**INSTITUTO SUPERIOR TECNOLÓGICO DEL SUR**

**CARRERA**

**DISEÑO Y PROGRAMACIÓN WEB**

**LENGUAJE WEB 2**

**“Estructuras de Control Especiales en Python”**

**PROFESOR(A) : Amado Cerpa Juan Andrés**

**ALUMNO : Vilca Apaza Christian**

**SEMESTRE : V**

25/10/2020

CONTENIDO

[FUNCIÓN TRY (MANEJO DE EXCEPCIONES) 3](#_Toc54549049)

[FUNCIÓN ZIP() DE PYTHON 7](#_Toc54549050)

[ENUMERATE() EN PYTHON 8](#_Toc54549051)

[PYTHON LIST COMPREHENSION 9](#_Toc54549052)

[ITERABLES 10](#_Toc54549053)

[ITERADORES 10](#_Toc54549054)

[BIBLIOGRAFÍA 12](#_Toc54549055)

# FUNCIÓN TRY (MANEJO DE EXCEPCIONES)

Es posible escribir programas que controlen las excepciones seleccionadas. Observe el siguiente ejemplo, que solicita al usuario la entrada hasta que se ha introducido un entero válido, pero permite al usuario interrumpir el programa (utilizando o lo que admita el sistema operativo); tenga en cuenta que una interrupción generada por el usuario se señala mediante la generación de la [excepción KeyboardInterrupt.](https://docs.python.org/3/library/exceptions.html#KeyboardInterrupt)Control-C

>>>

**>>> while** **True**:

**...**  **try**:

**...**  x = int(input("Please enter a number: "))

**...**  **break**

**...**  **except** ValueError:

**...**  print("Oops! That was no valid number. Try again...")

**...**

La instrucción [try](https://docs.python.org/3/reference/compound_stmts.html#try)funciona de la siguiente manera.

* En primer lugar, se ejecuta la *cláusula try* (las instrucciones entre las palabras clave [try](https://docs.python.org/3/reference/compound_stmts.html#try)y [except).](https://docs.python.org/3/reference/compound_stmts.html" \l "except)
* Si no se produce ninguna excepción, se omite la *cláusula except* y finaliza la ejecución de la instrucción [try.](https://docs.python.org/3/reference/compound_stmts.html#try)
* Si se produce una excepción durante la ejecución de la cláusula try, se omite el resto de la cláusula. A continuación, si su tipo coincide con la excepción con el nombre de la palabra clave [except,](https://docs.python.org/3/reference/compound_stmts.html" \l "except)se ejecuta la cláusula except y, a continuación, la ejecución continúa después de la instrucción [try.](https://docs.python.org/3/reference/compound_stmts.html#try)
* Si se produce una excepción que no coincide con la excepción nombrada en la cláusula except, se pasa a las instrucciones [try](https://docs.python.org/3/reference/compound_stmts.html#try)externas; si no se encuentra ningún controlador, es una *excepción no controlada* y la ejecución se detiene con un mensaje como se muestra anteriormente.

Una instrucción [try](https://docs.python.org/3/reference/compound_stmts.html#try)puede tener más de una cláusula except, para especificar controladores para diferentes excepciones. Como máximo, se ejecutará un controlador. Los controladores solo controlan las excepciones que se producen en la cláusula try correspondiente, no en otros controladores de la misma instrucción. Una cláusula except puede nombrar varias excepciones como una tupla entre paréntesis, por ejemplo:try

... **except** (RuntimeError, TypeError, NameError):

... **pass**

Una clase de una cláusula [except](https://docs.python.org/3/reference/compound_stmts.html" \l "except)es compatible con una excepción si es la misma clase o una clase base de la misma (pero no al revés— una cláusula except que enumera una clase derivada no es compatible con una clase base). Por ejemplo, el código siguiente imprimirá B, C, D en ese orden:

**class** **B**(Exception):

**pass**

**class** **C**(B):

**pass**

**class** **D**(C):

**pass**

**for** cls **in** [B, C, D]:

**try**:

**raise** cls()

**except** D:

print("D")

**except** C:

print("C")

**except** B:

print("B")

Tenga en cuenta que si las cláusulas except se invirtieron (con first), se habría impreso B, B, B — se desencadena la primera cláusula except coincidente.except B

La última cláusula except puede omitir los nombres de excepción, para que sirvan como comodín. ¡Utilice esto con extrema precaución, ya que es fácil enmascarar un error de programación real de esta manera! También se puede utilizar para imprimir un mensaje de error y, a continuación, volver a generar la excepción (permitiendo que un autor de la llamada controle la excepción también):

**import** **sys**

**try**:

f = open('myfile.txt')

s = f.readline()

i = int(s.strip())

**except** OSError **as** err:

print("OS error: *{0}*".format(err))

**except** ValueError:

print("Could not convert data to an integer.")

