Introducción

TECSUP
Pasión por la Tecnología

Introducción al lenguaje
Instalación de Python
Sintaxis
Uso de Operadores
Estructuras de control
Estructura de datos

Introducción

Python es un lenguaje de alto nivel, interpretado y orientada a objetos.

- Python is Interpreted: Python se procesa en tiempo de ejecución por el intérprete. No es necesario para compilar el programa antes de ejecutarlo. Esto es similar a Perl y PHP.

- Python is Object-Oriented: Python admite el estilo orientado a objetos o una técnica de programación que encapsula el código de los objetos.

- Python is a Beginner's Language: Python es un gran lenguaje de los programadores de nivel principiante y apoya el desarrollo de una amplia gama de aplicaciones de procesamiento de texto simple a los navegadores WWW para juegos.

Mi primer programa

- Todo programa en Python tiene la extensión .py . Escriba el siguiente código en un archivo ejemplo1.py:

```
print ("Hello, Python!")
```

Identificadores de Python

Un identificador Python es un nombre usado para identificar una variable, función, clase, módulo u otro objeto. Un identificador comienza con una letra A a la Z o A a la Z o un guión bajo (_) seguido de cero o más letras, guiones y los dígitos (0 a 9) .

Python no permite caracteres de puntuación tales como @, $,% y dentro de identificadores.

Aquí están las convenciones de nomenclatura de los identificadores Python:

- nombres de las clases comienzan con una letra mayúscula. Todos los otros identificadores comienzan con una letra minúscula.
- A partir de un identificador con un solo subrayado inicial indica que el identificador es privado.
- A partir de un identificador con dos subrayado iniciales indica un identificador fuerte privada.

Palabras reservadas

- Las palabras reservadas no se puede utilizar como constante o variable, o cualquier otro nombre de identificador. Todas las palabras clave de Python contienen sólo letras minúsculas.

and	exec	Not
as	finally	or
assert	for	pass
break	from	print
class	global	raise
continue	if	return
def	import	try
del	in	while
elif	is	with
else	lambda	yield
except		

Tipos de variables

Las variables son nada más que las posiciones de memoria reservadas para almacenar valores. Esto significa que cuando se crea una variable de reservar un poco de espacio en la memoria.

Con base en el tipo de datos de una variable, el intérprete asigna memoria y decide lo que se puede almacenar en la memoria reservada. Por lo tanto, mediante la asignación de diferentes tipos de datos de las variables, se puede almacenar números enteros, decimales o personajes de estas variables.

Asignación de valores

- Las variables de Python no necesitan declaración explícita para reservar espacio en la memoria. La declaración se produce automáticamente cuando se asigna un valor a una variable. El signo igual (=) se utiliza para asignar valores a las variables.
- El operando a la izquierda del operador = es el nombre de la variable y el operando a la derecha del operador = es el valor almacenado en la variable.

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string

print (counter)
print (miles)
print (name)
```

Asignación de valores

- Las variables de Python no necesitan declaración explícita para reservar espacio en la memoria. La declaración se produce automáticamente cuando se asigna un valor a una variable. El signo igual (=) se utiliza para asignar valores a las variables.
- El operando a la izquierda del operador = es el nombre de la variable y el operando a la derecha del operador = es el valor almacenado en la variable.

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string

print (counter)
print (miles)
print (name)
```

Asignación múltiple

Python le permite asignar un valor único a varias variables simultáneamente.

- Un objeto entero se crea con el valor 1, y las tres variables se asignan a la misma posición de memoria.

```
a = b = c = 1
```

- Dos número entero objetos con los valores 1 y 2 son asignados a las variables a y b respectivamente, y un objeto de cadena con el valor "john" se asigna a la variable c.

```
a, b, c = 1, 2, "john"
```

Operadores aritméticos Python

Supongamos que la variable a mantiene 10 y la variable B tiene 20, entonces:

Operador	Descripción	Ejemplo
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a - b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b% a = 0
** Exponent	Performs exponential (power) calculation on operators	a ** b = 10 a la potencia 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) :	9 // 2 = 4 y 9,0 // 2,0 = 4,0, -11 // 3 = -4, -11,0 // 3 = -4.0

Operadores de comparación Python

Estos operadores comparan los valores a ambos lados de ellos y deciden la relación entre ellos. También se les llama operadores relacionales.

