

# Python Course: Lecture 3

January 9, 2006

## 1 Things I Forgot Last Time

- Update operators

```
+= -= *= \=
```

- The `type()` function tells you what type of variable you're using.

```
>>> type("s")
<type 'str'>
>>> type(1)
<type 'int'>
>>> type(1.5)
<type 'float'>
>>> type(None)
<type 'NoneType'>
```

- Functions without return values: they do something and return `None`.
- Functions can call other functions.
- Docstrings are quoted strings that appear in the source code after a function definition. They are accessible from the interpreter.
- Comments are also useful for commenting out code during debugging.
- `math.sqrt` or `**0.5` because it's needed for the homework.

## 2 Interactive Mode Survival Skills

- So how do you discover what functions a given object has?
- `dir()` command
- `help()` command
- `dir()` without arguments
- Effect of `import` on `dir()`
- `__file__` attribute of modules: `math` is defined internally, `string` is a python library. (Python libraries are good examples of Python code.)
- `__name__` attribute, useful for telling where code came from
- How you actually use the interactive mode: not just for teaching Python classes, but for language exploration and quick testing.
- If you don't know if a particular construction is supported, it's often faster to try it out in the interactive mode than look it up in the documentation.
- When I'm writing code, I always have an interpreter window open. There are lots of things I don't bother to memorize, like the functions the string type supports.
- Other use of interactive mode is running the debugger, which we'll get to later in the course.

## 3 Booleans

- You want the computer to make evaluations, decisions, so it has to distinguish between different cases
- Reduce the distinguishing task to determining whether statements are true or false. Embody this idea in the Boolean data type.
- Boolean values are `True` and `False`
- `1 + 1 == 2`, `"a" == "a"`
- Logical operators `==` `!=` `>` `<` `<=` `>=` `not` `and` `or`
- Comparison operators are defined for strings
- There is an attempt to make boolean statements read like natural text.
- `3 != 2` and `not 3 == 2` are equivalent.

- Grouping with parentheses
- Short circuiting: once the truth value of a logical statement has been determined, Python stops evaluating terms.

## 4 Conditionals

- The computer can determine whether or not certain conditions obtain using Booleans, but we need some way for it to make use of that information.
- This is done with the `if` statement—only code in the true block is executed.

```
if x >= 10:
    print "x is big"
elif x >=5:
    print "x is big"
else:
    print "x is small"
```

- This is standard programming syntax, though every language has its own version of “else if”. I’ve given up trying to memorize them, and just rely on syntax highlighting in my source editor.
- `if` statements can recurse, but note that some recursion can be promoted to logical operators

```
if x == 5:
    if y == 10:
        print "x=5, y=10"
```

is the same as

```
if x == 5 and y == 10:
    print "x=5, y=10"
```

- Any Python object can go into an `if` statement. The type gets coerced into Boolean. Different types have different logical evaluations. For example, all numbers are `True` except 0. All strings are `True` except the empty string. All lists are `True` except `[]`. `None` is always `False`.
- You can evaluate what type a variable is.

```
>>> type("a") == type("")
```

- A more self-documenting way is with the `types` module.

```
>>> type("a") == types.StringType
```

## 5 while

- Computers are good at repeating things tirelessly. So we have a repeat-a-task-until-you're-done construction.

```
x = 0
while x < 10:
    print x
    x += 1
print "all done"
```

- Note same block structure as before.
- `continue` skips to the next iteration.
- `break` leaves the loop early (this enables the occasionally useful `while True:` idiom)
- This is also standard, though every language uses a different word for `continue`.

## 6 Sample Problem: Non-divisible factors

- I'm sick of talking: we're finally to the point where you can write a simple program for practice.
- As the class goes on there'll be less of me talking and more simple program writing.
- Write a function that takes a number `n` and prints a list of all the numbers from 2 to `n-1` that do not divide it evenly, e.g. 12 prints 5,7,8,9,10,11

```
def nonfact(n):
    f = 2
    while f < n:
        if n % f:
            print f
        f += 1
```

- Consider perverse input: float, non-number, negatives, zero.

## 7 The name `==` idiom

- A common Python idiom is to have simple test code for libraries in a `__name__` block.
- For the most part the fact that Python doesn't make much of a distinction between scripts and libraries is a good thing, but sometimes you want to have code that doesn't execute when you pull it in using `import`.

```
def adder(a,b):
    print __name__
    return a+b

if __name__ == "__main__":
    print adder(3,5)

>>> import hello
>>> hello.adder(3,5)
hello
8
```

- Even non-library scripts often have very simple one or two line `__name__` blocks and do most of their work in functions which could be imported. This makes debugging in the interactive environment easier.
- You could put test code on your homeworks behind a `__name__` block. That code won't count towards the grade, but it might be helpful to you during development.