**except**:

print("Unexpected error:", sys.exc\_info()[0])

**raise**

El [try](https://docs.python.org/3/reference/compound_stmts.html#try)... [except](https://docs.python.org/3/reference/compound_stmts.html#except)la instrucción tiene una *cláusula else* opcional, que, cuando está presente, debe seguir todas excepto las cláusulas. Es útil para el código que se debe ejecutar si la cláusula try no genera una excepción. Por ejemplo:

**for** arg **in** sys.argv[1:]:

**try**:

f = open(arg, 'r')

**except** OSError:

print('cannot open', arg)

**else**:

print(arg, 'has', len(f.readlines()), 'lines')

f.close()

El uso de la cláusula es mejor que agregar código adicional a la cláusula [try](https://docs.python.org/3/reference/compound_stmts.html#try)porque evita detectar accidentalmente una excepción que no fue provocada por el código que está protegido por la ... Declaración. Else try except

Cuando se produce una excepción, puede tener un valor asociado, también conocido como *argumento* de la excepción. La presencia y el tipo del argumento dependen del tipo de excepción.

La cláusula except puede especificar una variable después del nombre de la excepción. La variable está enlazada a una instancia de excepción con los argumentos almacenados en. Para mayor comodidad, la instancia de excepción define [\_\_str\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__str__)para que los argumentos se puedan imprimir directamente sin tener que hacer referencia a . También se puede crear una instancia de una excepción primero antes de generarla y agregar cualquier atributo a ella como desee.instance.args.args

>>>

**>>> try**:

**...**  **raise** Exception('spam', 'eggs')

**... except** Exception **as** inst:

**...**  print(type(inst)) *# the exception instance*

**...**  print(inst.args) *# arguments stored in .args*

**...**  print(inst) *# \_\_str\_\_ allows args to be printed directly,*

**...**  *# but may be overridden in exception subclasses*

**...**  x, y = inst.args *# unpack args*

**...**  print('x =', x)

**...**  print('y =', y)

**...**

<class 'Exception'>

('spam', 'eggs')

('spam', 'eggs')

x = spam

y = eggs

Si una excepción tiene argumentos, se imprimen como la última parte ('detalle') del mensaje para las excepciones no controladas.

Los controladores de excepciones no solo controlan las excepciones si se producen inmediatamente en la cláusula try, sino también si se producen dentro de las funciones a las que se llama (incluso indirectamente) en la cláusula try. Por ejemplo:

>>>

**>>> def** this\_fails():

**...**  x = 1/0

**...**

**>>> try**:

**...**  this\_fails()

**... except** ZeroDivisionError **as** err:

**...**  print('Handling run-time error:', err)

**...**

Handling run-time error: division by zero

# FUNCIÓN ZIP() DE PYTHON

La función devuelve un objeto zip, que es un iterador de tuplas donde el primer elemento de cada iterador pasado se empareja entre sí y, a continuación, el segundo elemento de cada iterador pasado se empareja.

Si los iteradores pasados tienen longitudes diferentes, el iterador con los elementos mínimos decide la longitud del nuevo iterador.

Sintaxis

zip(iterator1, iterator2, iterator3 ...)

Ejemplo

a = ("John", "Charles", "Mike")

b = ("Jenny", "Christy", "Monica")

x = zip(a, b)

print(tuple(x))

Resultado

(('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica'))

# ENUMERATE() EN PYTHON

Muchas veces cuando se trata de iteradores, también tenemos la necesidad de mantener un recuento de iteraciones. Python facilita la tarea de los programadores proporcionando una función integrada enumerate() para esta tarea.  
El método Enumerate() agrega un contador a un iterable y lo devuelve en forma de objeto de enumeración. Este objeto de enumeración se puede utilizar directamente en bucles for o convertirse en una lista de tuplas utilizando el método list().