Supongamos que una variable A tiene 10 y la variable B tiene 20, entonces:

Operador	Descripción	Ejemplo
==	If the values of two operands are equal, then the condition becomes true.	(a == b) no es cierto.
!=	If values of two operands are not equal, then condition becomes true.	
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) es verdadera. Esto es similar a! = Operador.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) no es cierto.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) es cierto.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) no es cierto.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) es cierto.

Operadores de asignación de Python

Supongamos que una variable A tiene 10 y la variable B tiene 20, entonces:

Operador	Descripción	Ejemplo
=	Assigns values from right side operands to left side operand	c = a + b asigna valor de a + b en c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c + = a es equivalente a c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c - = a es equivalente a C = C - una
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c * = a es equivalente a c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c / = a es equivalente a c = c / ac / = a es equivalente a c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c% = a es equivalente a c = c% una
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c ** = a es equivalente a c = c ** una
//= Floor Division	It performs floor division on operators and assign value to the left operand	c // = a es equivalente a c = c // una

Operadores lógicos Python

Asumir una variable A contiene Verdadero y Falso la variable B tiene entonces:

Operador	Descripción	Ejemplo
and Logical AND	If both the operands are true then condition becomes true.	(A y B) es falsa.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) es verdadera.
not Logical NOT	Used to reverse the logical state of its operand.	No (A y B) es verdadera.

La función de impresión

- Un Cambio elemental y ampliamente conocido en Python 3 es cómo se utiliza la función de impresión **print().** El uso de los paréntesis () con función de impresión es ahora obligatorio.

```
print "Hello World" #is acceptable in Python 2
print ("Hello World") # in Python 3, print must be followed by ()
```

La función de lectura

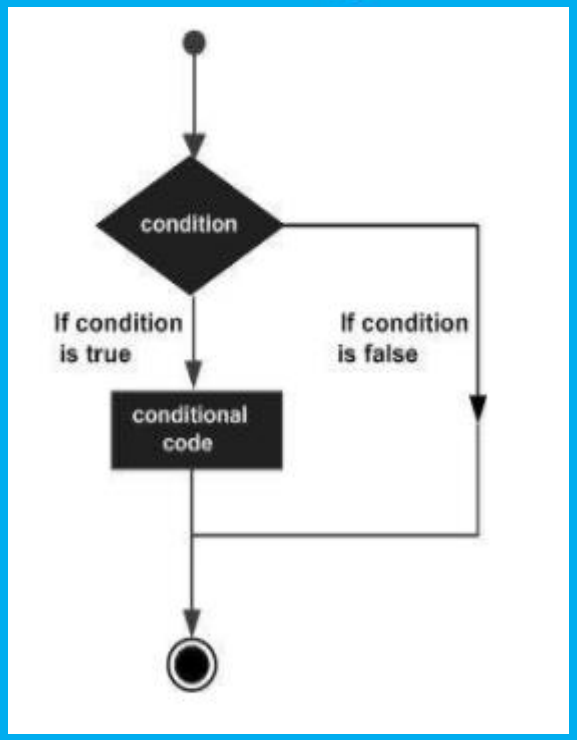
Python 3 emplea la función **input()** para la lectura de datos en tiempo de ejecución.

```
In Python 3
>>> x=input("something:")
something:10
>>> x
'10'
>>> x=input("something:")
something:'10' #entered data treated as string with or without ''
>>> x
"'10'"
>>> x=raw_input("something:") # will result NameError
Traceback (most recent call last):
  File "", line 1, in

    x=raw_input("something:")
NameError: name 'raw_input' is not defined
```

Toma de decisiones

- La toma de decisiones es la anticipación de condiciones que ocurren mientras que la ejecución del programa y que especifican las medidas tomadas de acuerdo con las condiciones.

- las estructuras de toma evalúan múltiples expresiones que producen VERDADERO o FALSO como resultado. Es necesario determinar qué medidas tomar y qué estados que se ejecuta si el resultado es verdadero o falso de lo contrario.



