Exceptions

dividing by zero creates an exception

```
>>> print(55/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
>>>
```

accessing a nonexistent list item:

```
>>> a = []
>>> print (a[5])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>>
```

Or trying to make an item assignment on a tuple:

```
>>> tup = ('a', 'b', 'd', 'd')
>>> tup[2] = 'c'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

we might prompt the user for the name of a file and then try to open it. If the file doesn't exist, we don't want the program to crash; we want to handle the exception

```
filename = raw_input('Enter a file name: ')
try:
    f = open (filename, "r")
except:
    print 'There is no file named', filename
```

If your program detects an error condition, you can make it **raise** an exception. Here is an example that gets input from the user and checks that the number is non-negative.

```
#
# learn_exceptions.py
#
def get_age():
    age = input('Please enter your age: ')
    if age < 0:
        raise ValueError, '%s is not a valid age' % age
    return age
```

If the function that called `get_age` handles the error, then the program can continue; otherwise, Python prints the traceback and exits:

```
>>> get_age()
Please enter your age: 42
42
>>> get_age()
Please enter your age: -2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "learn_exceptions.py", line 4, in get_age
    raise ValueError, '%s is not a valid age' % age
ValueError: -2 is not a valid age
>>>
```

The error message includes the exception type and the additional information you provided