python 3

Variables

Una variable es un espacio de memoria dónde podemos almacenar un dato, acceder a dicho dato y borrar el dato, tantas veces como queramos. La variable queda definida con tres parámetros:

- Nombre de la variable o del espacio de memoria
- Tipo de dato que se va a almacenar en dicha variable: int (enteros), float (reales), char (caracteres), String (texto), bool (True/False) ...
- Valor inicial a almacenar en la variable

Inicializar una variable

- `dato = 9`
- `dato= 6.8`
- `dato = ‘Antonio’`
- `dato = True`
- `dato = input("What's your name?")`
- `dato = [‘Antonio’ , ‘Pedro’ , ‘Juan’]`

La función input ( ), espera a que le pasemos un dato por pantalla y lo convierte a tipo string. En nuestro ejemplo, la variable name, almacena un string. Muchas veces necesitaremos que input nos devuelva un entero, por lo que tendremos que convertir el dato almacenado en la variable a entero con la función `int(name)`. Para visualizar el valor que ha tomado la variable hacemos imprimir en pantalla.

```
print("Hello" , name , "cómo vas?")
```

Funciones

Las funciones realizan una tarea concreta, pueden tener o no argumentos. Existen unas funciones predefinidas. Ejemplos de llamadas a función:

- int(argumento) → convierte un dato a entero
- float(argumento) → convierte un dato a decimal o real
- bool(argumento) → convierte un dato en binario, es decir , solo puede adoptar dos estados (verdadero/falso)
- print(argumento) → imprime en pantalla el valor de una variable o un literal entre comillas.

Podemos definir nuestras propias funciones, en el ejemplo una función que admite dos argumentos:

```
def sumar( sumando1 , sumando2):
        print(sumando1 + sumando2)
```

```
def sumar( sumando1 , sumando2):
        resultado= sumando1 + sumando2
        return  resultado
```

Notar que todos los comandos que pertenecen a una misma función están tabulados. Para llamar dicha función basta con escribir su nombre con sus argumentos:

```
sumar( 5 , 7 )
```

1. Crea una calculadora sumadora: imprime en pantalla numero1+numero2.
2. Ejemplo: Realizar un cajero automático con saludo inicial y que calcule el saldo resultante al retirar dinero con saldo inicial de 1000€
3. Es una calculadora de área y perímetro del rectángulo. De forma educada saluda, pide el lado y luego el alto. Como resultado da el área y el perímetro de dicho rectángulo. Haz el programa usando funciones propias.
4. Calculadora de ecuaciones de segundo orden. Nota: sube nota si calcula soluciones complejas.

Listas

Es una variable donde puedo almacenar una serie de elementos. Cada elemento queda referenciado por la posición que cupa en dicha lista empezando por el cero: lista[0], lista[1], lista[2], lista[3], lista[4], lista[5] . . .

- Crear una lista llama transportes: `transportes = [‘coche’, ‘camión’,’avión’,’tren’]`
- Acceder al primer elemento de la lista: `el_primer_elemento = transportes[0]`
- Acceder al último elemento de la lista: `el_ultimo_elemento = transportes[-1]`
- Añadir un elemento al final de la lista: `transportes.append(‘moto’)`
- Crear una lista vacía: `transportes = []`

5. Ejemplo: crea la lista de la compra, con 6 productos y luego e imprímela en pantalla
6. En base al ejemplo cinco, el final del programa debe:
    a. Preguntar por otro producto a añadir a la lista
    b. Preguntar qué elemento de la lista quieres visualizar por pantalla.
    c. Imprimir por pantalla los dos últimos elementos de la lista, es decir, el sexto y séptimo.

