Errors and Exceptions

## **Syntax Errors**

**while** **True** print('Hello world')

## **Exceptions**

Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it. Errors detected during execution are called *exceptions* and are not unconditionally fatal: you will soon learn how to handle them in Python programs.

10 \* (1/0)

4 + spam\*3

'2' + 2

## **Handling Exceptions**

**while** **True**:

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

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

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

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

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

The [try](https://docs.python.org/3/reference/compound_stmts.html#try) statement works as follows.

* First, the *try clause* (the statement(s) between the [try](https://docs.python.org/3/reference/compound_stmts.html#try) and [except](https://docs.python.org/3/reference/compound_stmts.html#except) keywords) is executed.
* If no exception occurs, the *except clause* is skipped and execution of the [try](https://docs.python.org/3/reference/compound_stmts.html#try) statement is finished.
* If an exception occurs during execution of the [try](https://docs.python.org/3/reference/compound_stmts.html#try) clause, the rest of the clause is skipped. Then, if its type matches the exception named after the [except](https://docs.python.org/3/reference/compound_stmts.html#except) keyword, the *except clause* is executed, and then execution continues after the try/except block.
* If an exception occurs which does not match the exception named in the *except clause*, it is passed on to outer [try](https://docs.python.org/3/reference/compound_stmts.html#try) statements; if no handler is found, it is an *unhandled exception* and execution stops with a message as shown above.

## **Raising Exceptions**

**raise** **NameError**('HiThere')

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: HiThere

## **Exception Chaining**

**raise** **RuntimeError** **from** **exc**

**def** func():

**raise** **ConnectionError**

**try**:

func()

**except** **ConnectionError** **as** exc:

**raise** **RuntimeError**('Failed to open database') **from** **exc**

| Exception | Description |
| --- | --- |
| AssertionError | Raised when the assert statement fails. |
| AttributeError | Raised on the attribute assignment or reference fails. |
| EOFError | Raised when the input() function hits the end-of-file condition. |
| FloatingPointError | Raised when a floating point operation fails. |
| GeneratorExit | Raised when a generator's close() method is called. |
| ImportError | Raised when the imported module is not found. |
| IndexError | Raised when the index of a sequence is out of range. |
| KeyError | Raised when a key is not found in a dictionary. |
| KeyboardInterrupt | Raised when the user hits the interrupt key (Ctrl+c or delete). |
| MemoryError | Raised when an operation runs out of memory. |
| NameError | Raised when a variable is not found in the local or global scope. |
| NotImplementedError | Raised by abstract methods. |
| OSError | Raised when a system operation causes a system-related error. |
| OverflowError | Raised when the result of an arithmetic operation is too large to be represented. |
| ReferenceError | Raised when a weak reference proxy is used to access a garbage collected referent. |
| RuntimeError | Raised when an error does not fall under any other category. |
| StopIteration | Raised by the next() function to indicate that there is no further item to be returned by the iterator. |
| SyntaxError | Raised by the parser when a syntax error is encountered. |
| IndentationError | Raised when there is an incorrect indentation. |
| TabError | Raised when the indentation consists of inconsistent tabs and spaces. |
| SystemError | Raised when the interpreter detects internal error. |
| SystemExit | Raised by the sys.exit() function. |
| TypeError | Raised when a function or operation is applied to an object of an incorrect type. |
| UnboundLocalError | Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable. |
| UnicodeError | Raised when a Unicode-related encoding or decoding error occurs. |
| UnicodeEncodeError | Raised when a Unicode-related error occurs during encoding. |
| UnicodeDecodeError | Raised when a Unicode-related error occurs during decoding. |
| UnicodeTranslateError | Raised when a Unicode-related error occurs during translation. |
| ValueError | Raised when a function gets an argument of correct type but improper value. |
| ZeroDivisionError | Raised when the second operand of a division or module operation is zero. |

# Python Lists

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are [Tuple](https://www.w3schools.com/python/python_tuples.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

Lists are created using square brackets:

thislist = ["apple", "banana", "cherry"]  
print(thislist)

## **List Items**

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc

thislist = ["apple", "banana", "cherry"]  
print(len(thislist))

# Python Sets

Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are [List](https://www.w3schools.com/python/python_lists.asp), [Tuple](https://www.w3schools.com/python/python_tuples.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

**There is no index attached to any element in a python set**

A set is a collection which is *unordered*, *unchangeable\**, and *unindexed*.

thisset = {"apple", "banana", "cherry"}  
print(thisset)

thisset = {"apple", "banana", "cherry"}  
print(len(thisset))

myset = {"apple", "banana", "cherry"}  
print(type(myset))

# Python Tuples

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are [List](https://www.w3schools.com/python/python_lists.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

A tuple is a collection which is ordered and **unchangeable**.

