Presentation 21

Dynamic Typing

Announcements for This Lecture

Assignments

- A4 is now graded
  - **Avg**: 89.8 **Median**: 92
  - **Std Dev**: 9.4
  - **Avg**: 9.1 hrs **Median**: 8 hrs
  - **Std Dev**: 5.1 hrs
- A5 graded by **Saturday**
- A6 is due on Sunday
  - Not guaranteed before exam

Prelim 2

- **Nov 19th at 9:30 am**
  - Working on seat/proctors
  - Will go up **Sunday**, likely
- **Material up to TODAY**
  - Recursion + Loops + Classes
  - Study guide is posted
  - Review Monday next week
- **Emergency conflicts only!**

Preparing for the Break

- This is the last "in-person" class presentation
  - We will not meet again until **December 1**
  - Thurs/Friday lab due when **we return to class**
  - But lab is still fair game for Prelim 2
- But I will post a lot of videos before then
  - **Lesson 26:** While Loops
  - **Lesson 27:** GUI Applications
  - ***All*** should be watched before returning

Preparing for the Break

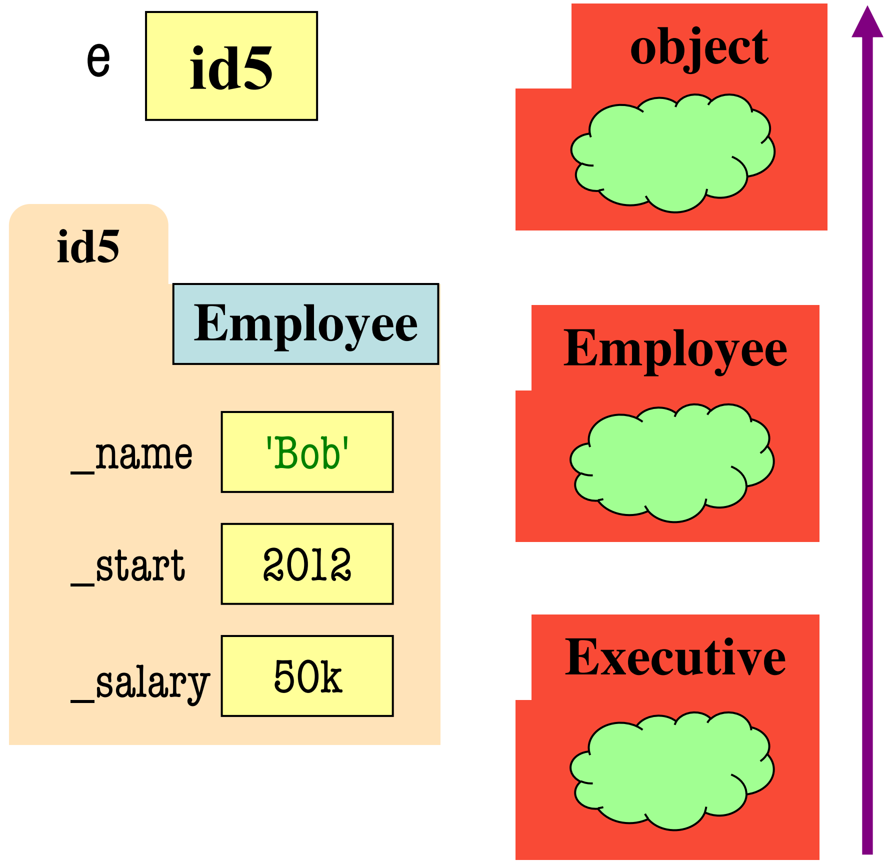
- This is the last "in-person" class presentation
  - We will not meet again until **December 1**
  - Thurs/Friday lab due when **we return to class**
  - But lab is still fair game for Prelim 2
- But I will post a lot of videos before then
  - **I**
  - **I**
  - *A*

Preparing for Assignment 7
coming **November 30th**

isinstance and Subclasses

```
>>> e = Employee('Bob',2011)
>>> isinstance(e,Executive)
???
```

A: True
B: False
C: Error
D: I don't know



isinstance and Subclasses

```
>>> e = Employee('Bob',2011)
>>> isinstance(e,Executive)
???
```

A: True
B: False   Correct
C: Error
D: I don't know

$$\begin{tikzcd} \text{object} \\ \text{Employee} \arrow[u] \\ \text{Executive} \arrow[u] \end{tikzcd}$$

→ means "extends" or "is an instance of"

Raising and Try-Except

```
def foo():
    x = 0
    try:
        raise Exception()
        x  = 2
    except Exception:
        x = 3
    return x
```

- The value of foo()?