Toma de decisiones

- Python asume ninguna **non-zero y non-null valores como verdadero, y si lo es, ya sea zero o null , entonces se asume como valor FALSO.**
- Python proporciona los siguientes tipos de declaraciones de toma de decisiones.

Declaración	Descripción
if statements	Una **if statement se compone de una expresión booleana seguida de una o más sentencias.**
if...else statements	Una **if statement puede ser seguido por una opcional else statement , que se ejecuta cuando la expresión booleana es FALSO.**
nested if statements	Puede utilizar uno **if o else if la declaración dentro de otro if o else if statement(s) .**

Declaración IF

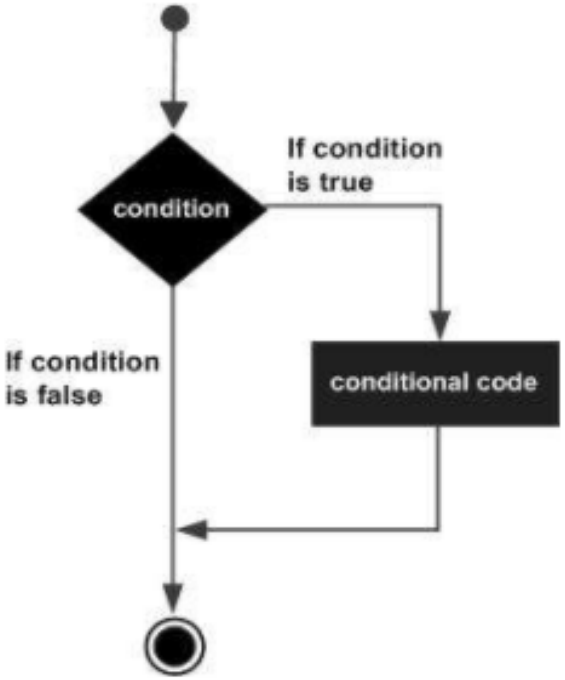
- La declaración IF contiene una expresión lógica mediante la cual los datos se comparan y se toma una decisión en base al resultado de la comparación.

Sintaxis

```
if expression:
   statement(s)
```

- Si la expresión booleana se evalúa como TRUE, entonces el bloque de statement(s) dentro de la sentencia if se ejecuta. En Python, las declaraciones en un bloque se sangran de manera uniforme después: símbolo. Si la expresión booleana se evalúa como FALSE, a continuación, se ejecuta el primer conjunto de código después del final del bloque.

Declaración IF




```
var1 = 100
if var1:
   print ("1 - Got a true expression value")
   print (var1)

var2 = 0
if var2:
   print ("2 - Got a true expression value")
   print (var2)
print ("Good bye!")
```

Cuando se ejecuta el código anterior, se produce el siguiente resultado -

```
1 - Got a true expression value
100
Good bye!
```

Declaración if...else

- La declaración else se puede combinar con una declaración if. Una declaración else contiene el bloque de código que se ejecuta si la expresión condicional de la sentencia if devuelve un 0 o un valor FALSO.
- La afirmación else es una declaración opcional y puede haber como máximo una sola else comunicado tras if .

Sintaxis

La sintaxis de la *if...else* declaración es -

```
if expression:
   statement(s)
else:
   statement(s)
```

```
amount=int(input("Enter amount: "))

if amount<1000:
    discount=amount*0.05
    print ("Discount",discount)
else:
    discount=amount*0.10
    print ("Discount",discount)

print ("Net payable:",amount-discount)
```

Declaración elif

- La declaración elif permite comprobar múltiples expresiones de TRUE y ejecutar un bloque de código tan pronto como una de las condiciones evalúa como TRUE.

sintaxis

```
if expression1:
   statement(s)
elif expression2:
   statement(s)
elif expression3:
   statement(s)
else:
   statement(s)
```

```
amount=int(input("Enter amount: "))

if amount<1000:
    discount=amount*0.05
    print ("Discount",discount)
elif amount<5000:
    discount=amount*0.10
    print ("Discount",discount)
else:
    discount=amount*0.15
    print ("Discount",discount)

print ("Net payable:",amount-discount)
```

Declaración if anidado

- Puede haber una situación en la que desea comprobar para tratar otra condición después de una condición se resuelve en true. En tal situación, se puede utilizar el anidado if constructo.
- En un anidado if constructo, se puede tener una if...elif...else construir dentro de otro if...elif...else construcción.

