Inspira Crea Transforma

UNIVERSIDAD EAFIT®

Programación de Computadores

ST0240

Semestre 2017-1

48 horas semestral

Semana 6

ST0240

Semestre 2017-1

48 horas semestral

Fundamentos de Programación

ST0286

Semestre 2017-1

48 horas semestral

Agenda

2 clases por Semana

15m Contenido Previo (Tareas, Lecturas, Ejercicios propuestos o Anterior)

30m Contenido de Clase - Entrada y Salida

30m Ejercicios Prácticos

15m Cierre: Trabajo Individual (Por fuera de Clase)

1 clase por Semana

30 m Contenido Previo (Tareas, Lecturas, Ejercicios propuestos o Anterior)

50 m Contenido de Clase - Entrada y Salida

20 m Break

50 m Ejercicios Prácticos

30 m Cierre: Trabajo Individual (Por fuera de Clase)

Errores y Excepciones.

- Los errores de sintaxis son conocidos también como errores de interpretación.
- El intérprete se posiciona en la línea y posición de error.

```
>>> while True print('Hola mundo')
Traceback (most recent call last):
...
    while True print('Hola mundo')
                     ^
SyntaxError: invalid syntax
```

Errores y Excepciones.

- La instrucción o declaración puede llegar a ser sintácticamente correcta y en el momento de compilación puede generar un error. Este tipo de errores se denominan **excepciones.**

- Algunas de las excepciones no son administradas por el programa lo que se convierte en un mensaje de error directamente.

Errores y Excepciones.

- Ejemplo 1:

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: division by zero
```

Errores y Excepciones.

- Ejemplo 2:

```
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'spam' is not defined
```

Errores y Excepciones.

- Ejemplo 3:

```
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: Can't convert 'int' object to str implicitly
```

Errores y Excepciones.

- La línea final del mensaje nos indica que tipo de error ha surgido.
- Las excepciones son de diferentes tipos. Por lo tanto este tipo se imprime como parte del mensaje. Basados en los ejemplos 1 a 3 los mensajes son:

ZeroDivisionError

Name Error

TypeError

- La cadena mostrada como tipo de la excepción es el nombre de la excepción predefinida que ocurrió.

Errores y Excepciones.

- Los nombres de las excepciones estándar son identificadores incorporados al intérprete (no son palabras clave reservadas).

- Lo demás en la línea del mensaje muestra un detalle basado en el tipo de la excepción y por qué se ha generado.

Tipos de Excepciones.

- *exception* ***BaseException***:

→ Es la clase base para todas las excepciones integradas. Y no está configurada para utilizarse por clases definidas por el usuario (en ese caso debe utilizar exception).

→ Al llamarse str( ) a una instancia de esta clase, se devolverá la representación del argumento(s) a la instancia, o en su defecto la cadena vacía cuando no hay argumentos.

Tipos de Excepciones.

- *exception* ***BaseException***:

→ **args**.

Es la tupla de argumentos dada a quién “construye” las excepciones. Algunas excepciones integradas (como OSError) esperan un cierto número de argumentos y asignan un significado especial a los elementos de esta tupla, mientras que otros normalmente se llaman sólo con una sola cadena arrojando un mensaje de error..

Tipos de Excepciones.

- *exception* ***BaseException***:

→ **with_traceback(tb).**

Este método establece **tb** como el nuevo traceback para la excepción devolviendo el objeto de excepción. Por lo general se utiliza en el código de gestión de excepciones como el que se visualiza en la siguiente diapositiva:

Tipos de Excepciones.

- *exception* ***BaseException***:

→ **with_traceback(tb).**

