Exceptions
(Oops! When things go wrong)

- Errors and Exceptions
- Syntax errors
- State (execution or runtime) errors

Types of errors

There are two types of errors:

- syntax errors
- erroneous state errors

```
>>> excuse = 'I'm sick'
SyntaxError: invalid syntax
>>> print(hour+':'+minute+':'+second)
Traceback (most recent call last):
  File "<pyshell#113>", line 1, in <module>
    print(hour+':'+minute+':'+second)
TypeError: unsupported operand type(s) for +:
'int' and 'str’
>>> infile = open('sample.txt')
Traceback (most recent call last):
  File "<pyshell#50>", line 1, in <module>
    infile = open('sample.txt')
FileNotFoundError: [Errno 2] No such file or
directory: 'sample.txt’
```

Syntax errors

A syntax error is a Python statement that is not correctly structured.

- It is detected while the interpreter is translating the statement to machine language, before it is executed.

```
>>> (3+4] SyntaxError:
invalid syntax
>>> if x == 5
SyntaxError: invalid syntax
>>> print 'hello'
SyntaxError: invalid syntax
>>> lst = [4;5;6]
SyntaxError: invalid syntax
>>> for i in range(10):
print(i)
SyntaxError: expected an indented block
```

State errors

When a state (runtime) error occurs, an "error" object is created

- This object has a type that is related to the *type* of error
- The object contains information about the error
- The default behavior is to print this information and interrupt the execution of the statement.

The "error" object is called an exception. The creation of an exception due to an error is called raising an exception

```
>>> int('4.5')
Traceback (most recent call last):
  File "<pyshell#61>", line 1, in <module>
ValueError: invalid literal for int() with
base 10: '4.5'
```

```
>>> 3/0
Traceback (most recent call last):
  File "<pyshell#56>", line 1, in <module>
ZeroDivisionError: division by zero
```

```
>>> lst
Traceback (most recent call last):
  File "<pyshell#57>", line 1, in <module>
NameError: name 'lst' is not defined
```

```
>>> lst = [12, 13, 14]
>>> lst[3]
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
IndexError: list index out of range
```

```
>>> lst * lst
Traceback (most recent call last):
  File "<pyshell#60>", line 1, in <module>
    lst * lst
TypeError: can't multiply sequence by non-
int of type 'list'
```

Exception types

Some built-in exception classes:

Exception	Explanation
`KeyboardInterrupt`	Raised when user hits Ctrl-C, the interrupt key
`OverflowError`	Raised when a floating-point expression evaluates to a value that is too large
`ZeroDivisionError`	Raised when attempting to divide by 0
`FileNotFoundError`	Raised when a file or directory is requested but doesn't exist
`IndexError`	Raised when a sequence index is outside the range of valid indexes
`NameError`	Raised when attempting to evaluate an unassigned identifier (name)
`TypeError`	Raised when an operation of function is applied to an object of the wrong type
`ValueError`	Raised when operation or function has an argument of the right type but incorrect value

Catching and handling exceptions

It is possible to override the default behavior (print error information and "crash") when an exception is raised using `try/except` statements

If an exception is raised while executing the `try` block, then the block of the associated `except` statement is executed

```
try:
    strAge = input('Enter your age: ')
    intAge = int(strAge)
    print('You are {} years old.'.format(intAge))
except:
    print('Enter your age using digits 0-9!')
```

The `except` code block is the exception handler

Default behavior:

```
>>> =============== RESTART ===============
>>> Enter your age: fifteen
Traceback (most recent call last):
  File "/Users/me/age1.py", line 2, in <module>
    intAge = int(strAge)
ValueError: invalid literal for int() with base
10: 'fifteen'
>>>
```

```
>>> ========== RESTART
==========
>>> Enter your age: fifteen
Enter your age using digits 0-
9!
>>>
```

Format of a `try/except` statement pair

The format of a `try/except` pair of statements is:

```
try:
    <indented code block>
except:
    <exception handler block>
<non-indented statement>
```

The exception handler handles any exception raised in the `try` block

The `except` statement is said to catch the (raised) exception

It is possible to restrict the `except` statement to catch exceptions of a specific type only

```
try:
    <indented code block>
except <ExceptionType>:
    <exception handler block>
<non-indented statement>
```

Format of a `try/except` statement pair

```
def readAge(filename):
    'converts first line of file filename to an integer and prints it'
    try:
        infile = open(filename)
        strAge = infile.readline()
        age = int(strAge)
        print('age is', age)
    except ValueError:
        print('Value cannot be converted to integer.')
```

It is possible to restrict the `except` statement to catch exceptions of a specific type only

```
1 fifteen
```

age.txt

```
>>> readAge('age.txt')
Value cannot be converted to integer.
>>> readAge('age.text')
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    readAge('age.text')
  File "/Users/me/ch7.py", line 12, in readAge
    infile = open(filename)
FileNotFoundError: [Errno 2] No such file or directory:
'age.text'
>>>
```

default exception handler prints this

Multiple except clauses

A try statement may have more than one except clause, to specify handlers for different exceptions. At most one handler will be executed.

```python
def readAge(filename):
    try:
        infile = open(filename)
        strAge = infile.readline()
        age = int(strAge)
        print('age is', age)
    except ValueError:
        print('Value cannot be converted to integer.')
    except FileNotFoundError:
        print('Input file is missing:', "'" + filename + "'")
```

An except clause may name multiple exceptions as a parenthesized tuple.

```python
def readAge(filename):
    try:
        infile = open(filename)
        strAge = infile.readline()
        age = int(strAge)
        print('age is', age)
    except (ValueError, FileNotFoundError):
        print('Conversion error or file not found.')
```