Sintaxis:

La sintaxis de la anidada *if...elif...else* construcción puede ser:

```
if expression1:
   statement(s)
   if expression2:
      statement(s)
   elif expression3:
      statement(s)
   else
      statement(s)
elif expression4:
   statement(s)
else:
   statement(s)
```

```
num=int(input("enter number"))
if num%2==0:
    if num%3==0:
        print ("Divisible by 3 and 2")
    else:
        print ("divisible by 2 not divisible by 3")
else:
    if num%3==0:
        print ("divisible by 3 not divisible by 2")
    else:
        print  ("not Divisible by 2 not divisible by 3")
```

Bucle for loop

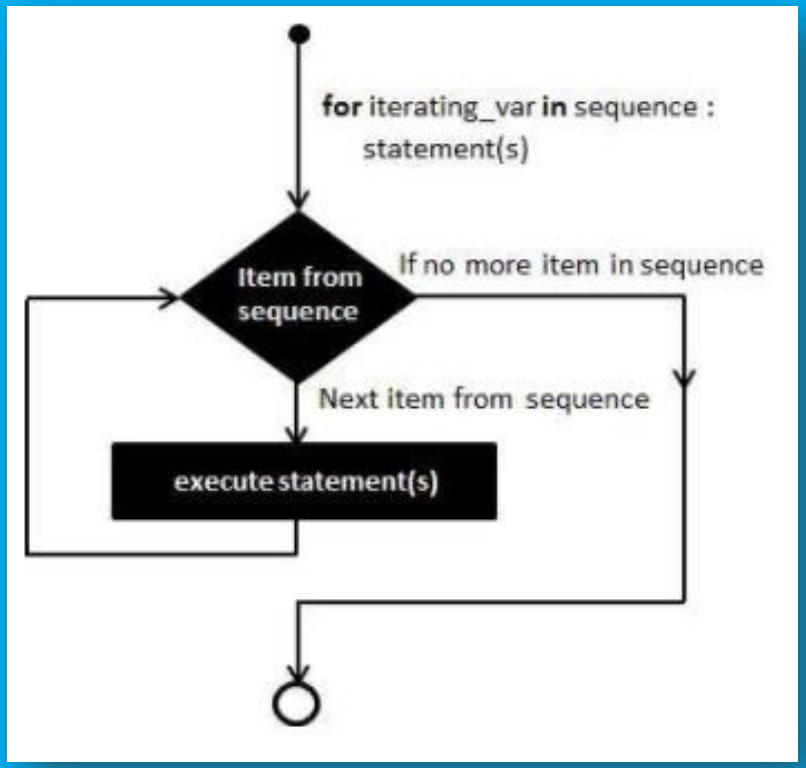
- Ejecuta una secuencia de instrucciones varias veces y abrevia el código que gestiona la variable de bucle.
- La *sentencia* en Python tiene la capacidad de iterar sobre los elementos de cualquier secuencia, como una lista o una cadena.

Sintaxis

```
for iterating_var in sequence:
   statements(s)
```

- Si una secuencia contiene una lista de expresiones, se evalúa primero. Entonces, el primer elemento de la secuencia se asigna a la variable de iteración iterating_var . A continuación, se ejecuta el bloque de declaraciones. Cada elemento de la lista se asigna a iterating_var , y la statement(s) bloque se ejecuta hasta que se agota toda la secuencia.

Bucle for loop




```
for letter in 'Python':     # traversal of a string sequence
   print ('Current Letter :', letter)
print()
fruits = ['banana', 'apple',  'mango']
for fruit in fruits:        # traversal of List sequence
   print ('Current fruit :', fruit)

print ("Good bye!")
```

Cuando se ejecuta el código anterior, se produce el siguiente resultado

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n

Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

Proposiciones de control Loop

- Las sentencias de control de bucle cambian la ejecución de su secuencia normal. Cuando la ejecución deja un ámbito, todos los objetos automáticos que se crearon en ese ámbito se destruyen.