Tuples are written with round brackets.

Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

thistuple = ("apple", "banana", "cherry")  
print(thistuple)

thistuple = ("apple", "banana", "cherry")  
print(len(thistuple))

thistuple = ("apple",)  
print(type(thistuple))  
  
#NOT a tuple  
thistuple = ("apple")  
print(type(thistuple))

mytuple = ("apple", "banana", "cherry")  
print(type(mytuple))

thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets  
print(thistuple)

# Python Dictionaries

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered\*, changeable and do not allow duplicates.

As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

Dictionaries are written with curly brackets, and have keys and values:

# thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964

# "year": 2020 } print(thisdict)

# print(thisdict["brand"])

# print(len(thisdict))

# print(type(thisdict))

# Logging in Python

# The logging is a powerful module used by the beginners as well as enterprises. This module provides a proficiency to organize different control handlers and a transfer log messages to these handlers.

Let's understand the following events and their working.

1. **DEBUG -** It is used to provide detailed information and only use it when there is diagnosing problems.
2. **INFO -** It provides the information regarding that things are working as we want.
3. **WARNING -** It is used to warn that something happened unexpectedly, or we will face the problem in the upcoming time.
4. **ERROR -** It is used to inform when we are in some serious trouble, the software hasn't executed some programs.
5. **CRITICAL -** It specifies the serious error, the program itself may be incapable of remaining executing.

import logging

logging.debug('The debug message is displaying')

logging.info('The info message is displaying')

logging.warning('The warning message is displaying')

logging.error('The error message is displaying')

logging.critical('The critical message is displaying')

# Python Collections Module

The collection Module in Python provides different types of containers. A Container is an object that is used to store different objects and provide a way to access the contained objects and iterate over them.

***Table of Content:***

* [*Counters*](https://www.geeksforgeeks.org/python-collections-module/#counters)

A [counter](https://www.geeksforgeeks.org/counters-in-python-set-1/) is a sub-class of the dictionary. It is used to keep the count of the elements in an iterable in the form of an unordered dictionary where the key represents the element in the iterable and value represents the count of that element in the iterable.

# A Python program to show different

# ways to create Counter

**from** collections **import** Counter

# With sequence of items

print(Counter(['B','B','A','B','C','A','B',

               'B','A','C']))

# with dictionary

**print**(Counter({'A':3, 'B':5, 'C':2}))

# with keyword arguments

print(Counter(A**=**3, B**=**5, C**=**2))

* [*OrderedDict*](https://www.geeksforgeeks.org/python-collections-module/#ordereddict)

An [OrderedDict](https://www.geeksforgeeks.org/ordereddict-in-python/)is also a sub-class of dictionary but unlike dictionary, it remembers the order in which the keys were inserted.

# A Python program to demonstrate working

# of OrderedDict

**from** collections **import** OrderedDict

print("This is a Dict:\n")

d **=** {}

d['a'] **=** 1

d['b'] **=** 2

d['c'] **=** 3

d['d'] **=** 4

**for** key, value **in** d.items():

**print**(key, value)

**print**("\nThis is an Ordered Dict:\n")

od **=** OrderedDict()

od['a'] **=** 1

od['b'] **=** 2

od['c'] **=** 3

od['d'] **=** 4

**for** key, value **in** od.items():

    print(key, value)

* [*DefaultDict*](https://www.geeksforgeeks.org/python-collections-module/#defaultdict)

A [DefaultDict](https://www.geeksforgeeks.org/defaultdict-in-python/) is also a sub-class to dictionary. It is used to provide some default values for the key that does not exist and never raises a KeyError.

# Python program to demonstrate

# defaultdict

**from** collections **import** defaultdict

# Defining the dict

d **=** defaultdict(int)

L **=** [1, 2, 3, 4, 2, 4, 1, 2]

# Iterate through the list

# for keeping the count

**for** i **in** L:

    # The default value is 0

    # so there is no need to

# enter the key first

    d[i] **+=** 1

print(d)

* [*ChainMap*](https://www.geeksforgeeks.org/python-collections-module/#chainmap)

A [ChainMap](https://www.geeksforgeeks.org/chainmap-in-python/) encapsulates many dictionaries into a single unit and returns a list of dictionaries.