```
try:
    ...
except SomeException:
    tb = sys.exc_info()[2]
raise OtherException(...).with_traceback(tb)
```

Tipos de Excepciones.

- *exception* ***Exception***:

→ Hace referencia a todas las excepciones integradas y que al no salir del sistema se derivan de esta clase. Todas las excepciones definidas por el usuario también deben derivarse de esta clase.

Tipos de Excepciones.

- *exception* ***ArithmeticError***:

→ Es la clase base para las excepciones integradas que se generan para varios errores aritméticos:

OverflowError

ZeroDivisionError

FloatingPointError.

Tipos de Excepciones.

- *exception* ***BufferError***:

→ Se omite cuando no se puede realizar una operación relacionada con el buffer.

Tipos de Excepciones.

- *exception* ***LookupError***:

→ Es la clase base para las excepciones que se generan cuando una clave o índice utilizado en una asignación o secuencia no es válida:

IndexError, **KeyError**.

Lo que puede plantearse directamente por:

codecs.lookup()

Algunas excepciones concretas.

- *exception* ***AssertionError***:

→ Se omite cuando falla una sentencia **assert.**

- *exception* ***AttributeError:***

→ Se omite cuando falla una referencia de atributo o asignación. (Cuando un objeto no admite referencias de atributos o asignaciones de atributo en absoluto, se genera el **TypeError**.)

Algunas excepciones concretas.

- *exception* ***EOFError***:

→ Se omite cuando la función **input( )** alcanza una condición de final de archivo (**EOF**) sin leer ningún dato. Un ejemplo de ello son los métodos **io.IOBase.read()** y **io.IOBase.readline()** son métodos que devuelven una cadena vacía cuando alcanzan (EOF).

Algunas excepciones concretas.

- *exception* ***FloatingPointError***:

→ Se omite cuando falla una operación de punto flotante. Esta excepción siempre se define, pero sólo se podrá generar cuando Python esté configurado con la opción **--with-fpectl** o el símbolo **WANT_SIGFPE_HANDLER** esté definido con el archivo **pyconfig.h.**

El Manejo de las Excepciones.

- Al programar un conjunto de instrucciones es posible manejar ciertas excepciones. Un ejemplo de ello se muestra a continuación:

>> un programa le pide al usuario una entrada hasta que ingrese un entero válido. Pero al interrumpir el programa se visualiza la excepción **KeyboardInterrupt**.

El Manejo de las Excepciones.

- Ejemplo 4:

```
>>> while True:
...     try:
...         x = int(input("Por favor ingrese un número: "))
...         break
...     except ValueError:
...         print("Oops! No era válido. Intente nuevamente...")
...
```

El Manejo de las Excepciones.

→ Funcionamiento de la declaración **try:**

1. Se deben ejecutar las instrucciones contenidas dentro del bloque **try** y **except.**
2. Al no generarse ninguna excepción el bloque **except** se omite y termina la ejecución mediante la declaración **try.**
3. Al generarse una excepción durante la ejecución del bloque **try,** el bloque restante se omite. Después, si el tipo coincide con la excepción nombrada (**except**), el bloque **except** se ejecutará, y la ejecución continuará después de la declaración **try.**

El Manejo de las Excepciones.

→ Funcionamiento de la declaración **try:**

4. Si se genera una excepción que no coincide con la excepción nombrada en el **except,** ésta continua a las declaraciones en **try** más adelante. De no encontrarse una instrucción que la maneje se denomina una ***excepción no manejada,*** de esta forma la ejecución parará arrojando un mensaje determinado.

El Manejo de las Excepciones.

→ Funcionamiento de la declaración **try:**

5. Es posible que una declaración **try** pueda tener más de un **except.** Esto con el fin de poder especificar manejadores para diferentes excepciones. Sólo operan excepciones generadas en **try** correspondiente, no en otros manejadores del mismo **try.** De esta forma un **except** podrá declarar múltiples excepciones por medio de paréntesis. Por ejemplo:

El Manejo de las Excepciones.

→ Funcionamiento de la declaración **try:**

- Ejemplo 5:

```
... except (RuntimeError, TypeError, NameError):
...     pass
```

El Manejo de las Excepciones.

→ Funcionamiento de la declaración **try:**

6. El último **except** nombrado puede omitir que excepción capturar.

7. También puede utilizarse para mostrar un mensaje de error y luego reutilizar la excepción.

Un ejemplo de esto se visualiza en la siguiente diapositiva:

El Manejo de las Excepciones.

Ejemplo 6:

```
import sys


try:
    f = open('miarchivo.txt')
    s = f.readline()
    i = int(s.strip())
except OSError as err:
    print("Error OS: {0}".format(err))
except ValueError:
    print("No pude convertir el dato a un entero.")
except:
    print("Error inesperado:", sys.exc_info()[0])
    raise
```

El Manejo de las Excepciones.

8. Las declaraciones try y except poseen un bloque else que es opcional. Que al nombrarse debe seguir a los except declarados. Será útil dentro del código que al ejecutarse, el try no genera ninguna excepción. Ej.7 :

