

Clase 5

Programar en Python

Introducción a funciones y ejemplos prácticos

En esta clase veremos una serie de códigos de ejemplo con diferente nivel de complejidad e introduciremos el concepto de función, el cual nos resultará sumamente práctico al ir creciendo en la dimensión de nuestros códigos.

Algunos ejemplos prácticos

A continuación, realizaremos varios ejemplos prácticos que nos servirán para ir agilizando nuestra codificación en python.

Antes de comenzar, es útil saber cómo realizar la codificación desde un archivo de texto y luego llamarla desde el intérprete.

Escribiendo y ejecutando el código desde un archivo

Los programas que realizamos pueden ser ejecutados directamente desde un archivo de texto almacenado en nuestro sistema.

Si queremos realizar los programas en un archivo, los mismos deben ser de texto plano, respetar a estructura de código python, es decir la indentación fundamentalmente, y almacenarse con la extensión .py, por ejemplo: holaMundo.py

Para ejecutar dicho código directamente desde un archivo en vez de copiarlo al intérprete interactivo, lo que tenemos que hacer es ingresar al directorio en donde se encuentra almacenado dicho archivo y escribir la siguiente línea: python3 holaMundo.py

Esto ejecutará directamente el código sin mostrarnos todas las líneas que escribimos, las cuales las podremos editar, de ser necesario, directamente desde el archivo que creamos.

El primer y más básico ejemplo práctico que podemos mencionar, y que ya hemos probado es el famoso y nunca bien ponderado “Hola Mundo”.

Hola Mundo

Para darle cierto aire de novedad a este código, vamos a escribirlo en un documento. Para ello en la consola de Linux, vamos a crear un nuevo archivo de la siguiente manera:

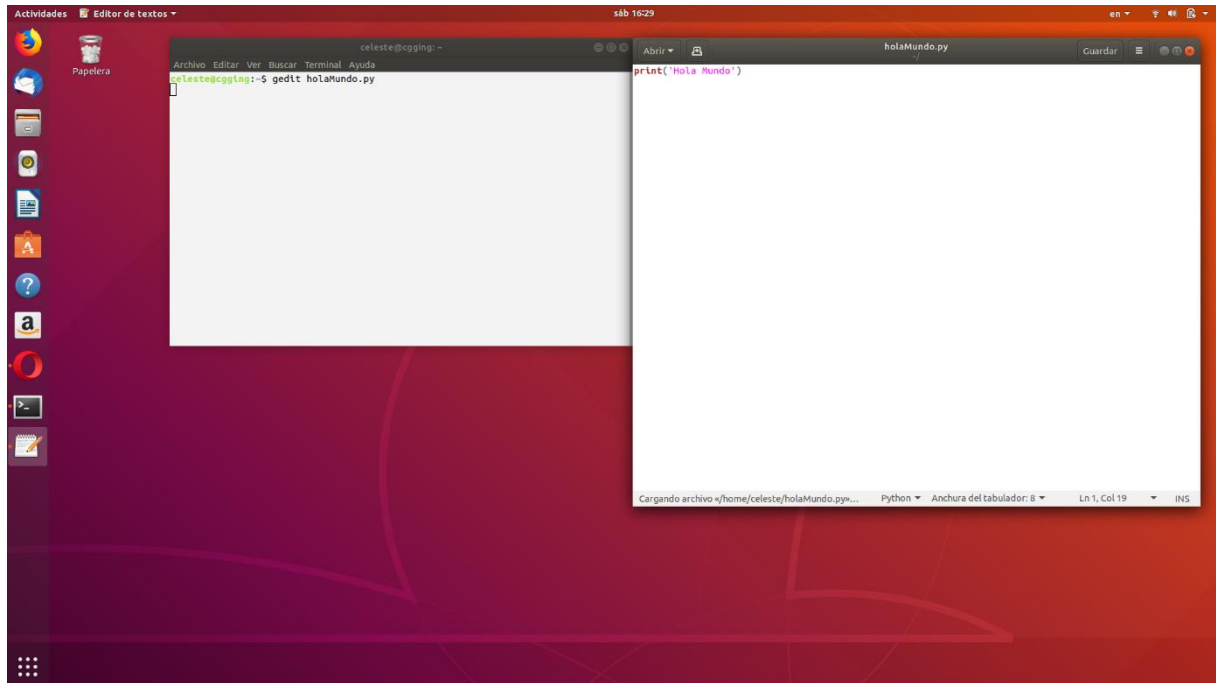
```
gedit holaMundo.py
```

Lo cual nos creará, en el caso de que no exista y abrirá en modo de edición el archivo holaMundo.py

Una vez adentro del archivo, procederemos a escribir las siguientes líneas de código:

```
print('Hola Mundo')
```

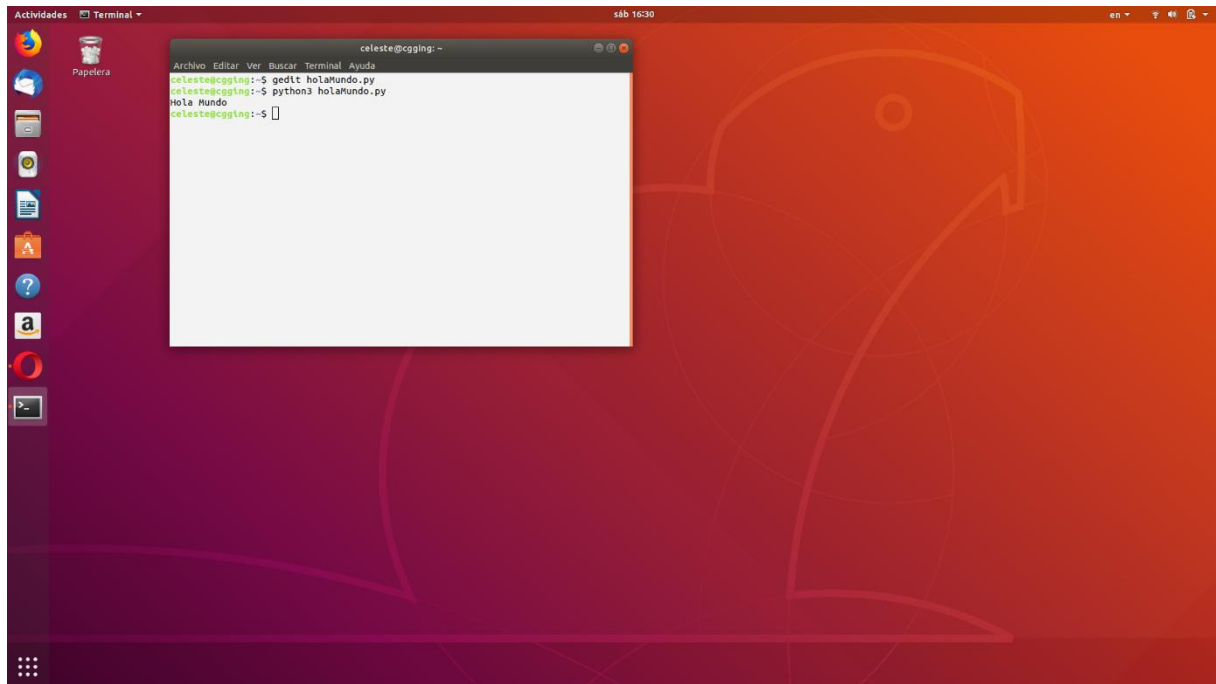
luego, procederemos a guardar y cerrar el archivo.



