Object-Oriented Scripting in Python

Overview

- Object-Oriented
- Object-Oriented Python
- Object-Oriented Scripting in Python
  - Unified Post Example
  - Exception Handling

Object-Oriented Programming

Objects and Classes

- What is an object?
  - A logical grouping of functions and data.
- What is a class?
  - A class is a blueprint for making an object.

Object-Oriented Programming

A Square Example

- A Square:
  - Has a width.
  - Has a color.
- Functions:
  - Circumference = 4*width
  - Area = width*width

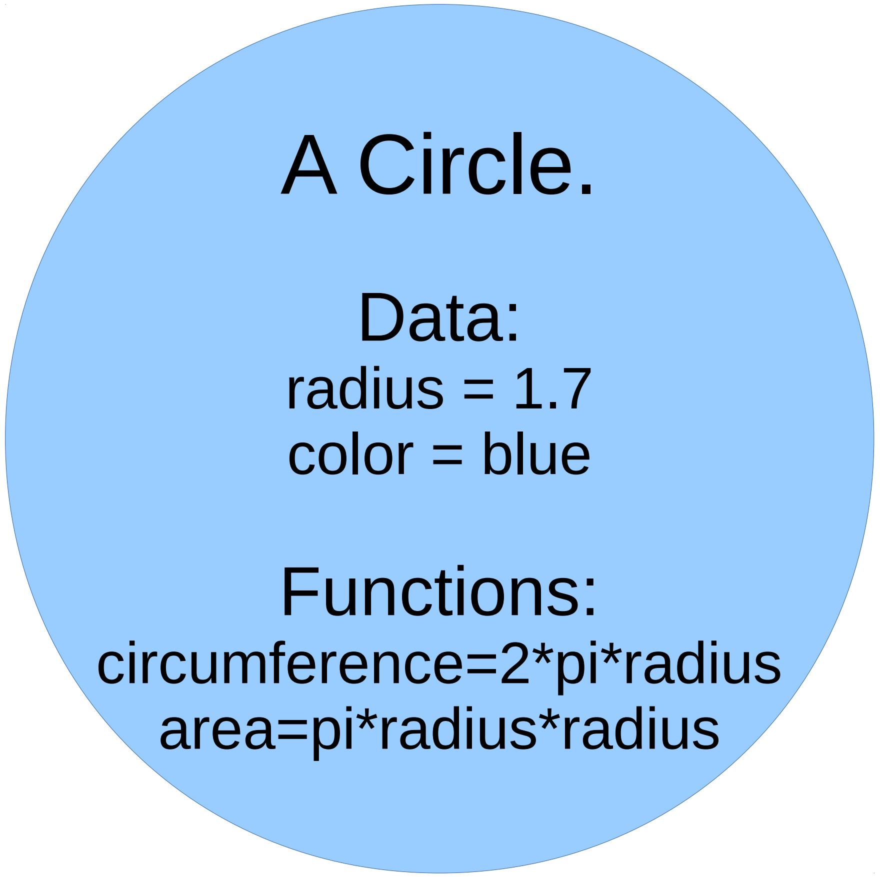
A Square.

Data:
width = 3
color = blue

Functions:
circumference=4*width
area=width*width

Object-Oriented Programming
A Circular Example

- A Circle:
  - Has a radius.
  - Has a color.
- Functions:
  - Circumference = 2*pi*radius
  - Area = pi*radius*radius

