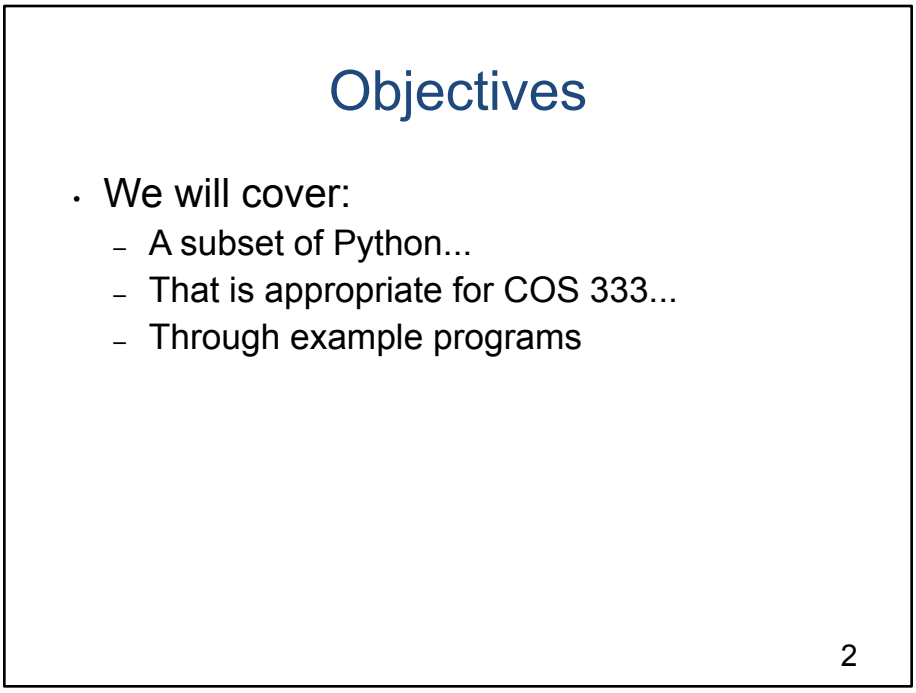
The Python Language (Part 3)

Robert M. Dondero, Ph.D.
Princeton University



Objectives

Continued from Part 2…

[see slide]

Agenda

- **Modules**
- Duck typing
- Packages
- Object-oriented programming

Modules

- ***Module***
  - A .py file that is designed to be included into a client .py file

Modules

- See **euclid.py**, **euclidclient3.py**
  - The job:
    - Same as euclidclient2.py

```
$ python euclidclient3.py
Enter the first integer: 8
Enter the second integer: 12
gcd: 4
lcm: 24
$ python euclidclient3.py
Enter the first integer: 8
Enter the second integer: 0
gcd: 8
lcm(i,j) is undefined if i or j is 0
$ python euclidclient3.py
Enter the first integer: 0
Enter the second integer: 0
gcd(i,j) is undefined if i and j are 0
$
```


Modules

[see slide]

Notes:

Factors gcd() and lcm() defs into distinct euclid **module**
euclidclient3.py is a **client** of the euclid module
euclidclient3.py must import from euclid.py

Modules

- Building and running

```
$ python euclidclient3.py
```

  - Automatically compiles/interprets euclid.py
  - Directory containing euclid.py must be in `sys.path`
  - "." is in `sys.path` automatically

Agenda

- Modules
- **Duck typing**
- Packages
- Object-oriented programming

Duck Typing

- See **euclidstrong.py**, **euclidclient4.py**
  - The job:
    - Same as euclid.py, euclidclient3.py
  - Which is better, euclid.py or euclidstrong.py?