Para ejecutar el código que acabamos de escribir, simplemente basta con escribir desde la consola:

```
python3 holaMundo.py
```

lo cual producirá la ejecución del código y nos mostrará en la consola los resultados



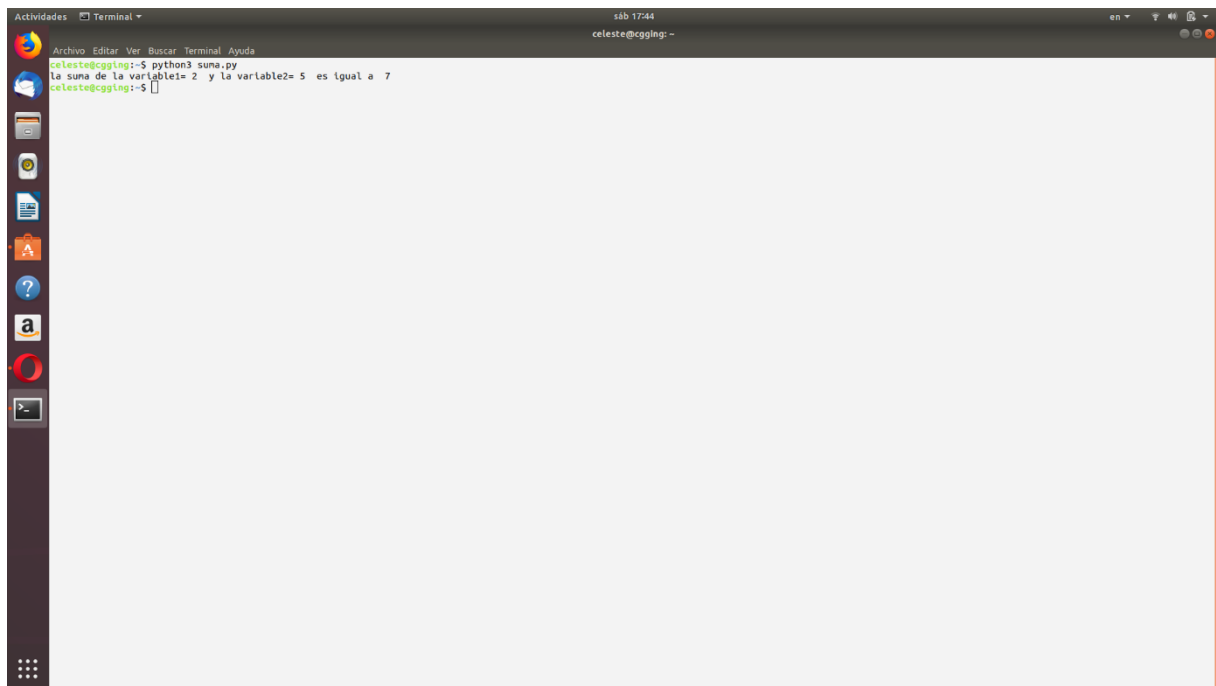
Si bien este ejemplo es sumamente básico, nos sirve para practicar la edición de un archivo de código y su ejecución directamente desde la consola en lugar de trabajar solo con la consola interactiva de python.

Suma

En el ejemplo que presentamos a continuación, inicializaremos dos variables, las sumaremos y mostraremos el resultado por pantalla:

```
variable1=3  
variable2=5  
suma= variable1 + variable2  
print("la suma de la variable 1=" , variable1, " y la variable 2=", variable2, "es igual a ",  
suma)
```

al ejecutar el código del archivo que acabamos de generar, observaremos lo siguiente:

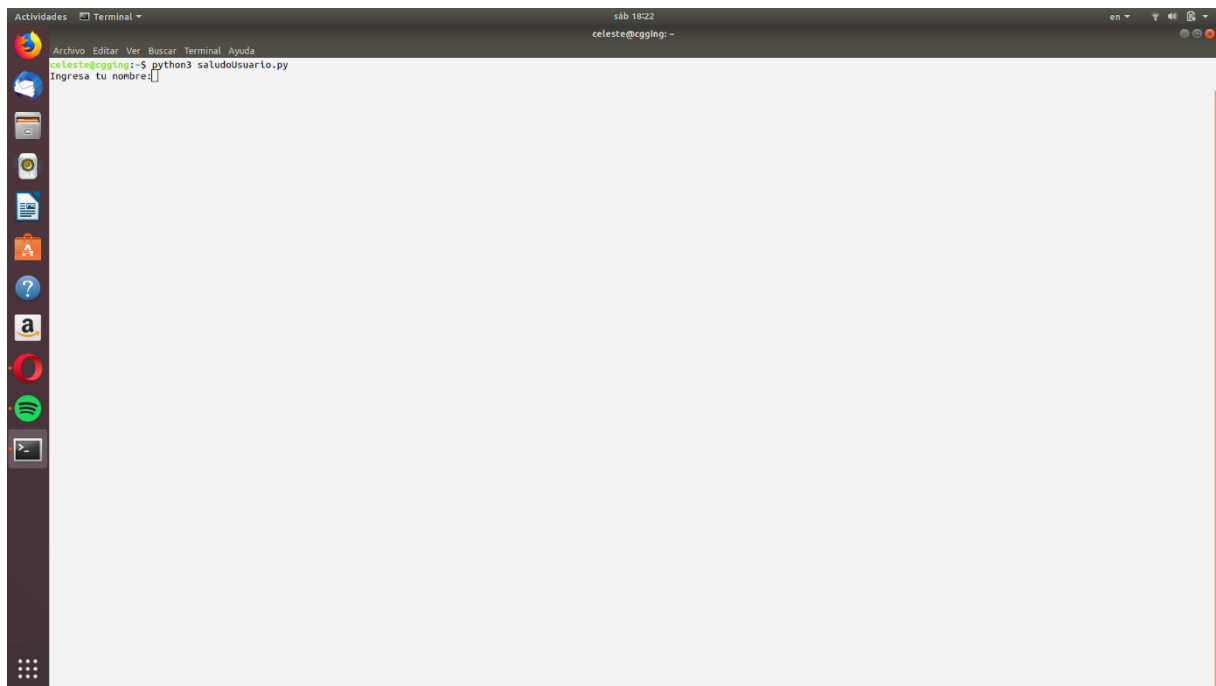


Saludando al usuario

Algo que nos resultara de suma importancia cuando programemos es la interacción con el usuario, habrá ocasiones en las cuales el usuario deberá proporcionar datos al programa para que este pueda ejecutarse. Normalmente la proporción de datos se produce mediante el teclado, y para poder recibir estos datos contamos con la función `input()` en python:

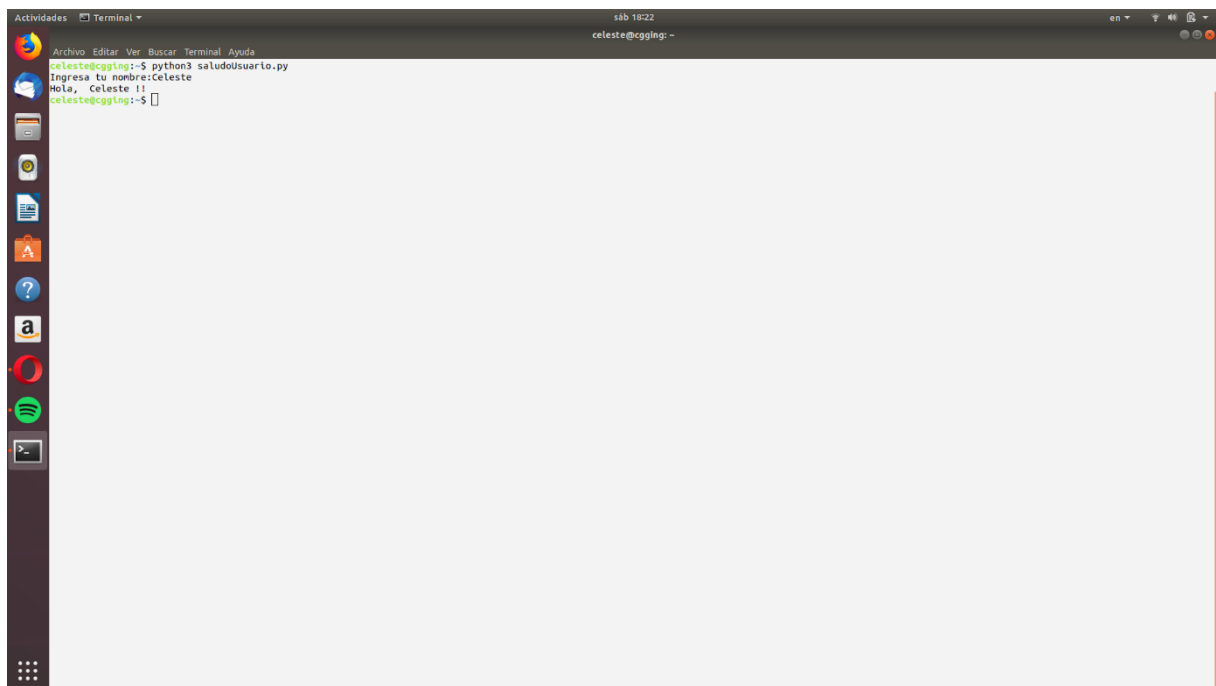
```
nombre= input("Ingresa tu nombre:")  
print("Hola, ",nombre,"!!")
```

Al ejecutar el código anterior, en una primera etapa se espera la interacción del usuario:



A terminal window titled "Terminal" with a menu bar containing "Actividades", "Archivo", "Editar", "Ver", "Buscar", "Terminal", and "Ayuda". The window shows the command prompt "celeste@ggling: ~" and the command "python3 saludoUsuario.py". The prompt "Ingresar tu nombre:" is displayed, and the cursor is positioned at the end of the line.

Para luego completar la ejecución del código:



A terminal window titled "Terminal" with a menu bar containing "Actividades", "Archivo", "Editar", "Ver", "Buscar", "Terminal", and "Ayuda". The window shows the command prompt "celeste@ggling: ~" and the command "python3 saludoUsuario.py". The prompt "Ingresar tu nombre:" is displayed, and the user has entered "Celeste". The output "Hola, Celeste !!\n" is displayed, and the prompt "celeste@ggling: ~" is shown again.

Errores comunes a la hora de programar

Con los ejemplos que hemos realizado hasta ahora, seguro que hemos comprobado que cometer errores a la hora de programar es algo muy normal. Algunos errores frecuentes en Python radican en olvidarnos los ":" luego de un if o un else o al comienzo de algún otro bloque. Equivocarnos u omitir la indentación en las líneas de un mismo bloque, escribir mal alguna palabra reservada, tal puede ser el caso de querer inicializar un booleano escribiendo 'true' en lugar de 'True', omitir las comillas en la función print, entre muchas otras.

Pero no solo podemos encontrar errores de sintaxis en Python, también existen otros errores, conocimos como excepciones y que se producen en tiempo de ejecución, por ejemplo, ingresar una letra en donde se espera un número. Este tipo de errores pueden preverse y manejarse en la programación.

El manejo de excepciones es algo de suma importancia para asegurar el funcionamiento de un código y será un tema que explayaremos en el próximo libro.

Suma con vitaminas

En el ejemplo anterior vimos cómo ingresar una cadena de texto por teclado, lo cual podría despertar el interrogante sobre si es posible ingresar números por teclado. Frente a este interrogante, la respuesta es positiva. Todos los datos que ingresamos por teclado utilizando la función input() se interpretan como cadenas de texto, pero es posible convertir estos datos utilizando ciertos modificadores.

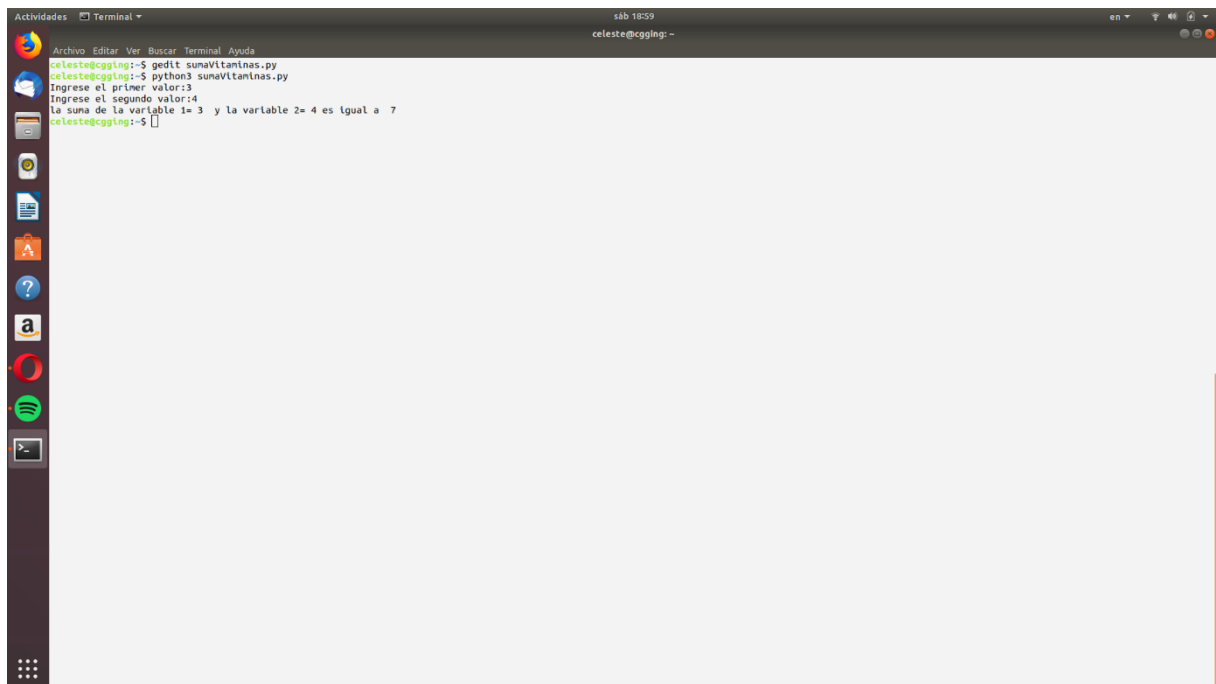
Para modificar el ejemplo de Suma pidiendo el ingreso de datos por teclado, basta con agregar el modificador int() encerrando a input() y de esta forma forzaremos a utilizar sólo datos numéricos enteros, si quisiéramos utilizar flotantes, bastaría con reemplazar int() con float(). ¿Qué ocurrirá si el usuario intentase ingresar una letra por teclado cuando el programa espera un dato numérico?, simplemente se producirá un error en tiempo de ejecución y no se continuará con el flujo del programa, esto mismo pasaría si no hiciéramos la conversión de datos en el input(), aún ingresando un dato numérico, pero veamos todo esto por partes:

