Error Types in Python

```
def foo():
    assert 1 == 2, 'My error'
    ...
```

```
>>> foo()
AssertionError: My error
```

```
def foo():
    x = 5 / 0
    ...
```

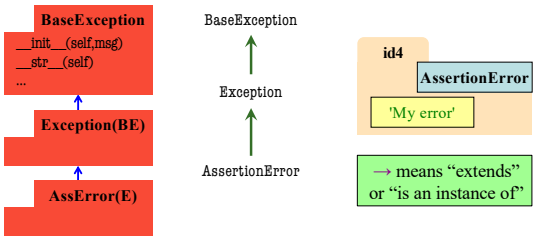
```
>>> foo()
ZeroDivisionError: integer
division or modulo by zero
```

Class Names

1

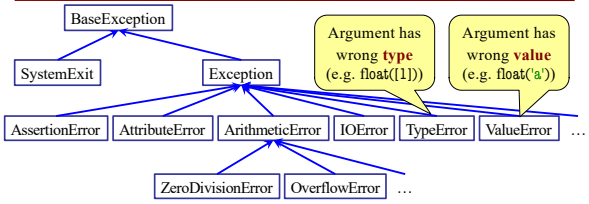
Error Types in Python

- All errors are instances of class BaseException
- This allows us to organize them in a hierarchy



2

Python Error Type Hierarchy



http://docs.python.org/library/exceptions.html

Why so many error types?

3

Handling Errors by Type

- try-except blocks can be restricted to **specific** errors
  - Doe except if error is **an instance** of that type
  - If error not an instance, do not recover
- **Example**:

```
try:
    val = input()        # get number from user
    x = float(val)       # convert string to float
    print('The next number is '+str(x+1))
except ValueError:
    print('Hey! That is not a number!')
```

May have IOError

May have ValueError

Only recovers ValueError. Other errors ignored.

4

Creating Errors in Python

- Create errors with raise
  - **Usage**: raise <exp>
  - exp evaluates to an object
  - An instance of Exception
- Tailor your error types
  - **ValueError**: Bad value
  - **TypeError**: Bad type
- Still prefer **asserts** for preconditions, however
  - Compact and easy to read

```
def foo(x):
    assert x < 2, 'My error'
    ...
```

Identical

```
def foo(x):
    if x >= 2:
        m = 'My error'
        err = AssertionError(m)
        raise err
```

5

Handling Errors by Type

- try-except can put the error in a variable
- **Example**:

```
try:
    val = input()        # get number from user
    x = float(val)       # convert string to float
    print('The next number is '+str(x+1))
except ValueError as e:
    print(e.args[0])
    print('Hey! That is not a number!')
```

Some Error subclasses have more attributes

6

Creating Your Own Exceptions

```
class CustomError(Exception):
    """An instance is a custom exception"""
    pass
```

Only issues is choice of parent error class. Use Exception if you are unsure what.

This is all you need
- No extra fields
- No extra methods
- No constructors

Inherit everything

7

Case Study: Files

- Can read the contents of any file with open()
  - Returns a file object with method read()
  - Method read() returns contents as a string
  - Remember to close() file when done
- There are **SO** many errors that can happen
  - FileNotFoundError: File does not exit
  - PermissionError: You are not allowed to read it
  - Other errors possible when processing data

8

Recall: JSON Files

```
{
  "wind" : {
    "speed" : 13.0,
    "crosswind" : 5.0
    },
  "sky" : [
    {
      "cover" : "clouds",
      "type" : "broken",
      "height" : 1200.0
    },
    {
      "type" : "overcast",
      "height" : 1800.0
    }
  ]
}
```

- Look like a nested dict
  - But read in as a string
  - You have to **convert** it
- Python module json
  - Function loads()
    Converts str -> dict
  - Function dumps()
    Convert dict -> str
- Conversion is sensitive
  - Stray commas crash it

9

Reading a JSON File

```
def read_json(fname):
    try:
        file = open(fname)
        data = file.read()
        file.close()
        result = json.loads(data)
        return result
    except FileNotFoundError:
        print(fname +' not found')
    except JsonDecodeError:
        print(fname +' is invalid')
    return None
```

Open file with name

Close file when done

Could not find file

JSON contents are not valid

If failed

Note that we can chain excepts like an if-elif statement

10

Aside: Pathnames

- Files obey the same rule as other modules
  - To read a file, it must be in the same folder
  - Otherwise, you must use a pathname for file
- **Relative path**: directions from current folder
  - **macOS**: '../../lec22/file.txt'
  - **Windows**: '..\..\lec22\file.txt'

Like navigating command shell

- **Absolute path**: directions that work anywhere
  - **macOS**: '/Users/white/cs1110/lect22/file.txt'
  - **Windows**: 'C:\Users\white\cs1110\lect22\file.txt'

11

Pathnames are OS Specific

- This makes reading files harder
  - May work on Windows but crash on macOS!
  - Yet another error message we need to handle
- **Solution**: Use the module os.path
  - Builds a pathname string for current os
- **Example**: os.path('..', 'cs1110', 'lec22', 'file.txt')
  - **macOS**: '../cs1110/lec22/file.txt'
  - **Windows**: '..\cs1110\lec22\file.txt'
- Absolute paths are a little trickier, but similar

12