Declaración de control	Descripción
break statement	Termina la ejecución de sentencias de bucle y las transferencias a la cuenta inmediatamente después del bucle.
continue statement	Hace que el bucle para saltar el resto de su cuerpo e inmediatamente volver a probar su condición antes de reiterar.
pass statement	La sentencia pass de Python se utiliza cuando se requiere una declaración sintácticamente pero no quiere ningún comando o código a ejecutar.

Sentencia break

- La sentencia break se utiliza para la terminación prematura del bucle. Después de abandonar el bucle, la ejecución en la siguiente sentencia se reanuda.
- La declaración break puede ser usada tanto en bucles while y for.
- Si está utilizando bucles anidados, la sentencia break se detiene la ejecución del bucle más interior e inicia la ejecución de la siguiente línea de código después del bloque.

```
for letter in 'Python':     # First Example
   if letter == 'h':
      break
   print ('Current Letter :', letter)

var = 10                    # Second Example
while var > 0:
   print ('Current variable value :', var)
   var = var -1
   if var == 5:
      break

print ("Good bye!")
```

Cuando se ejecuta el código anterior, se produce el siguiente resultado

```
Current Letter : P
Current Letter : y
Current Letter : t
Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Good bye!
```

Sentencia continue

- La declaración continue en Python devuelve el control al comienzo del bucle. El bucle comienza la próxima iteración sin ejecutar las sentencias restantes en la iteración actual.
- La declaración continue puede ser usada tanto en bucles while y for.

```
for letter in 'Python':     # First Example
   if letter == 'h':
      continue
   print ('Current Letter :', letter)

var = 10                    # Second Example
while var > 0:
   var = var -1
   if var == 5:
      continue
   print ('Current variable value :', var)
print ("Good bye!")
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Current variable value : 0
Good bye!
```

Sentencia pass

- Se utiliza cuando se requiere una declaración sintácticamente pero no quiere ningún comando o código a ejecutar.

- La declaración pass es una operación null; no pasa nada cuando se ejecuta. El pass también es útil en lugares donde su código con el tiempo ir, pero no se ha escrito todavía.

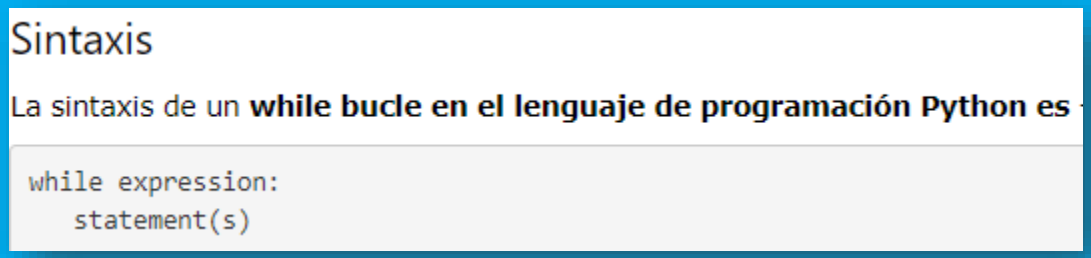
```
for letter in 'Python':
   if letter == 'h':
      pass
      print ('This is pass block')
   print ('Current Letter :', letter)

print ("Good bye!")
```

Cuando se ejecuta el código anterior, se produce

```
Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
Good bye!
```