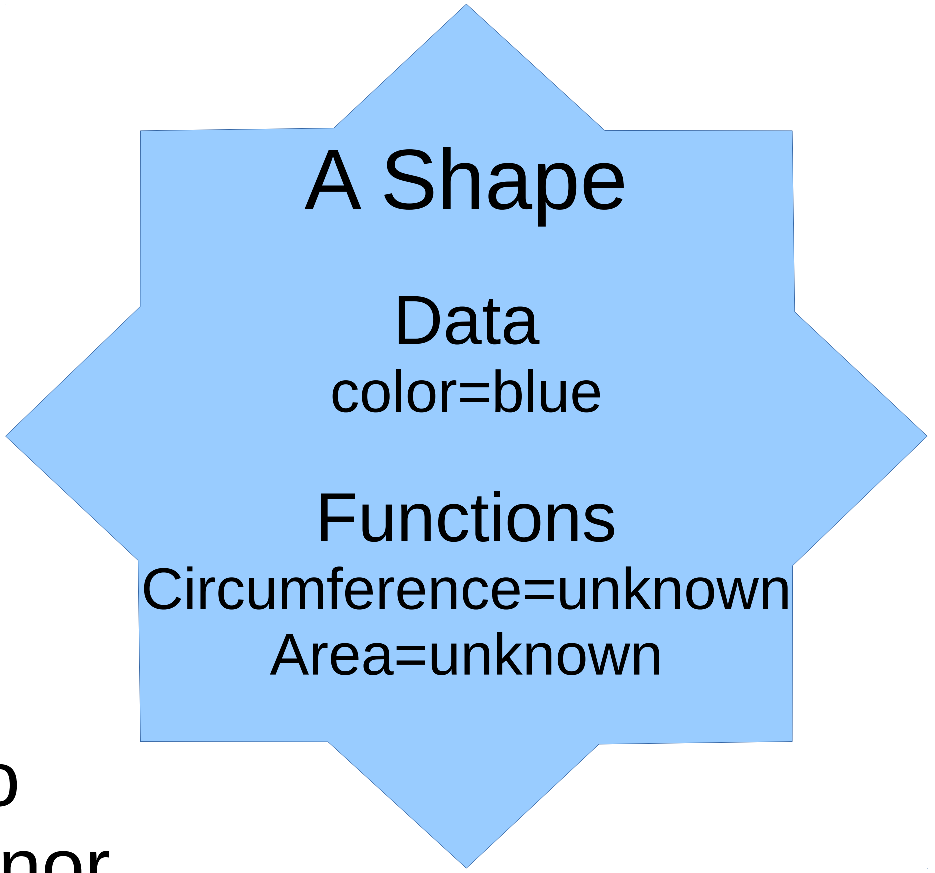


Object-Oriented Programming
Inheritance

- Squares and Circles both have colors, circumferences, and areas.
  - Why is there so much in common?
  - They are **Shapes**.
- Define a Shape class.

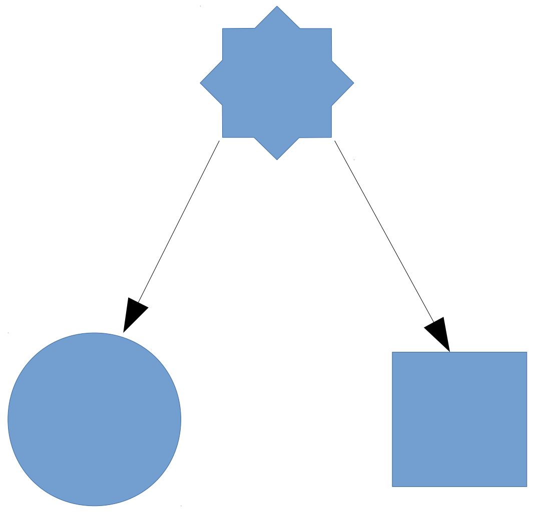
Object-Oriented Programming
A Shape Example

- A Shape:
  - Has a color.
  - Has a circumference
  - Has an area
- Pure virtual functions:
  - circumference
  - area
- Shape does not know how to determine its circumference nor its area.