Existe una forma sencilla de recorrer los elementos de una lista usando un bucle for. Este bucle es un ciclo, que repite indefinidamente lo que está en la segunda o en las segundas líneas mientras se cumpla la condición de la primera línea. En el ejemplo, la condición es que queden elementos en la sista 'transportes'. Mientras recorremos la lista, se van imprimiendo los elementos.

```
For elemento in transportes:
      print(elemento)
```

Comparadores y condicionales if

`if edad == 18:`
`  print("Ya puedes votar")`
`else:`
`  print("NO puedes votar")`	

`age = 12`
`if age < 4:`
`  price = 0`
`elif age < 18:`
`  price = 5`
`else:`
`  price = 10`
`print("La entrada vale: " , price)`	

7. Interface de un cajero automático: debe de preguntar tu nombre, saludarte con dicho nombre. Preguntar qué opción quieres: a) sacar dinero b) aportar dinero. Pregunta la cantidad a imponer o retirar. Despedida anunciando el saldo restante. Partimos de un saldo de 1000€. Subir nota: añade password al cajero.
8. Calculadora con un buen interface, que de la opción de sumar, restar, multiplicar y dividir 2 números

El bucle while

Realiza una serie de comandos mientras se cumpla la condición del while:

```
while n < 6:
      print(n)
      n = n+1
```

Este bucle es muy usado para realizar una tarea repetitiva o añadir datos a una variable hasta que el usuario quiera.

```
while True:
      respuesta = input("¿Quieres seguir añadiendo datos?")
      suma = suma + respuesta
      if respuesta == '0':
            break
```

9. Ejemplo: realiza un contador del uno al diez
10. Realiza un descontador del 100 al 0 de 5 en 5.
11. Crea una lista de la compra vacía. Dentro del bucle 'while' el programa en cada iteración pide un nuevo producto a añadir a la lista y pregunta si deseas continuar añadiendo uno nuevo. Si tecleas 'no', salimos del bucle e imprimimos toda la lista.
12. Vamos a añadir al cajero automático, un bucle infinito para que siempre esté funcionando. El programa saldrá del bucle si escribimos 'exit'. Parte del programa 7.
13. Diseña una aplicación en la que haya que adivinar el número que sale al lanzar un dado 6 veces. Al final nos dar el número de aciertos.
14. Crea una lista con 3 frutas. El programa preguntará al contrincante una fruta y comparará la respuesta con cada uno de los elementos de la lista. Si acierta imprime 'PREMIO'
15. El mismo programa anterior pero se le darán al contrincante 3 intentos. Usa un bucle while ()

16. Igual que el anterior pero la lista de frutas no está preprogramada, sino que debe completarse al principio de la ejecución del programa por tu compañero. Un compañero rellena una lista con tres frutas y el otro tiene 3 intentos para acertar.

La función range ( )

Existe una forma sencilla de hacer un bucle while ( ) cuando conoces de antemano el número de ciclos que necesitas. Estas dos funciones son equivalentes:

```
while (n<10):
      print('hola')
      n +=1
```

```
for n in range(0,9):
      print('hola')
```

17. Realiza el ejercicio anterior cambiando dos de los bucles while por range.

Definir funciones

18. Programar una calculadora con las funciones, suma, resta, multiplicación y división. La calculadora presenta un interface de bienvenida en el que al usuario se le proponen 4 opciones u operaciones. Después pide dos números y ofrece el resultado según la operación escogida.
19. Programa ahora la calculadora con funciones con argumentos, en los que los argumentos a pasar a cada función sean los dos operandos.
20. La calculadora la hacemos ahora con parámetros y un return en cada función. Cada función debe devolver al programa principal el resultado de la suma, resta, multiplicación o división según la operación.
21. Ahora la calculadora tiene una única función llamada 'operacion' (sin tilde). Como ves la función 'operacion(a, b, n_op) debe admitir 3 argumentos, los dos operandos y la opción elegida. Devuelve el resultado, que posteriormente imprime.
22. Guarda en un fichero denominado 'operacion.py' la definición de las funciones o métodos, suma, resta, multiplicación y división, todas admitiendo dos parámetros y devolviendo el resultado de la operación. Importa el módulo en el programa principal, para tener acceso a cada uno de los métodos según la opción elegida
23. Realizar un programa que me calcule el área y perímetro de un cuadrado, un círculo o un triángulo equilátero según la opción requerida. Para ello creamos un módulo denominado geometrias.py con la función triangulo(), cuadrado() y circulo(), sin argumentos y si return, porque cada función se encarga de pedir los datos y de imprimir los resultados.

