



Unidad 4

Manejo de arreglos y funciones en Python

Excepciones

Ocurren cuando algo sale mal, debido a código incorrecto o entradas incorrectas.

Cuando ocurre una excepción, el programa se detiene inmediatamente.

```
num1 = 7  
num2 = 0  
print(num1/num2)
```

Traceback (most recent call last):

File "/home/asantana/excepciones.py", line 3, in <module>

print(num1/num2)

ZeroDivisionError: integer division or modulo by zero

Excepciones

Excepciones comunes:

ImportError: falla de importación.

IndexError: un lista es indexada con un número fuera de rango.

NameError: una variable desconocida es utilizada.

SyntaxError: el código no puede ser analizado correctamente.

TypeError: una función es llamada con un valor de un tipo inapropiado.

ValueError: una función es llamada con un valor de tipo correcto pero con un valor incorrecto.

Excepciones

¿Cuál excepción es levantada por este código?

```
print("7" + 4)
```

- ☐ ZeroDivisionError
- ☐ TypeError
- ☐ ValueError

Excepciones

¿Cuál excepción es levantada por este código?

```
print("7" + 4)
```

- ☐ ZeroDivisionError
- ☒ TypeError
- ☐ ValueError

Manejo de excepciones

Para manejar excepciones y ejecutar código cuando ocurre una excepción, se puede utilizar la sentencia **try/except**.

El bloque **try** contiene el código que puede lanzar una excepción.

Si ocurre un excepción el código del bloque **try** deja de ser ejecutado y el código del bloque **except** se ejecuta.

Si no ocurre ningún error el código del bloque **except** no se ejecuta.

Manejo de excepciones

```
try:  
    num1 = 7  
    num2 = 0  
    print (num1/num2)  
    print ("División correcta")  
except ZeroDivisionError:  
    print ('Ha ocurrido un error')  
    print ('debido a una división por cero')
```

```
>>>
```


```
Ha ocurrido un error  
debido a una división por cero
```

```
>>>
```

Manejo de excepciones

```
try:  
    num1 = 7  
    num2 = 0  
    print (num1/num2)  
    print ("División correcta")  
except ZeroDivisionError:  
    print ('Ha ocurrido un error')  
    print ('debido a una división por cero')
```

Se define que tipo de excepción se va a manejar



```
>>>
```

```
Ha ocurrido un error  
debido a una división por cero
```

```
>>>
```


Manejo de excepciones

¿Cuál es la salida de este código?

```
try:
```

```
    variable = 10
```

```
    print (10.0/2)
```

```
except ZeroDivisionError:
```

```
    print ("Error")
```

```
print ("Fin de programa")
```

- ☐ Error
Fin de programa
- ☐ 5.0
Fin de programa
- ☐ 5.0

Manejo de excepciones

¿Cuál es la salida de este código?

try:

variable = 10

print (10.0/2)

except ZeroDivisionError:

print ("Error")

print ("Fin de programa")

- ☐ Error
Fin de programa
- ☒ 5.0
Fin de programa
- ☐ 5.0

Manejo de excepciones

Una sentencia **try** puede tener varios bloques de **except** para manejar diferentes excepciones.

Varias excepciones puede ser colocadas en un solo bloque **except** utilizando paréntesis, para tener un solo bloque **except** que las maneje a todas.

```
try:
    variable=10
    print(variable + "Hola")
    print (variable/2)
except ZeroDivisionError:
    print ("División por cero")
except (ValueError, TypeError):
    print ("Ocurrió un error")
```

```
>>>
Ocurrió un error
>>>
```

Manejo de excepciones

¿Cuál es la salida de este código?

try:

 estudiantes = 42

 print (estudiantes/0)

 print (“Los estudiantes de Bioingeniería”)

except (ValueError, TypeError):

 print (“Ocurrió un ValueError o TypeError”)

except ZeroDivisionError:

 print (“División por cero”)

- ☐ División por cero
- ☐ Ocurrió un ValueError o TypeError
- ☐ División por cero
Ocurrió un ValueError o TypeError

Manejo de excepciones

¿Cuál es la salida de este código?

try:

```
    estudiantes = 42
```

```
    print (estudiantes/0)
```

```
    print (“Los estudiantes de Bioingeniería”)
```

except (ValueError, TypeError):

```
    print (“Ocurrió un ValueError o TypeError”)
```

except ZeroDivisionError:

```
    print (“División por cero”)
```



División por cero



Ocurrió un ValueError o TypeError



División por cero

Ocurrió un ValueError o TypeError

Manejo de excepciones

Una sentencia **except** sin ninguna excepción especificada atrapa todos los errores.

