Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Estado de México

Proyecto Titor

Mensajes desplegados en el videojuego

Escrito por:

Héctor Gónzalez Sánchez - A01753863

Luis Adrián Abarca Gómez - A01798043

Gerardo Rios Mejía- A1753830

Juan Carlos Carro Cruz - A01748640

Alfredo Azamar López - A01798100

Campus Estado de México Mayo 2023

Índice

Instrucciones

1. Muévete hacia adelante con la letra "D" o con la flecha a la derecha ->
2. Muévete a la izquierda con el botón "A" o con la flecha izquierda <-
3. Puedes saltar con la tecla "Espacio"
4. Dispara con el click derecho del mouse
5. Puedes activar el menú de pausa con la tecla "Esc"
6. A Través de los niveles encontrarás estos mismos pergaminos
7. Los pergaminos contendrán información muy valiosa sobre programación
8. Deberás recolectar una cantidad determinada de pergaminos para poder desbloquear los niveles
9. En todos los niveles (incluido este) podrás encontrar pergaminos secretos los cuales desbloquearán un nivel secreto
10. TEMAS DE PROGRAMACIÓN

Variables y operadores en Python

¿Qué es una variable?

Una variable es un contenedor para almacenar información de un valor. Las cuales no tendrán que ser palabras reservadas y además deben ser asignadas mediante el operador de asignación (=).
Ejemplo:

EstaEsUnaVariable = int(100)

- si imprimo esta variable será el valor que le asigne en este caso el valor numérico de 100

Print(EstaEsUnaVariable) <- Aquí imprimo el valor de una variable el cual es de 100

>>100 <- la respuesta en la consola

Valores numéricos

Int = puede almacenar valores enteros ya sea positivos o negativos

Float = incluye valores positivos o negativos que pueden tener decimales o conocidos también como punto flotante

ValorEntero = int(10)

```
ValorFlotante = float(5.375)

print(ValorEntero)
>>10

print(ValorFlotante)
>>5.37
```

Cadenas

Secuencias de caracteres que no se pueden modificar, se pueden actualizar proporcionando una nueva. Los strings son **arreglos** una forma de ordenar datos el cual es inmutable (no se puede modificar) por lo cual podemos tener “casillas” de nombre índice para cada carácter en un string.

```
ValorString = String(“Hello, world!”)
print(ValorString)

>> Hello, world!
```

```
H e l l o ,  w o r l d !
 0 1 2 3 4 5 6 7 8 9 10 11 12
```

Así se ve un arreglo de caracteres del hello world con valores desde la izquierda a la derecha donde cada letra se corresponde a un índice. El primer índice siempre es 0

```
 H  e  l  l  o  ,     w  o  r  l  d !
-13 -12 -11 -10 -9  -8  -7  -6  -5  -4 -3 -2 -1
```

Estos serían los valores del arreglo si empezamos a contar desde la derecha a la izquierda números negativos donde los índices negativos van desde el -1 hasta el -13

Valores booleanos

Los valores booleanos son expresiones que representan dos valores, falso o verdadero.
Se utilizan al comparar dos valores.
Ejemplo

```
print (10 > 9)
>>True
print(10 == 9)
>>False
print (10 < 9)
>>False
```

La mayor parte de los valores son verdaderos, excepto por el cero que al compararlo será **False.**

Operadores

Los operadores como su nombre lo indica son usados para hacer operaciones de variables y valores.
Variables
x = 5
y = 2

Operadores aritméticos

(+) Adición es una suma de valores
x + y = 7

(-) Sustracción una resta o diferencia de valores
x - y = 3

(*) Multiplicación de los valores
x * y = 10

(/) División de valores
x / y = 2.5

(%) Módulo el módulo regresa el residuo de la división
x % y = 1
donde uno es el residuo que no se puede dividir entre 2

(**) Potencia de valores el primer valor se eleva a la potencia del segundo valor dado
x ** y = 5^5 = 25

(//) División del piso regresa valores enteros de la división (no muestra el residuo)
x // y = 2

Operadores de comparación

Los operadores de comparación como su nombre lo indica sirven para comparar los valores de una variables, pueden ser valores de cualquier tipo ya sean numéricos o texto y devolverá True o False dependiendo de la condición.