El código con la modificación para pedir el ingreso de datos por teclado quedaría de la siguiente manera:

```
variable1=int(input("Ingrese el primer valor:"))  
variable2=int(input("Ingrese el segundo valor:"))  
suma= variable1 + variable2  
print("la suma de la variable 1=" , variable1, " y la variable 2=", variable2, "es igual a ",  
suma)
```

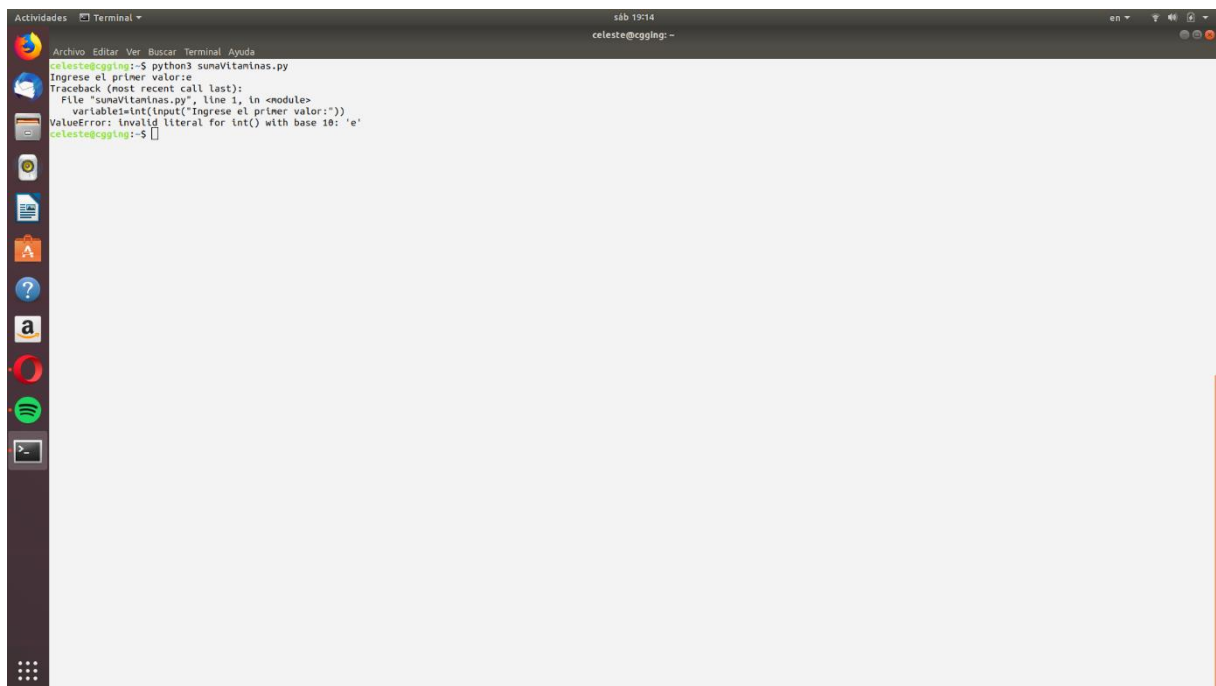
Con la modificación anterior, al ejecutar el programa, y si cargamos datos numéricos, todo funcionara sin problemas:





```
Actividades Terminal
sáb 18:39
celeste@cggling: ~
celeste@cggling:~$ gedit sumaVitaminas.py
celeste@cggling:~$ python3 sumaVitaminas.py
Ingrese el primer valor:3
Ingrese el segundo valor:4
La suma de la variable 1= 3 y la variable 2= 4 es igual a 7
celeste@cggling:~$
```

probemos lo que ocurriría si en vez de ingresar datos numéricos decidimos ingresar una letra:



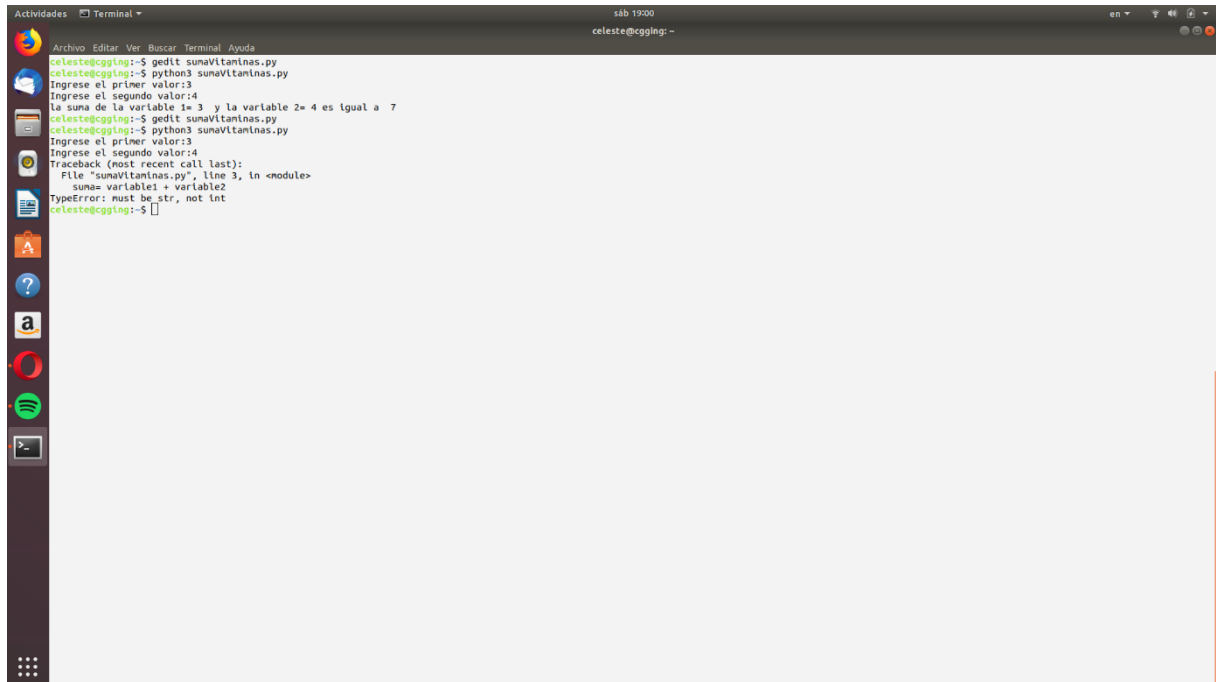
```
Actividades Terminal
sáb 19:14
celeste@cggling: ~
celeste@cggling:~$ python3 sumaVitaminas.py
Ingrese el primer valor:e
Traceback (most recent call last):
  File "sumaVitaminas.py", line 1, in <module>
    variables=int(input("Ingrese el primer valor:"))
ValueError: invalid literal for int() with base 10: 'e'
celeste@cggling:~$
```

