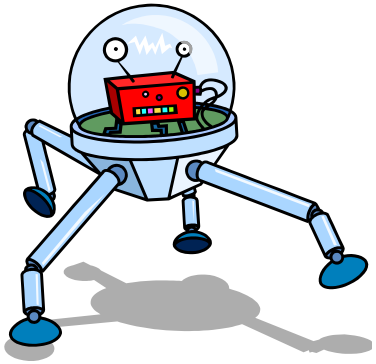
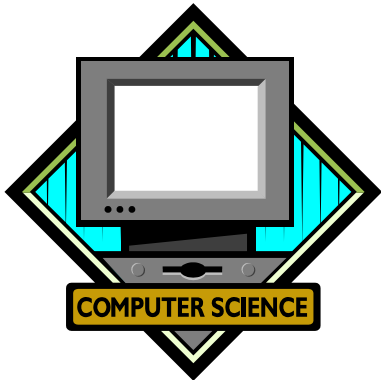
Caltech/LEAD Summer 2012 Computer Science





Lecture 2: July 10, 2012

Introduction to Python

Outline

- The Python shell
- Python as a calculator
- Arithmetic expressions
- Operator precedence
- Variables and assignment
- Types
- Functions and function definitions
- Comments

Warning!

- The code examples in this lecture are really simple and kind of boring
  - Mostly just basic arithmetic
- This doesn't mean that Python can only work with numbers!
- But numbers are the most primitive kind of data we work with, so we start there
- Ultimately, all data in a computer is represented using numbers

What is Python?

- Python is a computer programming language
- Named after *"Monty Python's Flying Circus"*
- Designed by Guido van Rossum starting in 1991
  - and continuing to this day (new versions)

Guido

LEAD
Leadership Education And Development

Guido today

Writing programs in Python

1. Write a program in a text editor (this is called *source code*)
2. Save the source code to a file on the computer's hard drive
   - (normally with a name that ends in "`.py`")
   - *e.g.* "`myprogram.py`"
3. Execute the program by running the `python` program on the file

Running programs in Python

- To run the Python program called **`myprogram.py`**, you would type this at the terminal prompt (not including the **`>`**):

```
> python myprogram.py
```

- This would run the program, and when it's done, return you to the terminal prompt:

```
>
```

Running programs in Python

- You can also run Python without giving a program name:

```
> python
```

- This brings up the Python interpreter, also known as the Python "shell"

The Python shell

- The Python "shell" is just an interactive interpreter of Python code
- It prints a *prompt* (>>>) and waits for you to enter Python source code
- Then it evaluates your code, prints the result, and prints another prompt, etc.
- We will use this a lot in our examples

Python as a calculator

```
>>> 1 + 1
2
>>> 2.2 * 3.4
7.48
>>> 1 + 2 * 3
7
>>> (1 + 2) * 3
9
```

The >>> prompt

- The >>> prompt is not part of the Python language
  - it's just the way that the Python shell tells you that it's waiting for more input
  - When you write Python code in files, there is no >>>

Arithmetic expressions

- Arithmetic expressions contain numbers (operands) combined with symbols (operators) which compute values given the numbers
- Operators: `+` `-` `*` `/` etc.
- Numbers can be integers (no decimal point) or floating-point (with decimals)

Operator precedence

- What does `1 + 2 * 3` mean?
- It could mean
  - `1 + (2 * 3)`
  - `(1 + 2) * 3`
- Computer languages have *precedence rules* to determine meaning of ambiguous cases
- Here, `*` has higher precedence than `+`, so the first meaning is correct

Operator precedence

- What does `1 + 2 * 3` mean?
- It could mean
  - `1 + (2 * 3)` **Correct!**
  - `(1 + 2) * 3`
- Computer languages have *precedence rules* to determine meaning of ambiguous cases
- Here, `*` has higher precedence than `+`, so the first meaning is correct

Operator precedence

- In general, `+` and `-` have lower precedence than `*` and `/`
  - and `=` is lower than either of them
- The `**` (power) operator is even higher precedence than `*` and `/`

```
>>> 2 * 3 ** 4
162
```

- Use parentheses to force a different order of evaluation if you need it

```
>>> (2 * 3) ** 4
1296
```

Variables and assignment

- Often, we want to give names to quantities
- In Python, use the **=** (assignment) operator to do this:

```
>>> pi = 3.1415926535897931
```

- From here on, **pi** stands for **3.1415...**

```
>>> 4.0 * pi
12.566370614359172
```

Variables and assignment

- Names assigned to can be reassigned:

```
>>> a = 10
>>> a
10
>>> a = 20
>>> a
20
```

Variables and assignment

- Not any sequence of letters is a valid name:

```
a = 10
b1 = 20
this_is_a_name = 30
&*%$2foo? = 40
```

- The first three are OK, the last not

Variables and assignment

- Names of variables ("identifiers") can only consist of the letters `a-z`, `A-Z`, the digits `0-9`, and the underscore character (`_`)
- Can have as many letters as you like
- Identifiers also cannot start with a digit
  - avoids confusion with numbers
- Identifiers can't contain spaces!
- Case of letters is significant!
  - `Foo` is a different identifier than `foo`

Variables and assignment

- Can have expressions on the right-hand side of assignment statements:

```
>>> x = 5 * 3
>>> x
15
```

- The expression is terminated by the end of the line

Variables and assignment

- Can use results of previous assignments in subsequent ones:

```
>>> y = x * 5
>>> y
75
>>> z = x + y
>>> z
90
```

Variables and assignment

- Can use results of previous assignments in subsequent ones:

```
>>> z = z + 10
>>> z
100
```

- Note: expressions like `z = z + 10` are perfectly legal!