Definición de las variables

w = 5
x = 5
y = 2
z = 2

(==) La igualdad compara dos variables y arroja True solo si ambas variables son iguales
print(w == x)
>>True
print(w == y)
>>False

(!=) La desigualdad compara las variables y obtiene True solo si ambas variables son diferentes
print(w == x)
>>False
print(w == y)
>>True

(>) El mayor que compara las variables y obtiene True si la variable que se encuentra a la izquierda es mayor que la variable del lado derecho.
print (x > y)
>>True
print(y > x)
>>False

(<) El menor que compara esas variables y obtiene True si el valor que se encuentra en la izquierda es menor que la la variable del otro lado
print (y < x)
>>True
print(x < y)
>>False

(<=)(>=) Los operadores mayor o igual que y el menor o igual que son la combinación de los anteriores permitiendo que el resultado sea True cuando se cumplan las condiciones.
print(w >= x)
>>True
print(y <= z)
>>True

Operadores lógicos

x = 5
Los operadores lógicos son usados para combinar declaraciones de condición
(and) regresa True si ambas declaraciones (condición) son verdaderas
print(x > 3 and x < 10)
>>True

(or) regresa True si una de las dos declaraciones (condición)
print(x > 3 or x < 4)
>>True

(not) invierte el resultado, si el resultado es True regresa False y viceversa
print(not(x > 3 and x < 10))
>>False

Condiciones y Ciclos (Bucles)

Las condiciones y las condiciones If, Como ya se vio anteriormente la condiciones se basan en los operadores lógicos y los operadores de comparación pero regresan True o False.

Identación

En python la manera de definir el alcance de una sección de código es mediante la sangría o indentación del código.

If, Elif y Else

if

Ejecuta una sección de código si se cumple (True) la condición establecida.

```
a = 33
b = 200
if b > a:
        print("b es mayor que a") <- en esta sección el código se encuentra identada en la
                                     condición If
>>b es mayor que a
```

elif

Elif es una palabra en python para decir “si las condiciones previas no se cumplieron(fueron True), entonces prueba esta condición”

```
a = 33
b = 33
if b > a:
        print("b es mayor que a")
elif a == b:
        print("a y b son iguales")
>>a y b son iguales
```

En la sección de código anterior la primera condición no se cumple entonces busca la siguiente condición.

else

Else es una palabra que captura todo lo que no se captó en condiciones arroja un resultado

```
a = 33
b = 200
if a > b:
  print("a es mayor que b")
elif a == b:
  print("a y b son iguales")
else:
  print("a es mayor que b")
>>a es mayor que b
```

En esta sección de código las primeras dos condiciones no se cumplen por lo cual no se ejecutan las instrucciones del print por lo que queda solo la última condición que se ejecuta si las demás no.

Ciclos

Los bucles o ciclos se utilizan en código para establecer sentencias o secciones de código que se repiten o iteran. El ciclo se repite mientras la condición establecida deja de cumplirse.

While

El ciclo while puede ejecutar una sección de código tantas veces como la condición sea verdadera (True).

```
i = 1
while i < 6:
  print(i)
  i += 1
>>1
>>2
>>3
>>4
>>5
>>6
```

Nota: Si la condición se mantiene verdadera se puede generar un bucle infinito.

Break

break puede detener un ciclo si la condición es verdadera (True)

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
>>1
>>2
>>3
```

En el caso anterior el ciclo va sumando uno a la variable i hasta que entra en la condición If que comprueba si el valor i sea igual a 3 cuando se cumple el break termina con el ciclo.

Continue

continue puede detener (o saltar por así decirlo) la iteración actual de un ciclo y continúa con las siguientes

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

```
>>1
>>2
>>4
>>5
>>6
```

El continue salto el print del valor 3

for

El ciclo for hace iteraciones en una secuencia, con el for puede ejecutar una sección de sentencias por cada elemento que haya en una lista, tupla o conjunto.

Range

El uso de range regresa una secuencia de número empezando desde el 0 por defecto e incrementando 1 por defecto también, y se detiene en un número especificado.
(start) Opcional. Un número entero que establece la posición donde se inicia
(stop) Requerido. Un entero que establece la posición a detenerse
(step) Opcional. Un entero que establece los incrementos que se harán
Range (Start, Stop, Step)

```
for i in range(0, 5, 1):
    print(i, end=", ")
>> 0 , 1, 2, 3, 4,
```

En la sentencia anterior se establecen los conteos con el inicio (start) en 0, el final del conteo en 5 (Stop), y el incremento(step) establecido en 1.

Obtener los elementos de una lista

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
>> apple
>> banana
>> cherry
```

Obtener los elementos de un string

```
for x in "banana":
  print(x)
>>b
>>a
>>n
```

>>a
>>n
>>a

Break
el uso del break hace que los ciclos se detengan en cuanto una condición se cumpla

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

>>apple
>>banana
Cuando se detectó el valor banana detuvo los ciclos por lo cual ya no se mostró “cherry”

Continue
continue como en el While sirve para saltar una seccion de codigo a realizar

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

>>apple
>>cherry
Como se puede observar el salto de la palabra banana y continua con el siguiente valor en la lista que es “cherry”

Funciones

Una función es un bloque de código que solo corre cuando es llamado.
Puedes darle datos conocidos como parámetros a una función puede regresar datos como resultado.

def

Con la palabra def puede definir una función la cual puede hacer lo que está contenido en ella.