cómo podemos ver en la figura anterior, al momento de intentar procesar la letra ‘e’ como un número, se produce un error en tiempo de ejecución y se da por terminado el flujo del programa.

Veamos ahora que sucede si modificamos el código anterior y no realizamos la conversión a entero en alguna de las variables que queremos cargar:

```
variable1=input("Ingrese el primer valor:")
variable2=int(input("Ingrese el segundo valor:"))
suma= variable1 + variable2
print("la suma de la variable 1=" , variable1, " y la variable 2=", variable2, "es igual a ",
suma)
```

Notar que hemos quitado el int(...) al ingresar los datos de la primer variable. Veamos que ocurre en ejecución:



```
celeste@cggling:~$ gedit sumaVitaminas.py
celeste@cggling:~$ python3 sumaVitaminas.py
Ingrese el primer valor:3
Ingrese el segundo valor:4
la suma de la variable 1= 3 y la variable 2= 4 es igual a 7
celeste@cggling:~$ gedit sumaVitaminas.py
celeste@cggling:~$ python3 sumaVitaminas.py
Ingrese el primer valor:3
Ingrese el segundo valor:4
Traceback (most recent call last):
  File "sumaVitaminas.py", line 3, in <module>
    suma= variable1 + variable2
TypeError: must be str, not int
celeste@cggling:~$
```

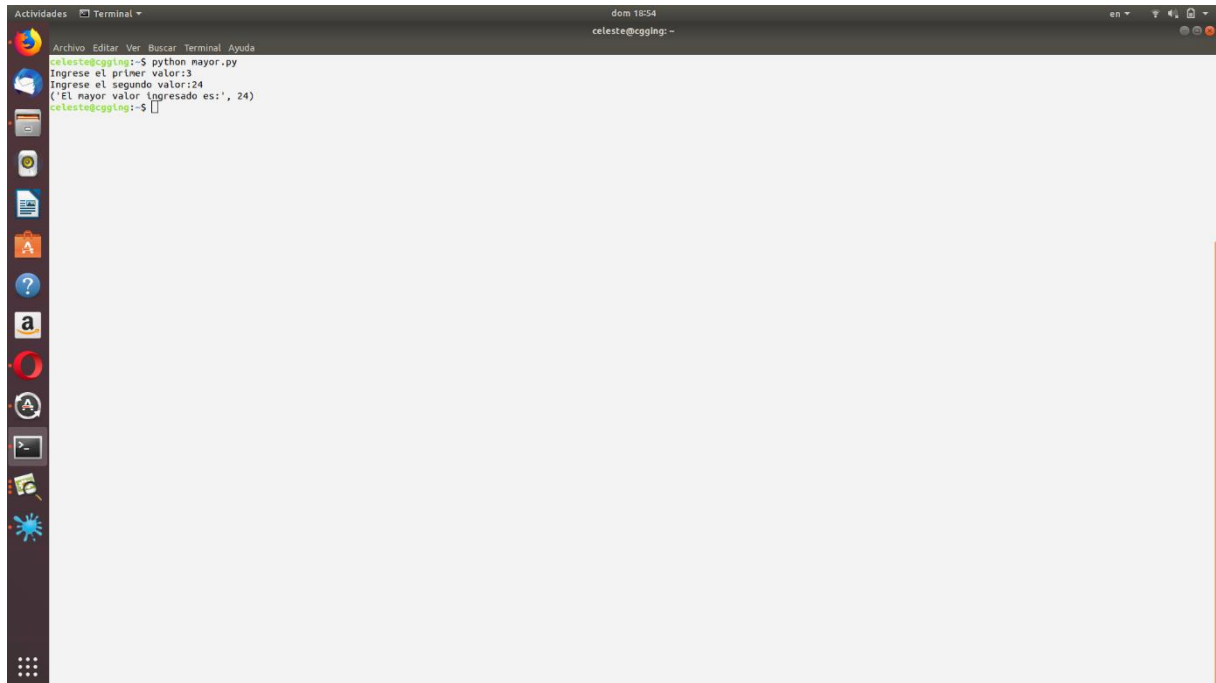
A pesar de haber ingresado dos datos numéricos, el intérprete no logra procesar el primero ya que omitimos realizar la conversión a entero y por ende no interpreta al valor entero 3, sino su representación en ASCII y por lo que entiende que es un caracter y no realmente el valor 3. Por este motivo no logra resolver la suma generando un error en tiempo de ejecución.

Obtener el mayor

Ahora supongamos que, en vez de sumar los dos valores ingresados por teclado, queremos simplemente quedarnos con el mayor de los dos. El código para dicha tarea seria:

```
variable1=int(input("Ingrese el primer valor:")) #Se pide el ingreso del primer valor
variable2=int(input("Ingrese el segundo valor:")) #Se pide el ingreso del 2do valor
if (variable1>variable2):
    mayor=variable1           #indentacion necesaria para marcar el bloque if
else:
    mayor=variable2           #indentacion necesaria para marcar el bloque else
print("El mayor valor ingresado es:",mayor)
```


Y al ejecutar el programa anterior,



```
Actividades Terminal
celest@cggle: ~
celest@cggle:~$ python mayor.py
Ingrese el primer valor:3
Ingrese el segundo valor:24
('El mayor valor ingresado es:', 24)
celest@cggle:~$
```

Pertenece al rango

Supongamos que tenemos un valor, y queremos saber si está en un cierto rango para luego realizar alguna tarea:

```
numero=5
if(numero>0 and numero<10):    #Es posible unir condiciones con operadores logicos
    print("El numero pertenece al rango")
else:
    print("El numero no pertenece al rango")
```

En el código anterior solo chequeamos si el valor pertenecía a un rango y mostramos por pantalla el mensaje indicando si se cumplía o no la condición.