8

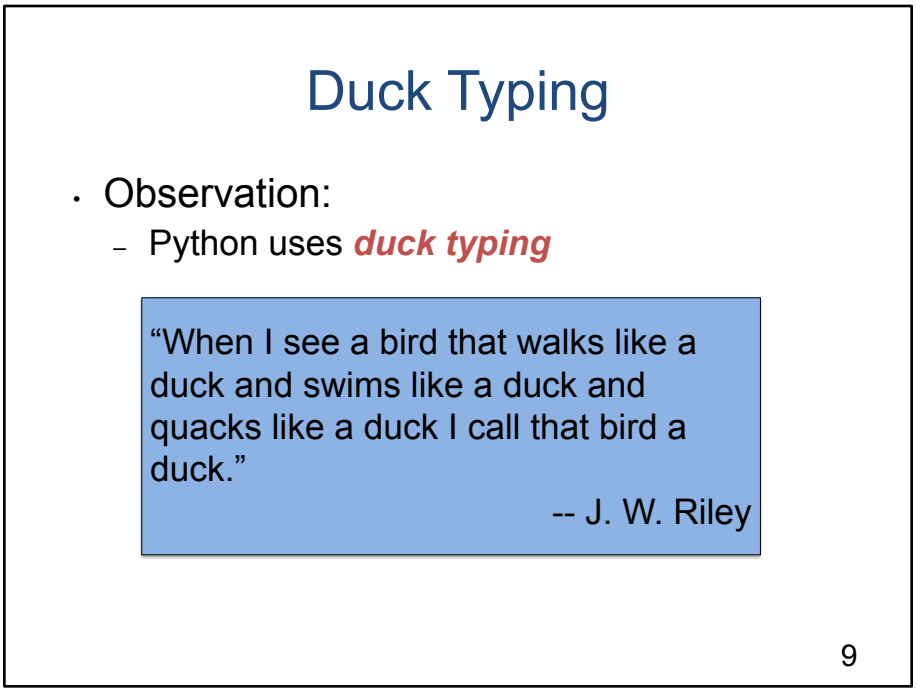
Duck Typing

See euclidstrong.py, euclidclient4.py

The job:
Same as euclid.py, euclidclient3.py

Note:
Parameter type validation

Which is better, euclid.py or euclidstrong.py?



Duck Typing

Observation:

Python doesn't care about the types of a function's parameters
Python cares only that a function's parameters respond to the messages that the function sends to them

In other words, Python uses **duck typing**

Referencing quote from J. W. Riley:

> "When I see a bird that walks like a duck and swims like a duck and quacks like a duck I call that bird a duck."

When I composed gcd(), I expected its parameters to be of type int
Suppose the parameters are not ints
If the parameters can be passed to abs(), can be compared to 0 via == and !=, and can be operands to %, then Python won't complain
If it behaves as an int, then Python is happy to consider it an int!

Duck Typing

- **Style 1**: Don't validate parameter types
  - Validating parameter types is constraining and slow
  - **So euclid.py is better**
- **Style 2**: Validate parameter types
  - Validating parameter types is safe
  - **So euclidstrong.py is better**
- We'll use Style 1

10

Duck Typing

Style 1: Don't validate parameter types

Rationale: Validating parameter types is constraining

Prohibits fortuitous use of functions/methods on parameters of unanticipated types

Rationale: Validating parameter is slow

Consumes run-time

So euclid.py is better

Style 2: Validate parameter types

Rationale: Validating parameter types is safe

When a parameter is of an unanticipated type, function/method may yield:

Errors at low call levels rather than immediately

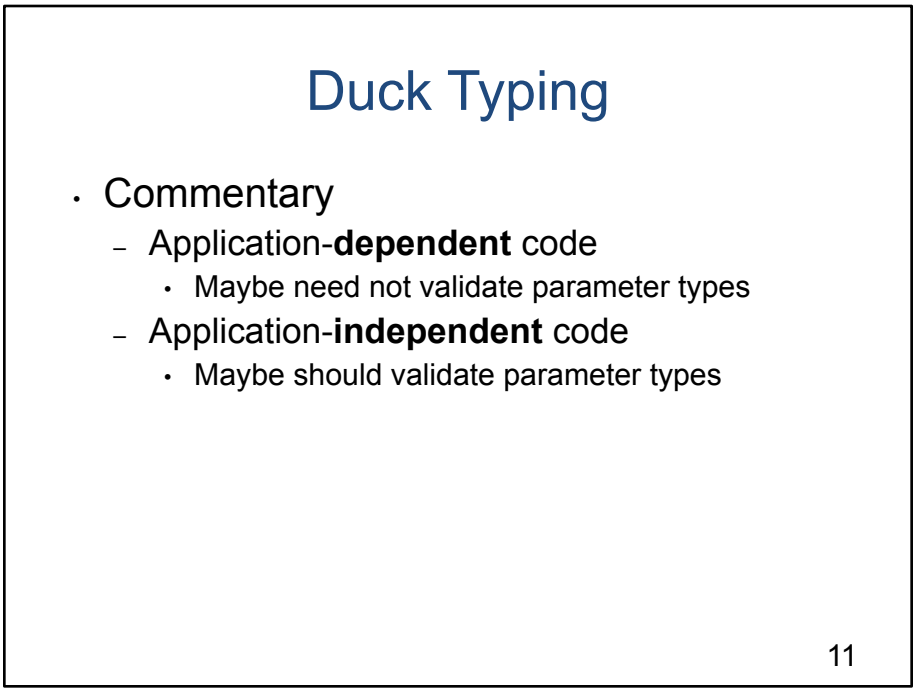
No errors and incorrect results!!!

So euclidstrong.py is better

Controversy!

Generally style 1 (do not validate parameter types) is considered more "Pythonic"

… So that's what we'll use in COS 333



Duck Typing

Commentary

Application-**dependent** code

Maybe need not validate parameter types

Application-**independent** code

Maybe should validate parameter types

The negative of accidental incorrect results outweighs the positive of fortuitous use of duck typing!!!

Duck Typing

- Commentary
  - **Small** projects:
    - Maybe need not validate parameter types
  - **Large** projects:
    - Maybe should validate parameter types

12

Duck Typing

Commentary

Small projects:
Maybe need not validate parameter types

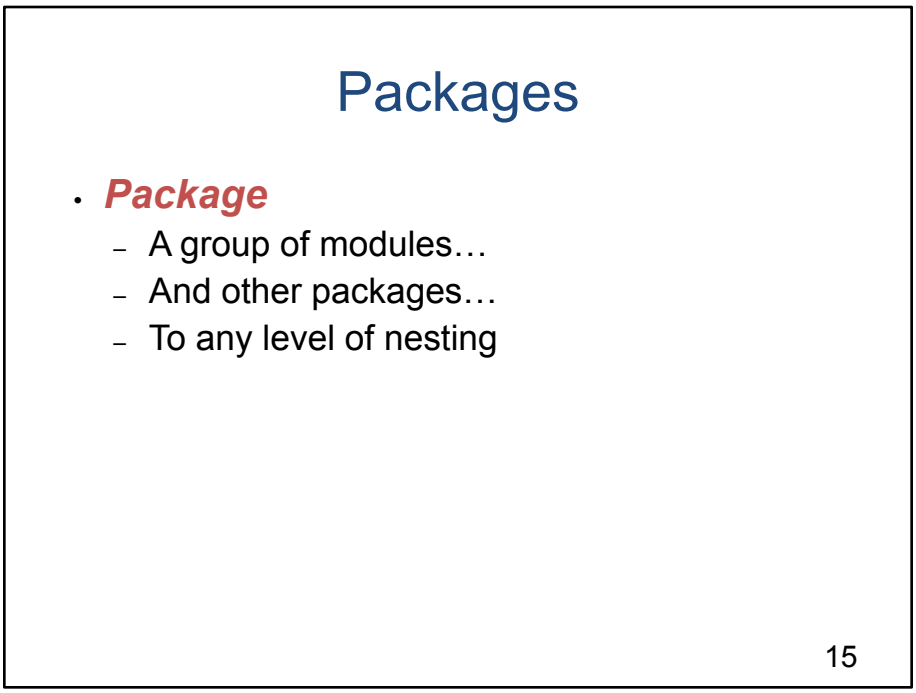
Large projects:
Maybe should validate parameter types
Safety is more important when the project is large and more programmers are involved

Duck Typing

- Commentary
  - But if you feel the need to validate parameter types, should you use Python anyway???