Object-Oriented Programming
A Shape Example

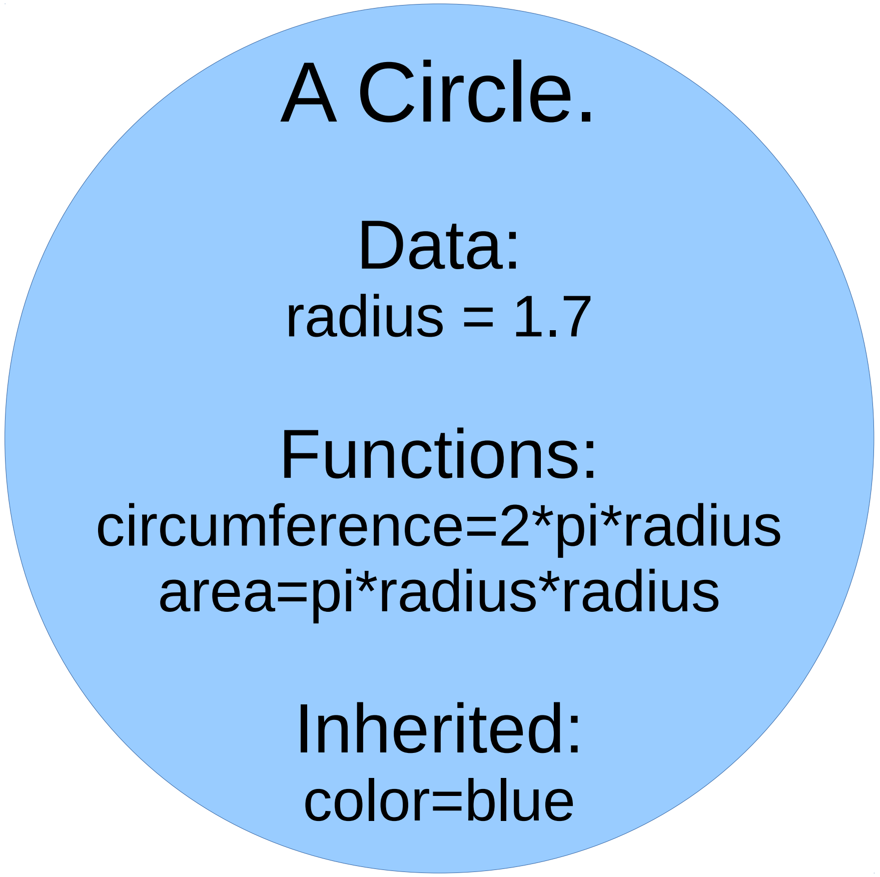
- Square and Circle are subclasses of Shape.
  - Shape implements the color.
  - Square calculates the circumference and area from the width.
  - Circle calculates the circumference and area fro the radius.



Object-Oriented Programming

A Circular Example

- A Circle:
  - Has a radius.
- Functions:
  - Circumference = 2*pi*radius
  - Area = pi*radius*radius
- Is a Shape:
  - This gives us the color.



Object-Oriented Python

class Shape

```
class Shape:
   def __init__(self,color):
      self.__color=color
   @property
   def color(self):
      return self.__color
   @property
   def circumference(self):
      return NotImplemented
   @property
   def area(self):
      return NotImplemented
```

Object-Oriented Python

class Circle

```
class Circle(Shape):
   def __init__(self,color,radius):
      super(self,Circle).__init__(color)
      self.__radius=radius
   @property
   def circumference(self):
      return math.pi*self.__radius*2
   @property
   def area(self):
      return math.pi*self.__radius**2
```

Object-Oriented Scripting

class UnifiedPost

```
class UnifiedPost:
   def __init__(self,infile,fixd,postexec,when):
      (self.infile,self.fixd.self.postexec,self.when)=\
            infile,     fixd,     postexec,     when
   def run_post(self):
      self.link_fix()
      self.make_itag()
      make_symlink(self.infile,"INFILE",
                   logger=self.log(),force=True)
      cmd=mpirun(mpi(self.postexec)<"itag")
      checkrun(cmd,all_ranks=true,logger=self.log())
   def link_fix(self):
      fixes=[f for f in glob.glob(fixd+"/*")]
      make_symlinks_in(fixes,".",logger=self.log())
```