Ahora podríamos modificar el código anterior para chequear si el valor se encuentra en alguno de tres rangos predeterminados y en caso de que así sea que se nos muestre por pantalla a que rango pertenece:

```

numero=5
if(numero>0 and numero<10):    #Es posible unir condiciones con operadores logicos
    print("El numero se encuentra entre 0 y 10")
elif(numero>10 and numero<20):
    print("El numero se encuentra entre 10 y 20")
elif(numero>20 and numero<30):
    print("El numero se encuentra entre 20 y 30")
else:
    print("El numero no pertenece al rango")

```

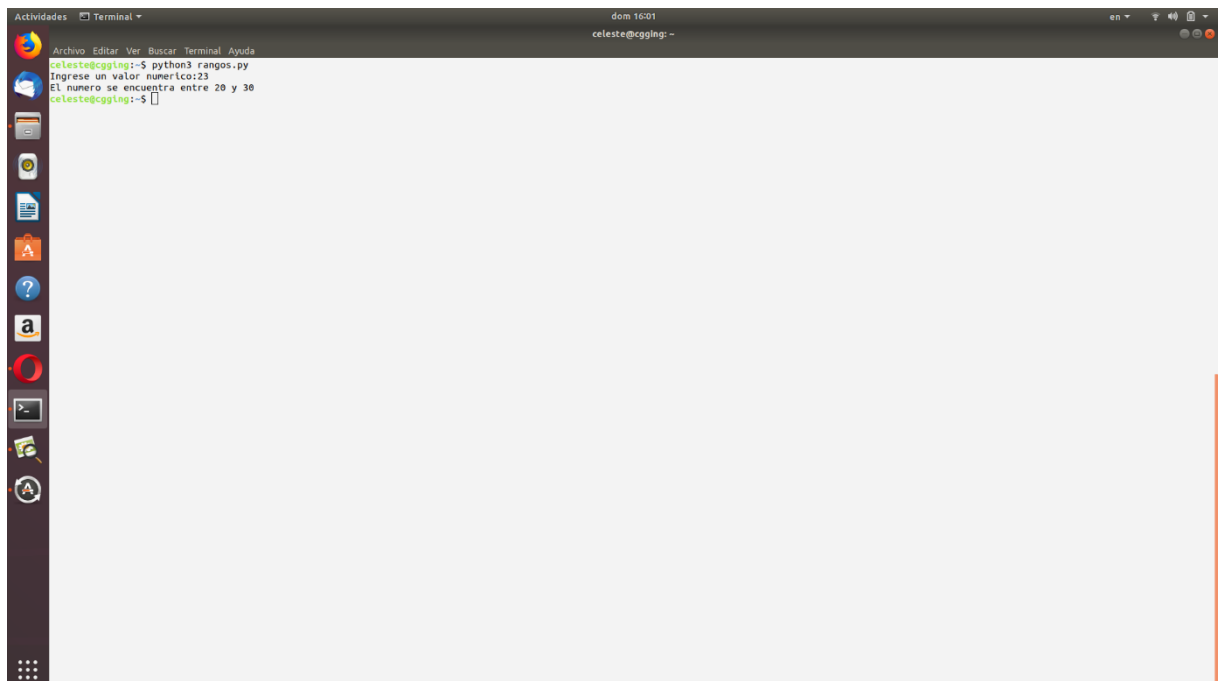
En los ejemplos anteriores, el valor fue establecido directamente desde el propio código, podemos modificar el código anterior para pedir el ingreso del número al usuario:

```

numero=int(input("Ingrese un valor numerico:"))
if(numero>0 and numero<10):    #Es posible unir condiciones con operadores logicos
    print("El numero se encuentra entre 0 y 10")
elif(numero>10 and numero<20):
    print("El numero se encuentra entre 10 y 20")
elif(numero>20 and numero<30):
    print("El numero se encuentra entre 20 y 30")
else:
    print("El numero no pertenece al rango")

```

al ejecutar esta última versión de nuestro código:



```

dom 16:01
celeste@cggling: ~
celeste@cggling:~$ python3 rangos.py
Ingrese un valor numerico:23
El numero se encuentra entre 20 y 30
celeste@cggling:~$

```

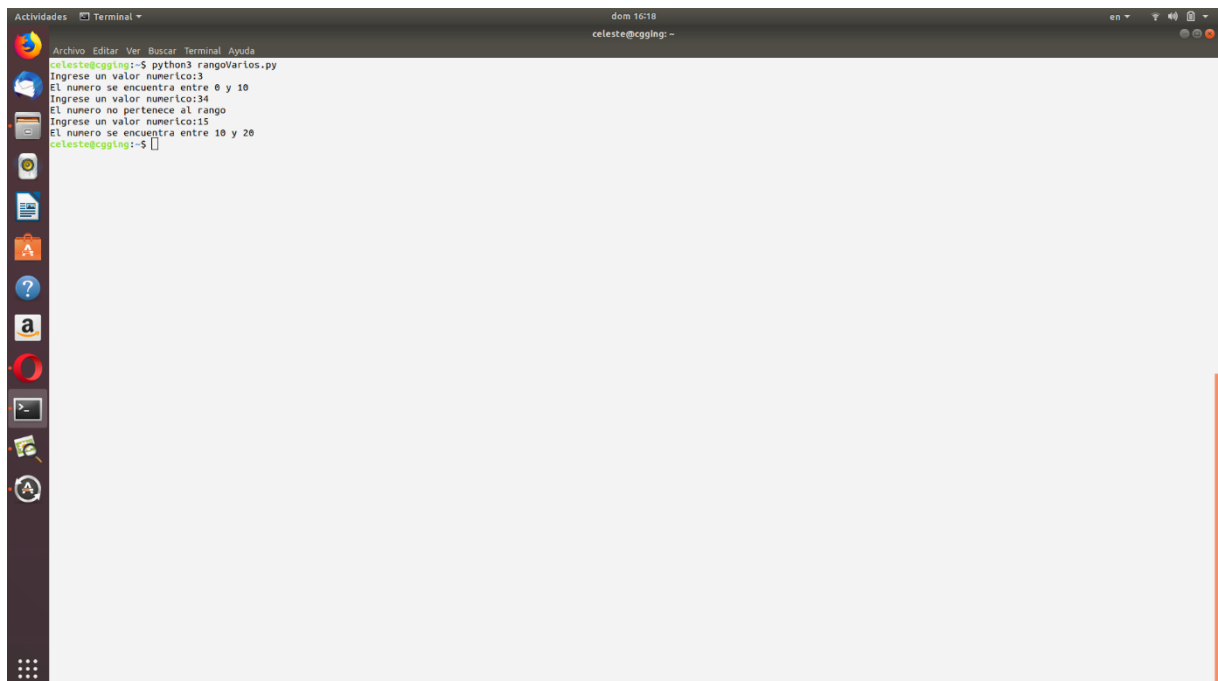
Por ultimo y para ya finalizar con este ejercicio de ejemplo, podemos agregar un bucle para chequear más de un valor cada vez que ejecutamos el programa:

```

i=3
while(i>0):
    numero=int(input("Ingrese un valor numerico:"))
    if(numero>0 and numero<10):
        print("El numero se encuentra entre 0 y 10")
    elif(numero>10 and numero<20):
        print("El numero se encuentra entre 10 y 20")
    elif(numero>20 and numero<30):
        print("El numero se encuentra entre 20 y 30")
    else:
        print("El numero no pertenece al rango")
    i=i-1