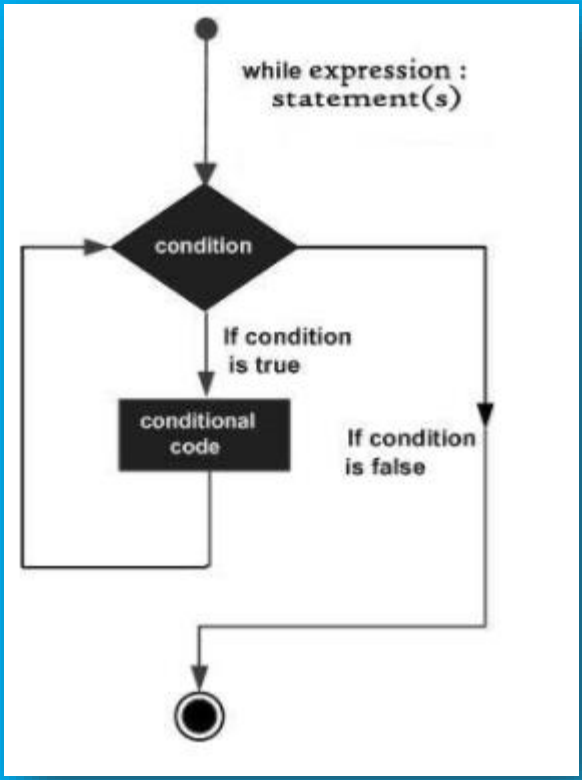
Bucle while loop

- La sentencia de bucle while en el lenguaje de programación Python ejecuta repetidamente una declaración de destino, siempre y cuando una determinada condición es verdadera..



- En este caso, statement(s) puede ser una sola instrucción o un bloque de instrucciones con guión uniforme. La condition puede ser cualquier expresión, y verdadero es cualquier valor distinto de cero. Los itera de bucle mientras que la condición es verdadera.
- Cuando la condición se convierte en falsa, el control de programa pasa a la línea inmediatamente siguiente del bucle.

Bucle while loop




```
count = 0
while (count < 9):
   print ('The count is:', count)
   count = count + 1

print ("Good bye!")
```

Cuando se ejecuta el código anterior, se produce el siguiente resultado

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

El bucle infinito

- Un bucle se convierte en bucle infinito si una condición nunca se convierte en falsa. Usted debe tener cuidado al usar los bucles while, debido a la posibilidad de que esta condición no se resuelve en un valor FALSO. Esto se traduce en un bucle que nunca termina. Tal bucle se llama un bucle infinito.
- Un bucle infinito podría ser útil en la programación cliente / servidor, donde el servidor necesita para funcionar de forma continua para que los programas cliente puede comunicarse con ella cuando sea necesario.

```
var = 1
while var == 1 :  # This constructs an infinite loop
   num = int(input("Enter a number  :"))
   print ("You entered: ", num)

print ("Good bye!")
```

Cuando se ejecuta el código anterior, se produce el siguiente resultado

```
Enter a number  :20
You entered:  20
Enter a number  :29
You entered:  29
Enter a number  :3
You entered:  3
Enter a number  :11
You entered:  11
Enter a number  :22
You entered:  22
Enter a number  :Traceback (most recent call last):
  File "examples\test.py", line 5, in
    num = int(input("Enter a number  :"))
KeyboardInterrupt
```

Bucles anidados

Sintaxis

```
for iterating_var in sequence:
   for iterating_var in sequence:
      statements(s)
   statements(s)
```

La sintaxis de una **nested while loop declaración en lenguaje de programación Python es el siguiente**

```
while expression:
   while expression:
      statement(s)
   statement(s)
```

Bucles anidados

- El siguiente programa utiliza un bucle for anidados para mostrar las tablas de multiplicar del 1-10

```
for i in range(1,11):
    for j in range(1,11):
        k=i*j
        print (k, end=' ')
    print()
```

- La función print() de bucle interior tiene final ='' que añade un espacio en lugar de salto de línea por defecto. Por lo tanto, los números aparecerán en una fila.
- Última print() se ejecuta al final de interior para bucle

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
```

Glosario

- **Toma de decisiones**: Son instrucciones para evaluar condiciones y ejecutar sentencias en función si cumple o no la condición
- **Lectura de datos:** Es la sentencia orientada a capturar valores ingresado por teclado
- **Impresión:** Se refiere a la salida de datos
- **Identificadores**: Son palabras clave del lenguaje
- **Variables:** Son posiciones de memoria reservadas para almacenar valores. Esto significa que cuando se crea una variable de reservar un poco de espacio en la memoria.
- **Operadores:** Son funciones orientadas a asignar valores y compararlas.
- **Bucle:** Es un proceso repetitivo para ejecutar una o más instrucciones
- **For:** Es el comando para ejecutar procesos cíclicos y/o repetitivos
- **Break:** Es el comando para detener el proceso cíclico
- **Continue:** Es el comando para continuar con el proceso cíclico