Agenda

- Modules
- Duck typing
- **Packages**
- Object-oriented programming



Packages

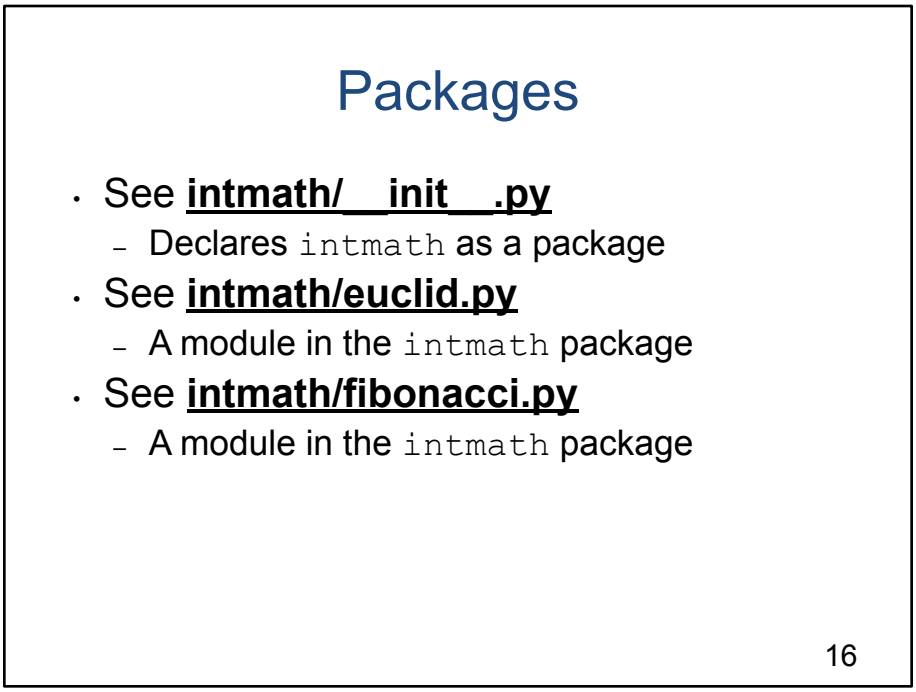
[see slide]

In Java

Class names must be unique per package, not per program

In Python

Module names must be unique per package, not per program



Packages

See **intmath/__init__.py**

Declares `intmath` to be a *package*
Contains code that python executes when package is imported
Here empty (as is often the case)

See **intmath/euclid.py**

A module in the `intmath` package
Same as previous

See **intmath/fibonacci.py**

A module in the `intmath` package
Tangential note:
Exception handling

Packages

- See **euclidclient5.py**
  - The job:
    - Same as euclidclient4.py

```
$ python euclidclient5.py
Enter the first integer: 8
Enter the second integer: 12
gcd: 4
lcm: 24
$ python euclidclient5.py
Enter the first integer: 8
Enter the second integer: 0
gcd: 8
lcm(i,j) is undefined if i or j is 0
$ python euclidclient5.py
Enter the first integer: 0
Enter the second integer: 0
gcd(i,j) is undefined if i and j are 0
$
```

17

Packages

[see slide]

Notes:

Client of the intmath package
Client of the intmath.euclid module
Imports gcd and lcm…
From the euclid module…
From the intmath package

Packages

- See **fibclient.py**
  - The job:
    - Write fib(0)…fib(9) to stdout

```
$ python fibclient.py
fib(0) = 0
fib(1) = 1
fib(2) = 1
fib(3) = 2
fib(4) = 3
fib(5) = 5
fib(6) = 8
fib(7) = 13
fib(8) = 21
fib(9) = 34
$
```

18

Packages

[see slide]

Notes:

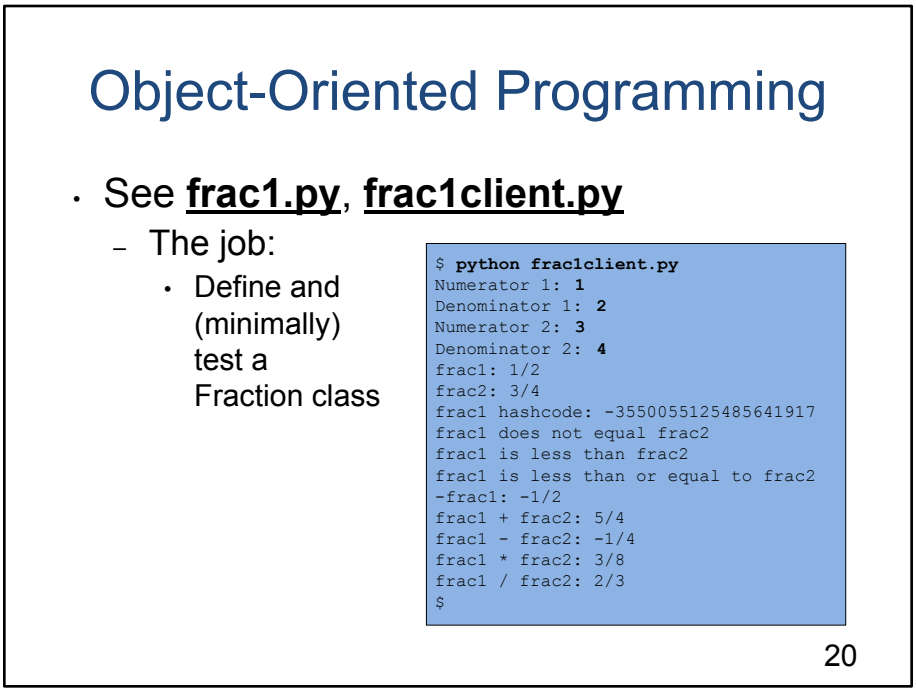
Client of the intmath package
Client of the intmath.fibonacci module
Imports fib…
From the fibonacci module…
From the intmath package

Tangential notes:

str() function
String concatenation
for statement
Easier approach using the % operator and a tuple

Agenda

- Modules
- Duck typing
- Packages
- **Object-oriented programming**



Object-Oriented Programming

[see slide]

Notes on frac1.py:

- Implicit inheritance from object class
- Constructor: __init__()
  - Creation of fields within constructor
- Explicit self parameter
  - Versus Java implicit this parameter
- **Lack of** private vs. public members
  - Use of underscore to suggest private

Notes on frac1client.py:

- Object instantiation by “calling the class”
- Method calls
  - f1.toString()
    - Argument f1 is matched with parameter self
  - f1.add(f2)
    - Argument f1 is matched with parameter self
    - Argument f2 is matched with parameter other

Aside: Python Class Dangers

- Consider this client code:
  - `frac1._num = 27`
    - Allowed
    - Corrupts `frac1`
    - Lack of “private” is dangerous
  - `frac1.num = 28  # Typo`
    - Allowed
    - `frac1` now contains three fields!!!
    - Lack of variable/field declarations is dangerous

Aside: Python Class Dangers

- Incidentally:
  - Use of leading double underscores causes ***name mangling***
    - Example: In Fraction, compiler turns `__num` into `_Fraction__num`

22

Object-Oriented Programming

Incidentally:

Use of leading double underscores causes **name mangling**

E.g.: In Fraction, compiler turns __num into _Fraction__num

That **encourages** privacy

E.g.: Client cannot assign to __num

But that still doesn't **guarantee** privacy

E.g.: Client can assign to _Fraction__num!

Object-Oriented Programming

- Fraction class defined in frac1.py
  - Not Pythonic
  - A Python programmer would use *operator overloading*
    - See next lecture

23

Object-Oriented Programming

Fraction class defined in frac1.py
- Not Pythonic
  - Composed using a Java-like style
- A Python programmer would use *operator overloading*
  - See next lecture

Summary

- We have covered these aspects of Python:
  - Modules
  - Duck typing
  - Packages
  - Object-oriented programming