```

El código anterior nos pedirá el ingreso de un valor, nos dirá en que rango se encuentra, luego nos pedirá que ingresemos otro valor y así se repetirá el proceso 3 veces.



```

dom 16:18
celestee@cgging: ~
celestee@cgging:~$ python3 rangoVarios.py
Ingrese un valor numerico:3
El numero se encuentra entre 0 y 10
Ingrese un valor numerico:34
El numero no pertenece al rango
Ingrese un valor numerico:15
El numero se encuentra entre 10 y 20
celestee@cgging:~$

```

Mostrar los números del 1 al 100: While vs For

En el ejemplo anterior utilizamos un bucle while para repetir la ejecución del programa 3 veces, ahora queremos mostrar por pantalla los valores desde el 1 hasta el 100:

Utilizando while:

```
i = 1
while( i<=100 ):
    print(i)
    i+=1
print("Fin del bucle")
```

utilizando for:

```
for i in range(1,101):
    print(i)
```

La primera diferencia que podemos notar en este código en particular es que utilizando for podemos resolver el código en una cantidad de líneas menor que utilizando un while. No obstante, ambas opciones son válidas para resolver este ejercicio.

Mostrar los números pares entre 1 y 100

```
#1º forma
print("1 forma")
for i in range(1,101):
    if( (i%2)==0 ):
        print(i)
```

```
print("")
```

```
#2º forma
print("2 forma")
for i in range(2,101,2):
    print(i)
```

Jugando con rangos

Comenzaremos generando un rango de 10 números entre 0 y 10:

```
rango = list( range(10) )
print(rango)
```

Ahora acotaremos el rango a 5 valores entre 5 y 10:

```
rango = list(range(5,10))
print(rango)
```

También podemos generar un rango con números decrecientes:

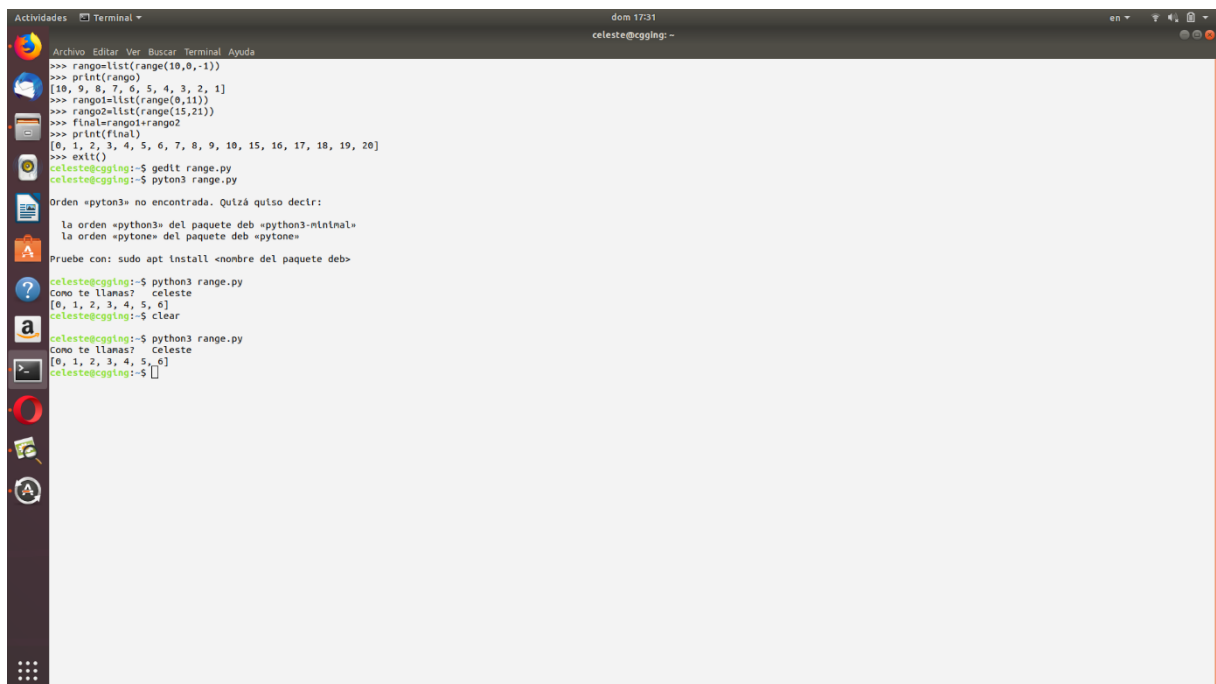
```
rango = list(range(10,0,-1))
print(rango)
```

O generar dos rangos separados y luego concatenarlos:

```
rango1 = list(range(0,11))
rango2 = list(range(15,21))
final = rango1 + rango2
print(final)
```

Lo que vamos a hacer a continuación es solicitarle el nombre al usuario y luego generar un rango desde 0 hasta la longitud de su nombre:

```
nombre=input("Como te llamas? ")
rango = list( range(0, len(nombre)))
print(rango)
```



```
dom 17:31
celestecg@cgling: ~$
Archivo Editar Ver Buscar Terminal Ayuda
>>> rango=list(range(10,0,-1))
>>> print(rango)
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> rango1=list(range(0,11))
>>> rango2=list(range(15,21))
>>> final=rango1+rango2
>>> print(final)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 16, 17, 18, 19, 20]
>>> exit()
celestecg@cgling:~$ gedit range.py
celestecg@cgling:~$ python3 range.py
Orden «python3» no encontrada. Quizá quisio decir:
  la orden «python3» del paquete deb «python3-minimal»
  la orden «pytone» del paquete deb «pytone»
Pruebe con: sudo apt install «nombre del paquete deb»
celestecg@cgling:~$ python3 range.py
Como te llamas? celeste
[0, 1, 2, 3, 4, 5, 6]
celestecg@cgling:~$ clear
celestecg@cgling:~$ python3 range.py
Como te llamas? Celeste
[0, 1, 2, 3, 4, 5, 6]
celestecg@cgling:~$
```

Cadenas de caracteres

Vamos a comenzar pidiendo al usuario dos cadenas de caracteres por teclado, luego intercambiaremos los primeros caracteres en ambas cadenas y las mostraremos concatenadas por pantalla con un espacio entre ambas:

```
cadena1 = input("Dame la primera cadena: ")
cadena2 = input("Dame la segunda cadena: ")
print( cadena2[2:] + cadena1[2:] + " " + cadena1[:2] + cadena2[:2] )
```