A: 0
B: 2
C: 3
D: No value. It stops!
E: I don't know

Raising and Try-Except

```
def foo():
    x = 0
    try:
        raise Exception()
        x  = 2
    except Exception:
        x = 3
    return x
```

- The value of foo()?

A: 0
B: 2
C: 3 **Correct**
D: No value. It stops!
E: I don't know

Raising and Try-Except

```
def foo():
    x = 0
    try:
        raise Exception()
        x  = 2
    except BaseException:
        x = 3
    return x
```

- The value of foo()?

A: 0
B: 2
C: 3
D: No value. It stops!
E: I don't know

Raising and Try-Except

```
def foo():
    x = 0
    try:
        raise Exception()
        x  = 2
    except BaseException:
        x = 3
    return x
```

- The value of foo()?

A: 0
B: 2
C: 3 **Correct**
D: No value. It stops!
E: I don't know

Raising and Try-Except

```
def foo():
    x = 0
    try:
        raise Exception()
        x  = 2
    except AssertionError:
        x = 3
    return x
```

- The value of foo()?

A: 0
B: 2
C: 3
D: No value. It stops!
E: I don't know

Raising and Try-Except

```
def foo():
    x = 0
    try:
        raise Exception()
        x  = 2
    except AssertionError:
        x = 3
    return x
```

- The value of foo()?

A: 0
B: 2
C: 3
D: No value. **Correct**
E: I don't know

Python uses isinstance to match Error types

The Angle Class Revisited

```
class Angle(object):
    """A class representing an angle in DMS format

    The class does not allow a finer grained measurement than seconds
    (e.g. microseconds). All of the values must be integral."""
    # Attribute _degrees: The angle in degrees
    # Invariant: _degrees is (any) int
    # Attribute _minutes: Part of single degree
    # Invariant: _minutes is an int 0..59
    # Attribute _seconds: Part of single minute
    # Invariant: _seconds is an int 0..59
```

Enforcing Preconditions

```
class Angle(object):
    ...
    def setMinutes(self,value):
        """Sets the number of minutes


        Paramater value: The number of minutes
        Precondition: value is an int 0..59"""
1       assert type(value) == int
2       assert value >= 0 and value < 60
        self._minutes = value
```

Enforcing Preconditions

```
class Angle(object):
    …
    def setMinutes(self,value):
        """Sets the number of minutes


        Pa                                er of m
        Preco     ition: value is an int 0..59
1       assert type(value) == int
2       assert value >= 0 and value < 60
        self._minutes = value
```

Currently raises an AssertionError

What *should* (**1**) raise?

A: AssertionError

B: ValueError

C: TypeError

D: ArthmeticError

E: I don’t know

Enforcing Preconditions

```
class Angle(object):
    …
    def setMinutes(self,value):
        """Sets the number of minutes

        Pa                          er of m
        Preco      ition: value is an int 0..59
1       assert type(value) == int
2       assert value >= 0 and value < 60
        self._minutes = value
```

Currently raises an AssertionError

What *should* (**1**) raise?

A: AssertionError

B: ValueError

[✓ C: TypeError]

D: ArthmeticError

E: I don't know

Enforcing Preconditions

```
  class Angle(object):
      …
      def setMinutes(self,value):
          """Sets the number of minutes

          Pa                         er of m

          Preco      ition: value is an int 0..59
1         assert type(value) == int
2         assert value >= 0 and value < 60
          self._minutes = value
```

Currently raises an AssertionError

What *should* **(2)** raise?

A: AssertionError

B: ValueError

C: TypeError

D: ArthmeticError

E: I don't know

Enforcing Preconditions

```
class Angle(object):
    …
    def setMinutes(self,value):
        """Sets the number of minutes

        Pa                         er of m

        Prec       ition: value is an int 0..59
1       assert type(value) == int
2       assert value >= 0 and value < 60
        self._minutes = value
```

Currently raises an AssertionError

What *should* **(2)** raise?

A: AssertionError

[✓ B: ValueError]

C: TypeError

D: ArthmeticError

E: I don’t know

Enforcing Preconditions

```
class Angle(object):
    …
    def __add__(self,value):
        """Returns an angle that the sum of this angle and value

        Parameter value: The angle to add
        Precondition: value is an Angle"""
        assert type(value) == Angle
        d = self.getDegrees() + value.getDegrees()
        m = self.getMinutes() + value.getMinutes()
        …
```

Enforcing Preconditions

```
class Angle(object):
    …
    def __add__(self,value):
        """Returns an angle that the sum of this angle and value

        Pa                                  gle to add
        Precon           on: value is an Angle"""
        assert type(value) == Angle
        d = self.getDegrees() + value.getD
        m = self.getMinutes() + value.getM
        …
```

This has issues

What to use instead?

A: isintance(…)

B: Duck Typing

C: Does not matter

D: I don't know

Questions?