Clases y objetos

Python es un lenguaje de programación orientada a objetos (POO). La programación orientada a objetos tiene sus raíces en la década del 60, pero es en los 80 que se convierte en el principal paradigma de la programación de un nuevo software.

Hasta ahora los ejercicios y ejemplos que he utilizado han sido utilizando la programación funcional (realizando funciones que den solución a los problemas). En la programación orientada a objetos la atención se centra en la creación de objetos que contienen los datos y funcionalidad juntos.

¿Qué es una Clase?

Una clase es una idea abstracta de una realidad, un concepto. Por ejemplo una clase es la idea de 'vehículo'. Esta clase no vale para nada si no la concretamos en un determinado concepto, por ejemplo coche. Otro objeto sería moto, o camión. Coche sería el objeto de la clase vehículos.. La clase tiene unos atributos o propiedades (variables de estado) y unas funciones o procedimientos (métodos). La clase 'vehículo' tiene unos atributos como es número de ruedas, tipo de motor, número de plazas ... y unos métodos, como pueden ser avance, giro, repostar gasolina ... Hasta que no identificamos los atributos la clase no vale para nada. Por ejemplo, cuatro ruedas, gasoli, 7 plazas. Estás hablando de un monovolumen diesel: ya tienes un objeto.

¿Qué es un Objeto?

Es definir los atributos de una clase de forma que obtengamos un objeto en particular. En la clase 'mascotas', tenemos atributos como 'nombre' 'raza' 'color de pelo' ... y unos métodos: 'comer' 'dormir' 'cazar' ... Para crear un objeto, defino los atributos de la clase: toby, mastín, oscuro. Un objeto es una instancia de clase.

¿Qué es un Atributo?

Los atributos o propiedades de los objetos son las características que puede tener un objeto: Si el objeto fuera Perro, los atributos podrían ser: tamaño, edad, color, raza, etc...

¿Qué es un Método?

Los métodos son la acción o función que realiza un objeto. Si nuestro objeto es Perro, los métodos pueden ser: caminar, ladrar, saltar, dormir, etc...

pygame-zero

Cómo funciona un videojuego

El desarrollo de una jugada el programa debe atender en dentro de un bucle tres funciones:

1. Actualizar las propiedades de todos los personajes y escenarios. Estas propiedades, generalmente son su posición X e Y en los juegos 2D, pero también se pueden actualizar los colores, sonidos, formas y escalas de estos personajes.
2. Dibujar dichos personajes y escenarios en la pantalla, de acuerdo a las actualizaciones anteriores.
3. Atender a los eventos. Un evento de click de ratón, o de tecla pulsada. También puede ser un evento de fin de juego o de tecla de "pause".

Qué es pygame-zero

Es una librería que facilita la creación de videojuegos 2D. Su filosofía es similar a la de Processing. Al igual que Processing, su motor está continuamente llamando de forma indefinida a dos funciones principales:

- función **update( )**
- función **draw ( )**

A demás de esto permite detectar y gestionar eventos de forma muy sencilla. Teclas presionadas, clicks de ratón o eventos de reloj.

La función update ( )

Esta función es llamada por pygame 60 veces por segundo. En ella se actualiza las variables propias de los actores. Lo más común es actualizar la variable de posición X e Y, para dar sensación de movimiento. Pero también se actualizan los colores, la propia imagen o el color del actor, su escala ... Podemos decir que la escena transcurre a 60 fotogramas por minuto

La función draw ( )

Con esta función se redibuja toda la ventana del juego. La frecuencia con la que se redibuja una ventana con sus actores y escenario depende de la función update. Si entre dos fotogramas no han cambiado las propiedades de los actores o escenario no será necesario redibujar la pantalla.

La función draw( ) es dominio de todas las variables locales, por lo que no es necesario definirlas como "**global"**