**from** collections **import** ChainMap

d1 **=** {'a': 1, 'b': 2}

d2 **=** {'c': 3, 'd': 4}

d3 **=** {'e': 5, 'f': 6}

# Defining the chainmap

c **=** ChainMap(d1, d2, d3)

print(c)

* [*NamedTuple*](https://www.geeksforgeeks.org/python-collections-module/#namedtuple)

A [NamedTuple](https://www.geeksforgeeks.org/namedtuple-in-python/) returns a tuple object with names for each position which the ordinary tuples lack. For example, consider a tuple names student where the first element represents fname, second represents lname and the third element represents the DOB.

**from** collections **import** namedtuple

# Declaring namedtuple()

Student **=** namedtuple('Student',['name','age','DOB'])

# Adding values

S **=** Student('Nandini','19','2541997')

# Access using index

**print** ("The Student age using index is : ",end **=**"")

print (S[1])

# Access using name

**print** ("The Student name using keyname is : ",end **=**"")

print (S.name)

# Python Datetime

A date in Python is not a data type of its own, but we can import a module named datetime to work with dates as date objects.

import datetime  
x=datetime.datetime.now()  
print(x)

print(x.year)  
print(x.strftime("%A"))

import datetime  
x=datetime.datetime(2018, 6, 1)  
print(x.strftime("%B"))

| **Directive** | **Description** | **Example** | **Try it** |
| --- | --- | --- | --- |
| %a | Weekday, short version | Wed | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_a) |
| %A | Weekday, full version | Wednesday | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_a2) |
| %w | Weekday as a number 0-6, 0 is Sunday | 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_w) |
| %d | Day of month 01-31 | 31 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_d) |
| %b | Month name, short version | Dec | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_b) |
| %B | Month name, full version | December | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_b2) |
| %m | Month as a number 01-12 | 12 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_m) |
| %y | Year, short version, without century | 18 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_y) |
| %Y | Year, full version | 2018 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_y2) |
| %H | Hour 00-23 | 17 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_h2) |
| %I | Hour 00-12 | 05 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_i2) |
| %p | AM/PM | PM | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_p) |
| %M | Minute 00-59 | 41 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_m2) |
| %S | Second 00-59 | 08 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_s2) |
| %f | Microsecond 000000-999999 | 548513 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_f) |
| %z | UTC offset | +0100 |  |
| %Z | Timezone | CST |  |
| %j | Day number of year 001-366 | 365 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_j) |
| %U | Week number of year, Sunday as the first day of week, 00-53 | 52 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_u2) |
| %W | Week number of year, Monday as the first day of week, 00-53 | 52 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_w2) |
| %c | Local version of date and time | Mon Dec 31 17:41:00 2018 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_c) |
| %C | Century | 20 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_century) |
| %x | Local version of date | 12/31/18 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_x) |
| %X | Local version of time | 17:41:00 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_x2) |
| %% | A % character | % | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_percent) |
| %G | ISO 8601 year | 2018 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_g) |
| %u | ISO 8601 weekday (1-7) | 1 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_datetime_strftime_u) |
| %V | ISO 8601 weeknumber (01-53) | 01 |  |

# Python RegEx

import re

import re  
txt= "TheraininSpain"  
x = re.search("^The.\*Spain$", txt)

| **Character** | **Description** | **Example** | **Try it** |
| --- | --- | --- | --- |
| [] | A set of characters | "[a-m]" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta1) |
| \ | Signals a special sequence (can also be used to escape special characters) | "\d" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta2) |
| . | Any character (except newline character) | "he..o" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta3) |
| ^ | Starts with | "^hello" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta4) |
| $ | Ends with | "planet$" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta5) |
| \* | Zero or more occurrences | "he.\*o" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta6) |
| + | One or more occurrences | "he.+o" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta7) |
| ? | Zero or one occurrences | "he.?o" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta10) |
| {} | Exactly the specified number of occurrences | "he.{2}o" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta8) |
| | | Either or | "falls|stays" |  |

import re  
  
txt= "TheraininSpain"  
x=re.findall("ai", txt)  
print(x)

import re  
  
txt= "TherainSpain"  
x=re.findall("Portugal", txt)  
print(x)

import re  
txt= "TheraininSpain"  
x=re.search("\s", txt)  
print("The first white-space character is located in position:", x.start())