Se debe tener cuidado con esto porque se pueden atrapar errores inesperados y esconder errores de programación.

```
try:
    palabra="Bioingeniería"
    print(palabra/0)
except:
    print ("Ocurrió un error")
```

```
>>>
Ocurrió un error
>>>
```

Nota: El manejo de excepciones es particularmente útil cuando se trata de entradas de usuarios.

Excepciones

```
varo = True
while var:
    try:
        num1 = input('Ingrese el primer número: ')
        num2 = input('Ingrese el segundo número: ')
        print (num1/num2)
        var=False
    except TypeError:
        print('Ingresó un dato incorrecto. Intente de nuevo')
    except NameError:
        print('Por favor ingrese la cadena entre comillas.
              Intente nuevamente')
```

Calculadora

Crear una calculadora sencilla en Python, que permita escoger una de las cuatro operación aritméticas básicas e imprimir el resultado. En menú debe ser como se muestra a continuación.

1. Suma
2. Resta
3. Multiplicación
4. División
5. Salir

Condiciones:

* Que el algoritmo valide el ingreso de los datos, es decir, si ingresan caracteres en vez de número o si el ingreso de los caracteres es sin las comillas, adicional que cree la excepción para la división por cero



Complemento

Manejo de archivos

Abriendo archivos

Se puede utilizar Python para leer y escribir el contenido de **archivos**.

Los archivos de texto son los más fáciles de manipular.

Antes de que el archivo pueda ser editado, debe ser abierto con la función **open**.

```
mi_archivo = open("archivo_python.txt")
```

El argumento de la función **open** es la **ruta** al archivo. Si el archivo está en el mismo directorio del programa, basta solo con el nombre del archivo

Abriendo archivos

Se puede especificar el **modo** utilizado para abrir un archivo al pasar un segundo argumento a la función **open**. Así:

“**r**” : significa modo lectura. (predeterminado)

“**w**” : significa modo de escritura, para reescribir el contenido de un archivo.

“**a**” : significa modo de anexo, para agregar nuevo contenido al final de un archivo.

“**b**” : lo abre en modo **binario**, se utiliza para archivos que no son texto. (tales como imágenes y sonidos)

Abriendo archivos

#Modo escritura

```
open("archivo_python.txt","w")
```

#Modo lectura

```
open("archivo_python.txt","r")
```

```
open("archivo_python.txt")
```

#Modo escritura binaria

```
open("archivo_python.txt","b")
```

Abriendo archivos

Una vez que el archivo haya sido abierto y utilizado, se debe cerrar.

Esto se logra con el método **close** de un objeto archivo.

```
file = open("archivo_python.txt","w")  
#Hacer algo con el archivo  
file.close()
```

Abriendo archivos

¿Cómo se cerraría un archivo almacenado en la variable “text_file”?

- ☐ `close(“text_file”)`
- ☐ `text_file.close()`
- ☐ `close(text_file)`

Abriendo archivos

¿Cómo se cerraría un archivo almacenado en la variable “text_file”?

- ☐ `close(“text_file”)`
- ☒ `text_file.close()`
- ☐ `close(text_file)`

Leyendo archivos

Los contenidos de un archivo que ha sido abierto en modo texto puede ser leído utilizando el método **read**.

```
archivo = open("archivo_python.txt","r")
contenido = archivo.read()
print (contenido)
archivo.close()
```


Leyendo archivos

Luego que todo el contenido de un archivo haya sido leído, cualquier intento de leer más de ese archivo devolverá una cadena vacía, se está tratando de leer desde el final del archivo.

```
archivo = open("archivo_python.txt","r")
contenido1=archivo.read()
contenido2=archivo.read()
print (contenido1)
print ("Re-leyendo")
print (contenido2)
print ("Fin de programa")
archivo.close()
```

Leyendo archivos

Rellene los espacios en blanco para abrir un archivo, leer su contenido e imprimir su longitud.

```
archivo = _____ ("nombre_archivo.txt", "r")  
cadena = archivo._____  
print (len(cadena))  
archivo.close()
```