Sintaxis:

enumerate(iterable, start=0)

Ejemplo:

l1 = ["eat","sleep","repeat"]

s1 = "geek"

obj1 = enumerate(l1)

obj2 = enumerate(s1)

print "Return type:",type(obj1)

print list(enumerate(l1))

print list(enumerate(s1,2))

Resultado:

Return type: < type 'enumerate' >

[(0, 'eat'), (1, 'sleep'), (2, 'repeat')]

[(2, 'g'), (3, 'e'), (4, 'e'), (5, 'k')]

# PYTHON LIST COMPREHENSION

Las comprensiones de listas proporcionan una forma concisa de crear listas.

Consta de corchetes que contienen una expresión  
seguida de una cláusula for, a continuación, cero o más para o if cláusulas. Las expresiones pueden ser cualquier  
cosa, lo que significa que puede poner en todo tipo de objetos en listas.

El resultado será una nueva lista resultante de  
la evaluación de la expresión en el contexto de las cláusulas for y if que la siguen.

La comprensión de lista siempre devuelve una lista de resultados.

La sintaxis básica utiliza corchetes.

[ expression for item in list if conditional ]

Ejemplos

squares = []

for x in range(10):

squares.append(x\*\*2)

print squares

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

result:

squares = [x\*\*2 for x in range(10)]

print squares

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# ITERABLES

Nuestro encuentro con for-loops introdujo el término iterable - un objeto que se puede "iterar sobre", como en un bucle for.

Un iterable es cualquier objeto Python capaz de devolver sus miembros de uno en uno, lo que permite que se itera en un bucle for.

Ejemplos conocidos de iterables incluyen listas, tuplas y cadenas: cualquier secuencia de este tipo se puede iterar en un bucle for. También encontraremos importantes colecciones no secuenciales, como diccionarios y conjuntos; estos son iterables también. También es posible tener un iterable que "genera" cada uno de sus miembros en la iteración - lo que significa que nunca almacena todos sus miembros en la memoria a la vez. Dedicamos una sección completa a generadores, un tipo especial de iterador, porque son muy útiles para escribir código eficiente.

# ITERADORES

Un iterador es un objeto que permite recorrer uno a uno los elementos almacenados en una estructura de datos, y operar con ellos.

En particular, en Python, los iteradores tienen que implementar un método next que debe devolver los elementos, de a uno por vez, comenzando por el primero. Y al llegar al final de la estructura, debe levantar una excepción de tipo StopIteration.

Algunos itereadores

* list, , , : construir una lista, tupla, [diccionario,](https://www.pythonlikeyoumeanit.com/Module2_EssentialsOfPython/DataStructures_II_Dictionaries.html)o [establecer,](https://www.pythonlikeyoumeanit.com/Module2_EssentialsOfPython/DataStructures_III_Sets_and_More.html" \l "The-%E2%80%9CSet%E2%80%9D-Data-Structure)respectivamente, a partir del contenido de un iterabletupledictset
* sum: suma el contenido de un iterable.
* sorted: devuelve una lista del contenido ordenado de un
* any: devuelve y finaliza la iteración inmediatamente si fuera para cualquier elemento en el iterable.Truebool(item)True
* all: devuelve solo si era para todos los elementos en iterable. True bool (item) True
* max: devuelve el valor más grande en un iterable.
* min: devuelve el valor más pequeño en un iterable.
* for elemento in secuencia: *# hacer algo con elemento*
* iterador = iter(secuencia)
* while True:

try:

elemento = iterador.next()

except StopIteration:

break

*# hacer algo con elemento*

# BIBLIOGRAFÍA

* <https://docs.python.org/3/tutorial/errors.html>
* <https://www.w3schools.com/python/python_try_except.asp>
* <https://www.w3schools.com/python/ref_func_zip.asp>
* <https://www.programiz.com/python-programming/methods/built-in/zip>
* <https://www.geeksforgeeks.org/enumerate-in-python/>
* <https://www.programiz.com/python-programming/methods/built-in/enumerate>
* <https://www.programiz.com/python-programming/list-comprehension>
* <https://realpython.com/list-comprehension-python/>
* <https://www.pythonforbeginners.com/basics/list-comprehensions-in-python>
* <https://www.pythonlikeyoumeanit.com/Module2_EssentialsOfPython/Iterables.html>
* <https://uniwebsidad.com/libros/algoritmos-python/capitulo-16/iteradores>
* https://stackoverrun.com/es/q/1898