```
for arg in sys.argv[1:]:
    try:
        f = open(arg, 'r')
    except IOError:
        print('no pude abrir', arg)
    else:
        print(arg, 'tiene', len(f.readlines()), 'lineas')
        f.close()
```

El Manejo de las Excepciones.

9. Se recomienda utilizar el **else** cuando sea necesario incluir código adicional en el bloque del **try**, ya que esto evitará capturar accidentalmente una excepción que no haya sido generada por el código ya protegido por la declaración **try** y **except**.

10. En algunas ocasiones cuando se genera una excepción, esta puede tener un valor asociado. A esto se le conoce como el **argumento de la excepción**. Significa que la presencia y el tipo de argumento dependen del tipo de excepción.

El Manejo de las Excepciones.

11. Un comando **except** puede especificar una variable luego del nombre de la excepción. Dicha variable se vincularía a una "instancia" de excepción basada en los argumentos almacenados en la instrucción **instance.args.** Esta instancia de excepción define **__str__( )**, esto con el fin de mostrar los argumentos directamente, sin necesidad de hacer referencia a **.args**

12. También es posible "instanciar" la excepción inicialmente, antes de generarla. De esta forma se podrán incluir los atributos deseados. Un ejemplo de ello se visualiza en la siguiente diapositiva:

El Manejo de las Excepciones.

- Ejemplo 8:

```
>>> try:
...     raise Exception('meteoroide', 'meteorito')
... except Exception as inst:
...     print(type(inst))    #la instancia de excepción
...     print(inst.args)     #argumentos guardados en .args
...     print(inst)    #str permite imprimir args directamente,
...                    #pero puede ser cambiado en subclases de la exc
...     x, y = inst          #desempacar argumentos
...     print('x =', x)
...     print('y =', y)
...
<class 'Exception'>
('meteoroide', 'meteorito')
('meteoroide', 'meteorito')
x = meteoroide
y = meteorito
```

El Manejo de las Excepciones.

13. Si una excepción posee argumentos, estos se imprimirán como la parte final del mensaje para las excepciones que no están “manejadas”.

14. Los diferentes “manejadores” de excepciones no solo manejan las excepciones ejecutadas en el bloque del **try,** también manejan las excepciones que se generan dentro de las funciones que se llaman (inclusive indirectamente) dentro del bloque del **try**. Un ejemplo de ello se visualiza en la siguiente diapositiva:

El Manejo de las Excepciones.

- Ejemplo 9:

```
>>> def esto_falla():
...     x = 1/0
...
>>> try:
...     esto_falla()
... except ZeroDivisionError as err:
...     print('Manejando error en tiempo de ejecución:', err)
...
Manejando error en tiempo de ejecución: int division or modulo by zero
```

Ejercicio para realizar en clase.

→ **Sumar una lista de números:**

Crear un programa que sume todos los números introducidos por el usuario mientras ignora las líneas introducidas que no son números válidos. Su programa debe mostrar la suma actual después de introducir cada número. Debe mostrar un mensaje de error apropiado después de cualquier entrada no válida, y luego continuar sumando los números adicionales introducidos por el usuario. Su programa debe salir cuando el usuario introduce una línea en blanco. Asegúrese de que su programa funciona correctamente tanto para números enteros como para los de punto flotante.

Nota 1: Este ejercicio requiere que utilice excepciones sin utilizar archivos.

Nota 2: Puede resolverse en 26 líneas de código.

Recursos Adicionales

https://spoj.com/

https://ideone.com/

https://www.pythonanywhere.com

http://python.swaroopch.com/

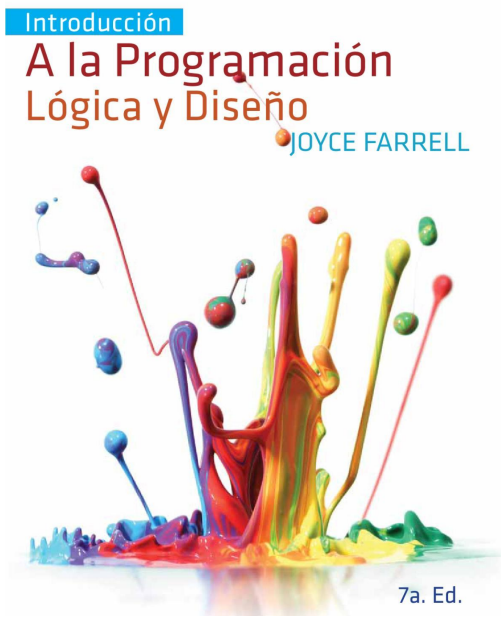
https://pragprog.com/book/gwpy2/practical-programming

https://coderbyte.com/course/learn-python-in-one-week

https://developers.google.com/edu/python/

Para la Próxima Clase/Semana

1. Leer Capítulos 4 y 5 del libro
2. Buscar el algoritmo para ubicación por GPS
3. Consultar cual es el método de triangulación.
4. Exponer problemas que hayan tenido en el foro de la materia.
5. Realizar las tareas dejadas en clase



Referencias y Lecturas Adicionales

- Python Google Class - https://developers.google.com/edu/python/
- Hacker rank - https://www.hackerrank.com/dashboard