Para seguir, pediremos al usuario que ingrese una cadena de caracteres por teclado e informaremos al usuario si la cadena que ingreso es un palíndromo:

```
cadena1 = input("Dame una cadena: ")
cadena_al_reves = cadena1[::-1]
print(cadena_al_reves)
if( cadena1 == cadena_al_reves ):
    print("Es palindromo")
else:
    print("No es palindromo")
```

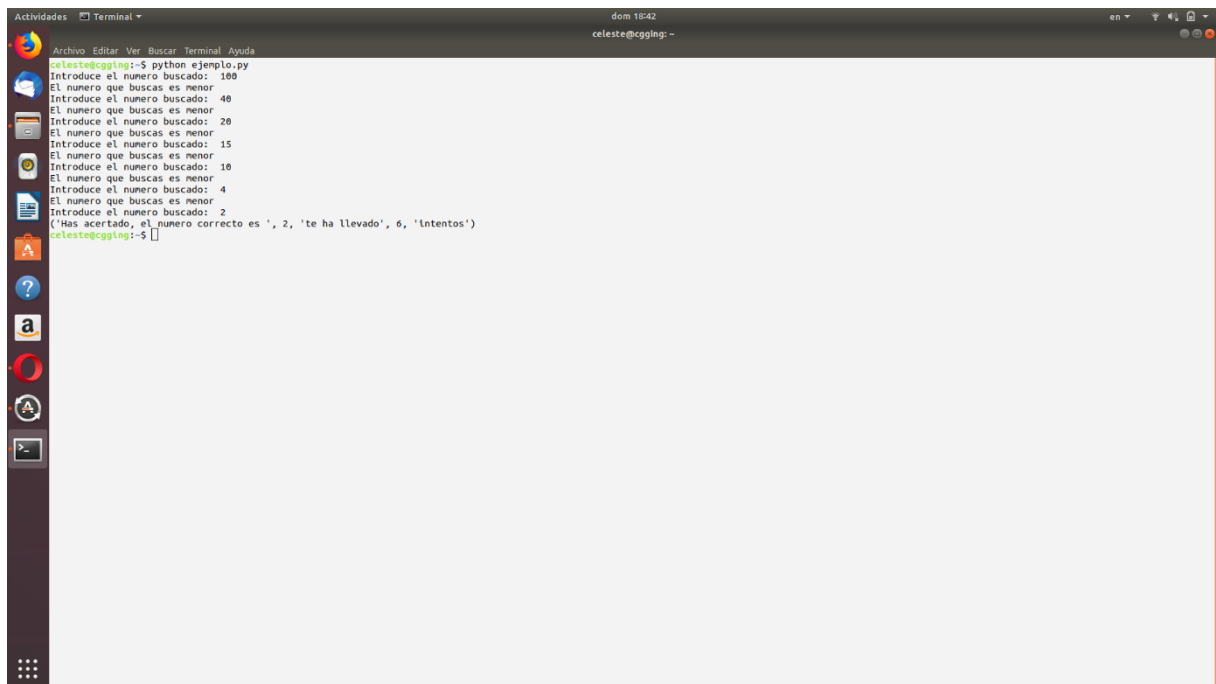
Ejemplo final: integrando todo lo visto y algo mas

El código que presentaremos a continuación genera aleatoriamente un valor y espera que el usuario sea capaz de adivinarlo, en cada intento erróneo se le indica al usuario si el valor que está buscando es mayor o menor al que ingresó y una vez acertado se le informa la cantidad de intentos que requirió para adivinar el numero:

```
from random import *
def generaNumeroAleatorio(minimo, maximo):
    return randint(minimo, maximo)

numero_buscado=generaNumeroAleatorio(1,100)
encontrado=False
intentos=0

while not encontrado:
    numero_usuario=int(input("Introduce el numero buscado: "))
    if numero_usuario>numero_buscado:
        print("El numero que buscas es menor")
        intentos=intentos+1
    elif numero_usuario<numero_buscado:
        print("El numero que buscas es mayor")
        intentos=intentos+1
    else:
        encontrado=True
        print("Has acertado, el numero correcto es ", numero_usuario, "te ha llevado",
            intentos, "intentos")
```



```
dom 18:42
celestecg@cel: ~
celestecg@cel:~$ python ejemplo.py
Introduce el numero buscado: 100
El numero que buscas es menor
Introduce el numero buscado: 40
El numero que buscas es menor
Introduce el numero buscado: 20
El numero que buscas es menor
Introduce el numero buscado: 15
El numero que buscas es menor
Introduce el numero buscado: 10
El numero que buscas es menor
Introduce el numero buscado: 4
El numero que buscas es menor
Introduce el numero buscado: 2
'Has acertado, el numero correcto es 2, te ha llevado 6, Intentos'
celestecg@cel:~$
```

En este ejemplo sumamos varias cosas interesantes: para empezar, importamos un módulo, además sumamos el uso de funciones.

Comencemos analizando el porqué del módulo. Dijimos que este código debe generar un número aleatorio, para eso Python cuenta con un módulo llamado Random que cuenta con diferentes funciones para crear y manejar números aleatorios.

Para poder utilizar los elementos de un módulo en un código que queramos realizar, antes que nada, debemos importar el módulo, por ese motivo para poder utilizar la función randint() en el código anterior, necesariamente necesitamos importar el módulo random.

Lo siguiente que vemos en este código es la definición de una función. Veamos a continuación como crear una función propia:

Funciones

Una función, es la forma de agrupar expresiones y sentencias (algoritmos) que realicen determinadas acciones, pero que éstas, solo se ejecuten cuando son llamadas. Es decir, que, al colocar un algoritmo dentro de una función, al correr el archivo, el algoritmo no será ejecutado si no se ha hecho una referencia a la función que lo contiene.

En Python, la definición de funciones se realiza mediante la instrucción `def` más un nombre de función descriptivo -para el cuál, aplican las mismas reglas que para el nombre de las variables- seguido de paréntesis de apertura y cierre. Como toda estructura de control en Python, la definición de la función finaliza con dos puntos (`:`) y el algoritmo que la compone, irá indentado con 4 espacios:

```
def mi_funcion():  
    # aquí el algoritmo
```

Una función, no es ejecutada hasta tanto no sea invocada. Para invocar una función, simplemente se la llama por su nombre:

```
def mi_funcion():  
    print "Hola Mundo"
```

```
funcion()
```

Cuando una función, haga un retorno de datos, éstos, pueden ser asignados a una variable:

```
def funcion():  
    return "Hola Mundo"
```

```
frase = funcion()  
print frase
```

Una función, puede tener cualquier tipo de algoritmo y cualquier cantidad de ellos y, utilizar cualquiera de las características vistas hasta ahora. No obstante, ello, una buena práctica, indica que la finalidad de una función, debe ser realizar una única acción, reutilizable y por lo tanto, tan genérica como sea posible.

Para ampliar un poco más este concepto, veamos el siguiente video:

https://youtu.be/VY448UWAQ_0

Lo que aprendimos en esta clase

En esta clase hemos realizado una serie de códigos de ejemplo a fin de comenzar a practicar la implementación de todo lo que hemos visto a lo largo de este curso.