELO","PAJARO"]
rnd=random.choice(palabras)
print (rnd)
```

random.shuffle

Mezcla los valores de una lista

```
import random
palabras=["CASA","TELEFONO","CIELO","PAJARO"]
rnd=random.shuffle(palabras)
print (palabras)
```

Resultado en consola

```
PS D:\salaPython> & C:/Python310/python.exe d:/salaPython/function.py
['TELEFONO', 'PAJARO', 'CIELO', 'CASA']
PS D:\salaPython> & C:/Python310/python.exe d:/salaPython/function.py
['CASA', 'CIELO', 'TELEFONO', 'PAJARO']
PS D:\salaPython> & C:/Python310/python.exe d:/salaPython/function.py
['TELEFONO', 'PAJARO', 'CASA', 'CIELO']
PS D:\salaPython> & C:/Python310/python.exe d:/salaPython/function.py
['CASA', 'PAJARO', 'CIELO', 'TELEFONO']
PS D:\salaPython>
```

random.seed

random.**seed**(a = None , versión = 2)

Inicialice el generador de números aleatorios.

Si se omite a=None o, se utiliza la hora actual del sistema. Si a es un int, se usa directamente.

Con la versión 2 (la predeterminada), un objeto str, bytes o bytearray se convierte en un int y se utilizan todos sus bits.

Con la versión 1 (proporcionada para reproducir secuencias aleatorias de versiones anteriores de Python), el algoritmo str genera bytes una gama más estrecha de semillas.

Para más información ver https://docs.python.org/3/library/random.html

Parámetros predeterminados

Se puede definir un parámetro opcional, en la firma de una función. Este siempre debe ser el último, y para declararlo como opcional solo se le asigna un valor

```
def sumar(a,b=5):
    return a + b
```

llamada de la función sumar con valor opcional

```
print(sumar(4))
```

Sobrecarga de la firma de una función

La sobrecarga en Python no es posible, a diferencia de otros lenguajes, podemos escribir dos funciones con el mismo nombre, pero siempre prevalece la segunda que fue declarada

```
def sumar(a,b=5):
    return a + b

def sumar(a,b=1):
    return a + b
print(sumar(4))
```

consola

```
PS D:\salaPython> & C:/Python310/python.exe d:/salaPython/function.py
5
PS D:\salaPython>
```

Para aprovechar la sobrecarga como en otros lenguajes, se debe importa la librería multipledispatch. Par ello, abrimos una terminal de consola y ejecutamos el instalador de python

pip3 install multipledispatch

Esta librería me ofrece un decorado llamado dispatch, que me permite definir la firma de la sobrecarga. Para que esto tenga éxito, la sobrecarga entre las funciones deben diferir en cantidad de parámetros o tipos de datos de los parámetros.

```
from multipledispatch import dispatch

@dispatch(str,str)
def sumar(a,b):
    return int(a) + int(b)

@dispatch(int,int)
def sumar(a,b):
    return a + b

@dispatch(int,int,int)
def sumar(a,c,b):
    return a + b + c

print(sumar("4","1"))
```

Bibliografía

Sweigart, A. (2012). Making Games with Python & Pygame: A Guide to Programming with Graphics, Animation, and Sound. Recuperado de https://inventwithpython.com/makinggames.pdf

Pygame. (s.f.). [Página oficial de pygame]. Recuperado 2 julio, 2019, de https://www.pygame.org/news