Useful exceptions

Built-in exceptions

We're going to show you a short list of the most useful exceptions. While it may sound strange to call "useful" a thing or a phenomenon which is a visible sign of failure or setback, as you know, to err is human and if anything can go wrong, it will go wrong.

Exceptions are as routine and normal as any other aspect of a programmer's life.

For each exception, we'll show you:

- its name;
- its location in the exception tree;
- a short description;
- a concise snippet of code showing the circumstances in which the exception may be raised.

There are lots of other exceptions to explore - we simply don't have the space to go through them all here.

`ArithmeticError`

Location:

`BaseException ← Exception ← ArithmeticError`

Description:

an abstract exception including all exceptions caused by arithmetic operations like zero division or an argument's invalid domain

`AssertionError`

Location:

`BaseException ← Exception ← AssertionError`

Description:

a concrete exception raised by the assert instruction when its argument evaluates to `False`, `None`, `0`, or an empty string

Code:

```
from math import tan, radians
angle = int(input('Enter integral angle in degrees: '))

# we must be sure that angle != 90 + k * 180
assert angle % 180 != 90
print(tan(radians(angle)))
```

`BaseException`

Location:

`BaseException`

Description:

the most general (abstract) of all Python exceptions - all other exceptions are included in this one; it can be said that the following two `except` branches are equivalent: `except:` and `except BaseException:`.

IndexError

Location:

`BaseException ← Exception ← LookupError ← IndexError`

Description:

a concrete exception raised when you try to access a non-existent sequence's element (e.g., a list's element)

Code:

```
# the code shows an extravagant way
# of leaving the loop

list = [1, 2, 3, 4, 5]
ix = 0
doit = True

while doit:
      try:
            print(list[ix])
            ix += 1
      except IndexError:
            doit = False

print('Done')
```

KeyboardInterrupt

Location:

`BaseException ← KeyboardInterrupt`

Description:

a concrete exception raised when the user uses a keyboard shortcut designed to terminate a program's execution (*Ctrl-C* in most OSs); if handling this exception doesn't lead to program termination, the program continues its execution. Note: this exception is not derived from the `Exception` class. Run the program in IDLE.

Code:

```
# this code cannot be terminated
# by pressing Ctrl-C
```

```
from time import sleep
seconds = 0
while True:
	try:
		print(seconds)
		seconds += 1
		sleep(1)
	except KeyboardInterrupt:
		print("Don't do that!")
```

LookupError

Location:

```
BaseException ← Exception ← LookupError
```

Description:

an abstract exception including all exceptions caused by errors resulting from invalid references to different collections (lists, dictionaries, tuples, etc.)

MemoryError

Location:

```
BaseException ← Exception ← MemoryError
```

Description:

a concrete exception raised when an operation cannot be completed due to a lack of free memory

Code:

```
# this code causes the MemoryError exception
# warning: executing this code may be crucial
# for your OS
# don't run it in production environments!

string = 'x'
try:
	while True:
		string = string + string
		print(len(string))
except MemoryError:
	print('This is not funny!')
```

OverflowError

Location:

```
BaseException ← Exception ← ArithmeticError ← OverflowError
```

Description:

a concrete exception raised when an operation produces a number too big to be successfully stored

Code:

```
# the code prints subsequent
# values of exp(k), k = 1, 2, 4, 8, 16, ...

from math import exp
ex = 1
try:
      while True:
            print(exp(ex))
            ex *= 2
      except OverflowError:
            print('The number is too big.')
```

ImportError

Location:

```
BaseException ← Exception ← StandardError ← ImportError
```

Description:

a concrete exception raised when an import operation fails

Code:

```
# one of this imports will fail - which one?

try:
      import math
      import time
      import abracadabra

except:
      print('One of your imports has failed.')
```

KeyError

Location:

```
BaseException ← Exception ← LookupError ← KeyError
```

Description:

a concrete exception raised when you try to access a collection's non-existent element (e.g., a dictionary's element)

Code:

```
# how to abuse the dictionary
# and how to deal with it

dict = { 'a' : 'b', 'b' : 'c', 'c' : 'd' }
ch = 'a'
try:
      while True:
            ch = dict[ch]
            print(ch)
except KeyError:
      print('No such key:', ch)
```

We are done with exceptions for now, but they'll return when we discuss object-oriented programming in Python. You can use them to protect your code from bad accidents, but you also have to learn how to dive into them, exploring the information they carry.

Exceptions are in fact objects - however, we can tell you nothing about this aspect until we present you with classes, objects, and the like.

For the time being, if you'd like to learn more about exceptions on your own, you look into Standard Python Library at https://docs.python.org/3.6/library/exceptions.html.