Object-Oriented Scripting

HWRFPost, NEMSPost

```
class HWRFPost(UnifiedPost):
    def make_itag(self):
        with open(“itag”,”wt”) as f:
            itagdata=self.when.strftime(
              “INFILE\nnetcdf\n%Y-%m-%d_%H:%M:%S”
              “\nNMM NEST\n”)
            f.write(itagdata)

class NEMSPost(UnifiedPost):
    def make_itag(self):
        with open(“itag”,”wt”) as f:
            itagdata=self.when.strftime(
              “INFILE\nnetcdf\n%Y-%m-%d_%H:%M:%S”
              “\nNEMS\n”)
            f.write(itagdata)
```

Object-Oriented Scripting
Missing Pieces

- **What do we do if something fails?**
  - **Next slide...**
- How do we plug it in to scripts/, ush/ and Rocoto/ecFlow?
- How do we know when the input is “ready?”
  - Database (later presentation)
- How do we deliver the output?
  - Database (later presentation)
- How do we know what fields to produce?
  - Configuration and fix files (later presentation)

Object-Oriented Exception Handling

try/except/finally

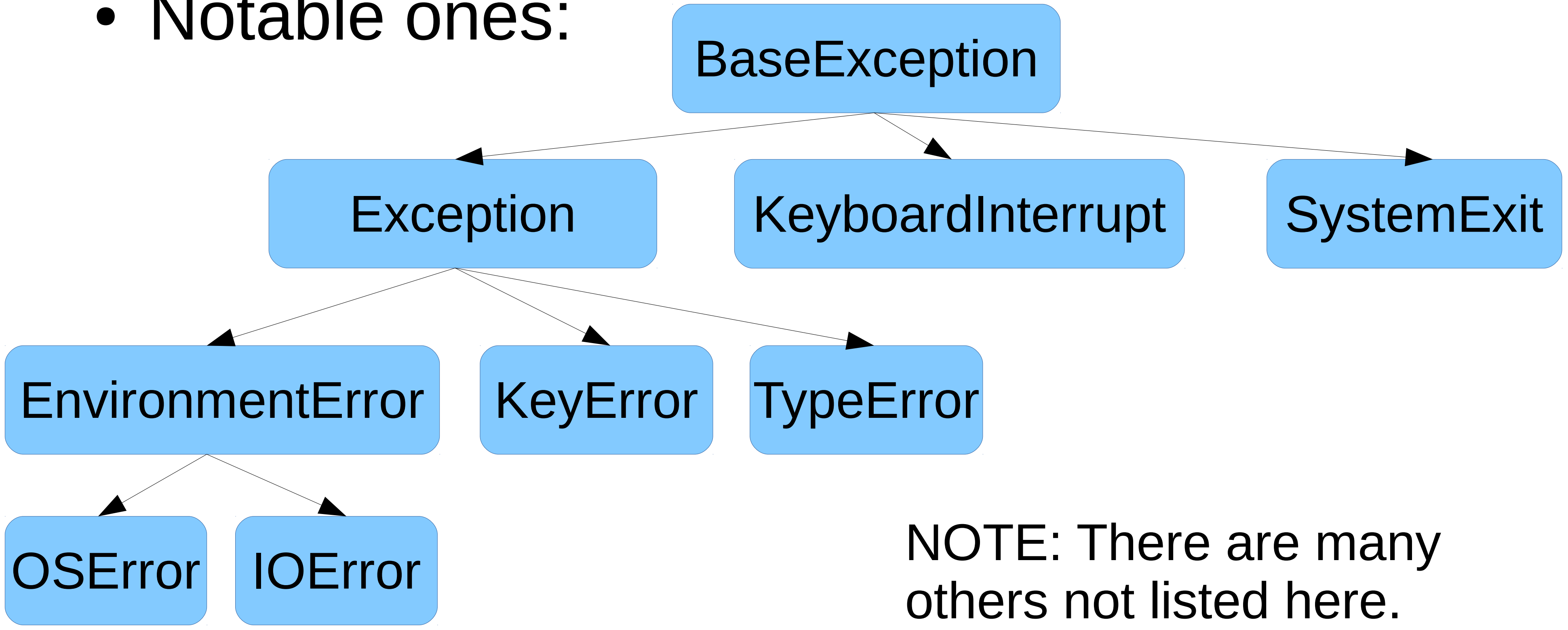
```
try:
    ... code that may break ...
except ExceptionClass as e:
    print 'Something broke!'
except AnotherExceptionClass as a:
    print 'Something else broke!'
finally:
    print 'This line is always run.'
```