Ejemplo de salida de texto
En el siguiente ejemplo definimos una función llamada mi_funcion, que imprime el mensaje de “Hola desde una función”, la función no se activa hasta que es llama en la línea donde se aplica solo el mi_funcion()

```
def mi_funcion():
  print("Hola de una función")
mi_funcion()
>> Hola de una función
```

Argumentos de la función
Los argumentos pueden ser especificados después del nombre de la función entre los paréntesis, puedes poner muchos argumentos separados por comas.

En el ejemplo siguiente al darle un argumento de algún alimento se le sumará el string “ de manzana”

```
def funcion_alimento(alimento):
  print(alimento + " de manzana")
funcion_alimento("Helado")
funcion_alimento("Pay")
funcion_alimento("Ensalada")
>>Helado de manzana
>>Pay de manzana
>>Ensalada de manzana
```

Return
El return es una declaración en Python que se utiliza para devolver un valor desde una función. Cuando una función encuentra una declaración return, se detiene en ese punto y devuelve el valor especificado. Si no se utiliza el return en una función, se considera que la función devuelve None de forma implícita al final de su ejecución.

```
def calcular_cuadrado(numero):
    cuadrado = numero * numero
    return cuadrado

resultado = calcular_cuadrado(5)
print(resultado)
>> 25
```

Función sin return:

```
def calcular_cuadrado(numero):
    cuadrado = numero * numero
    return cuadrado

resultado = calcular_cuadrado(5)
print(resultado)
>> None
```

Datos extra

1.

El CamelCase es una convención en la nomenclatura de la programación la cual se identifica por usar mayúsculas o letra capital

UpperCamelCase = la primera letra de las palabras es mayúscula
lowerCamelCase = a diferencia de la anterior la primera letra es minúscula

2.

Si bien en este caso el lenguaje utilizado (Python) reconoce las variables que le estamos dando de manera automática, lo recomendable es hacer **casting** que es especificar el tipo de variable que estamos usando.
Un ejemplo de ello seria:

```
flotante = 15.55
```

Donde el valor tiene la definición de ser un flotante y el valor va entre paréntesis () para identificarlo.

```
flotanteConCasting = float(15.55)
```

3.

Las operaciones en el lenguaje de programación python se hacen mediante la jerarquía de operaciones y se hacen en el siguiente orden:

A. (), Los paréntesis y lo que se encuentra dentro de ellos va a ser la primera operación.
B. ** sqrt, La potencia y las raíces serían la segunda operación.
C. * /, La multiplicación y la división sería la tercera operación.
D. + -, La suma y la resta será la cuarta y última operación.

4.

El primer programa de computadora de la historia fue escrito por Ada Lovelace en 1843. Ada Lovelace, una matemática británica, escribió un algoritmo para ser ejecutado en la Máquina Analítica, un dispositivo mecánico diseñado por Charles Babbage. Este algoritmo se considera el primer programa de la historia, lo que hace de Ada Lovelace la primera programadora conocida.

Este dato destaca la importancia histórica de la programación y cómo ha evolucionado desde sus inicios. A partir de Ada Lovelace, muchas personas han contribuido al desarrollo de la programación y han sentado las bases para los lenguajes y las tecnologías que utilizamos hoy en día.

5.

¿Sabías que en Python puedes utilizar bibliotecas para agregar funcionalidades adicionales a tus programas? Las bibliotecas son como cajas de herramientas llenas de funciones y herramientas útiles que otros programadores han creado y compartido. Imagina que estás construyendo una casa y necesitas herramientas especiales para hacer el trabajo más rápido y fácil. Las bibliotecas en Python son como esas herramientas especiales que te ayudan a realizar tareas comunes de programación de manera más eficiente.

Lo increíble de las bibliotecas en Python es que hay una para casi cualquier cosa que puedas imaginar. Puedes encontrar bibliotecas para manipular imágenes, realizar operaciones matemáticas avanzadas, interactuar con bases de datos, crear gráficos y mucho más. Estas bibliotecas son desarrolladas por una comunidad de programadores apasionados que las comparten libremente para que todos podamos aprovecharlas.