Leyendo archivos

Rellene los espacios en blanco para abrir un archivo, leer su contenido e imprimir su longitud.

```
archivo = open ("nombre_archivo.txt","r")  
cadena = archivo.read()  
print (len(cadena))  
archivo.close()
```

Leyendo archivos

Para obtener cada línea de un archivo, se puede utilizar la función **readlines** para devolver una lista donde cada elemento es una línea del archivo

```
archivo = open("archivo_python.txt","r")
print (archivo.readlines())
archivo.close()
```

```
>>>
```

```
['Estudiantes\n', 'de\n', 'la\n', 'Universidad\n', 'de\n', 'Antioquia\n']
```

```
>>>
```

Leyendo archivos

Se puede también utilizar un bucle **for** para iterar a través de las líneas del archivo.

```
archivo = open("archivo_python.txt","r")
for linea in archivo:
    print (linea)
archivo.close()
```

```
>>>
Estudiantes

de

la

Universidad

de

Antioquia

>>>
```

Escribiendo archivos

Para escribir archivos se utiliza el método **write**, el cual escribe una cadena en un archivo.

```
archivo = open("archivo_python2.txt","w")
archivo.write("Contenido nuevo")
archivo.close()
```

```
archivo = open("archivo_python2.txt","r")
print(archivo.read())
archivo.close()
```

```
>>>
```

```
Contenido nuevo
```

```
>>>
```

Nota: El modo “w” creará un archivo nuevo si no existe

Escribiendo archivos

¿Cuál línea de código escribe “Hola Mundo!” a un nuevo archivo?

- ☐ `write(file, “Hola Mundo!”)`
- ☐ `file.write(“Hola Mundo!”)`
- ☐ `write(“Hola Mundo!”,file)`

Escribiendo archivos

¿Cuál línea de código escribe “Hola Mundo!” a un nuevo archivo?

- ☐ `write(file, “Hola Mundo!”)`
- ☒ `file.write(“Hola Mundo!”)`
- ☐ `write(“Hola Mundo!”,file)`

Escribiendo archivos

Cuando un archivo es abierto en modo de escritura, el contenido existente del archivo es borrado.

```
archivo = open("archivo_python.txt","r")
print ("Lectura inicial del contenido del archivo")
print (archivo.read())
print ("Fin de lectura")
archivo.close()
```

```
archivo = open("archivo_python.txt","w")
archivo.write("Texto nuevo para el contenido")
archivo.close()
```

```
archivo = open("archivo_python.txt","r")
print ("Lectura con el nuevo contenido del archivo")
print (archivo.read())
print ("Fin de lectura")
archivo.close()
```

Escribiendo archivos

El método **write** también escribe en el archivo si se pasa una variable como argumento de la función.

```
msg = "Hola Mundo!"  
archivo = open("archivo_python.txt","w")  
archivo.write(msg)  
archivo.close()  
  
archivo = open("archivo_python.txt","r")  
print (archivo.read())  
archivo.close()
```

Actualizando archivos

Abrir el archivo con el modo “**a**”, a diferencia del modo “**w**”, el archivo de abre y el contenido que se incluya NO elimina el contenido actual, los anexa al final del archivo en cuestión.

```
msg = "Hola Mundo!"
archivo = open("archivo_python2.txt","w")
archivo.write(msg)
archivo.close()

archivo = open("archivo_python2.txt","r")
print (archivo.read())
archivo.close()

archivo = open("archivo_python2.txt","a")
archivo.write("\nContenido nuevo del archivo")
archivo.close()
```

Detectando archivos en un directorio

Importando el módulo **os** y utilizando el método **listdir**, se pueden hallar todos los archivos existentes en una carpeta, dicha lectura es devuelta en un lista.

```
import os
files = os.listdir('C:\manejoarchivos/archivos2')
print (files)
```

```
>>>
```

```
['f1.txt', 'f2.txt', 'f3.txt']
```

```
>>>
```

Detectando archivos en un directorio

Importando el módulo **os** y utilizando el método **listdir**, se pueden hallar todos los archivos existentes en una carpeta, dicha lectura es devuelta en un lista.

```
import os
files = os.listdir('C:\manejoarchivos/archivos2')
print (files)

for i in files:
    archivo = open('C:\manejoarchivos/archivos2/'+i, 'r')
    f3=archivo.readlines()
```