- NOTE: finally and except are optional; only one of them must be present

Object-Oriented Exception Handling

Exception Classes

- Exceptions are objects.
- Python has pre-defined classes of exceptions.
- Notable ones:

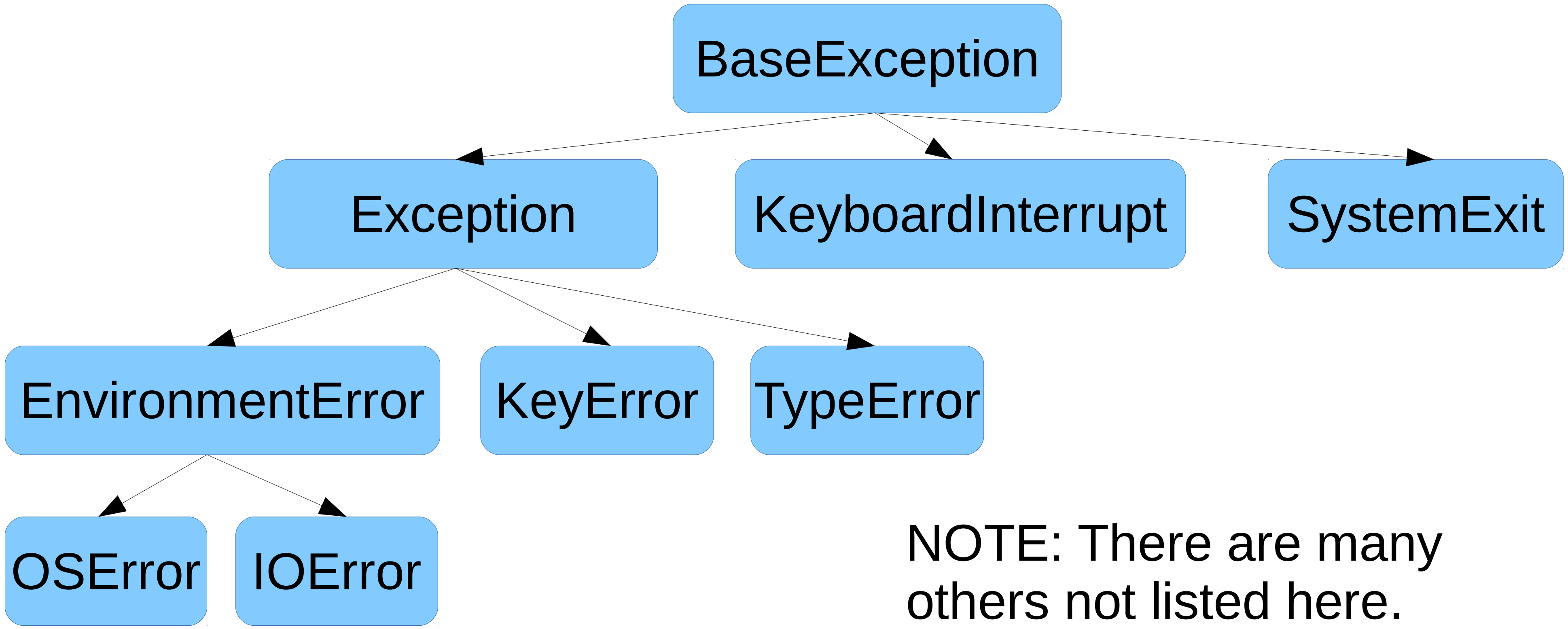


NOTE: There are many others not listed here.

Object-Oriented Exception Handling

Exception Classes

- Never catch or raise BaseException (not safe)
- Ideally, only catch subclasses of Exception.



NOTE: There are many others not listed here.

Object-Oriented Exception Handling
Custom Exception Classes

- Let's define some:

```
class PostException(Exception): pass

class PostMissingInfile(PostException): pass

class PostNoExecutable(PostException): pass
```

- PostException - base class of post exceptions
- PostMissingInfile - means INFILE is missing
  - The file from WRF, NEMS, GFS, etc. being posted.
- PostNoExecutable - post executable is missing

Object-Oriented Exception Handling

Custom Exception Classes

```
class UnifiedPost:
   ...
   def run_post(self):
      self.link_fix()
      self.make_itag()
      if not isnonempty(self.infile):
        raise PostMissingInfile(“%s: empty or nonexistent”
           %(self.infile,))
      make_symlink(self.infile,”INFILE”,
                   logger=self.log(),force=True)
      if not isnonempty(self.postexec):
        raise PostNoExecutable(“%s: empty or nonexistent”
           %(self.infile,))
      cmd=mpirun(mpi(self.postexec)<”itag”)
      checkrun(cmd,all_ranks=true,logger=self.log())
   ...
```

Object-Oriented Scripting
Missing Pieces

- **How do we plug it in to scripts/, ush/ and Rocoto/ecFlow?**
  - **Next slide...**
- How do we know when the input is “ready?”
  - Database (later presentation)
- How do we deliver the output?
  - Database (later presentation)
- How do we know what fields to produce?
  - Configuration and fix files (later presentation)

Object-Oriented Scripting
Workflow Object Structure

- ush/hwrf_expt.py:

```
post=UnifiedPost('/path/to/infile',
      '/path/to/fixd', '/path/to/hwrf_post',
      to_datetime('2015081818'))
```

- scripts/exhwrf_run_post.py:

```
import hwrf_expt
hwrf_expt.init_module()
hwrf_expt.post.run_post()
```

- Rocoto/ecFlow would be configured to run the new ex-script.

Object-Oriented Scripting

Smarter ex-script

- scripts/exhwrf_run_post.py:

```
import hwrf_expt, sys
hwrf_expt.init_module()
log=hwrf_expt.conf.log(“runpost”)
try:
   hwrf_expt.post.run_post()
except PostException as pe:
   log.error(“Post failed: “+str(pe))
   sys.exit(1)
except EnvironmentError as ee:
   log.error(“IO or OS error: “+str(ee))
   sys.exit(2)
```

Object-Oriented Scripting

“Dumb” way to wait for input.

```
...
try:
    while True:
        try:
            hwrf_expt.post.run_post()
            break # exit “while True” loop
        except PostMissingInfile as pmi:
            time.sleep(20)
except PostException as pe:
    log.error(“Post failed: “+str(pe))
    sys.exit(1)
except EnvironmentError as ee:
    log.error(“IO or OS error: “+str(ee))
    sys.exit(2)
```

Object-Oriented Scripting
Missing Pieces

- Database problems:
  - Input file is hard-coded.
  - We do not check to see if the input is ready.
  - We don't deliver the output file.
- Configuration file problems:
  - We don't know what fields to produce.
  - We don't know the correct paths to anything
- Rocoto/ecFlow:
  - How do we add this new job to the workflow?
- Later presentations will cover these aspects.

Review

- Object Oriented programming reduces code duplication.
- Object Oriented exception handling allows intelligent handling of exceptional conditions
  - and reduces code duplication
- Later presentations will cover related topics:
  - Database
  - Configuration files
  - Rocoto

?