Variables and assignment

- Evaluation rule for assignment statements:
  1. Evaluate the right-hand side
  2. Assign the resulting value to the variable on the left-hand side
- This explains why `z = z + 10` works:
  - previously, `z` was `90`
  - evaluate `z + 10` → `100`
  - assign `100` to `z` (new value)
- Variables can vary!

Types

- Data in programming languages is subdivided into different "types":
  - integers: `0  -43  1001`
  - floating-point numbers: `3.1415  2.718`
  - boolean values: `True  False`
  - strings: `'foobar'  'hello, world!'`
  - and many others

Types

- Names for types:
  - integers:
    - called "`int`" in Python
  - floating-point numbers:
    - called "`float`" in Python
  - boolean values:
    - called "`bool`" in Python
  - strings:
    - called "`str`" in Python

Types

- In Python, variables can hold data of any type:

```
a = 'foobar'
b1 = 10.3245
c_45 = 13579
some_boolean = True
```

Types

- In Python, the same variable can hold data of different types at different times:

```
>>> a = 'foobar'
>>> a
'foobar'
>>> a = 3.1415926
>>> a
3.1415926
```

Functions

- A function takes some *input data* and transforms it into *output data*
- Functions must be *defined* and then *called* with the appropriate arguments
- A few functions are built-in to Python
  - *e.g.* **`abs`**, **`max`**, **`min`**
  - ... so we don't have to define them ourselves

Functions

- Examples of function calls:

```
>>> abs(-5)
5
>>> min(5, 3)
3
>>> max(5, 3)
5
```

Functions

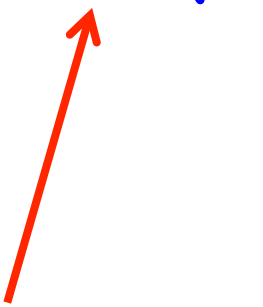
- Anatomy of a function call:

```
max(5, 3)
```

Functions

- Anatomy of a function call:

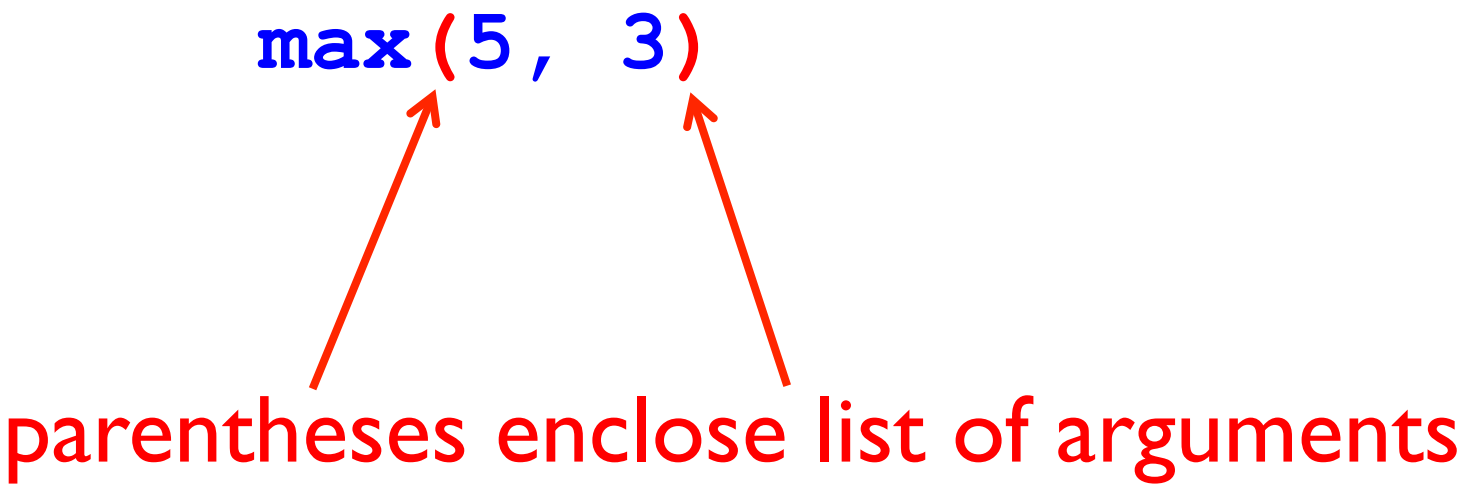
```
max(5, 3)
```



name of function

Functions

- Anatomy of a function call:



Functions

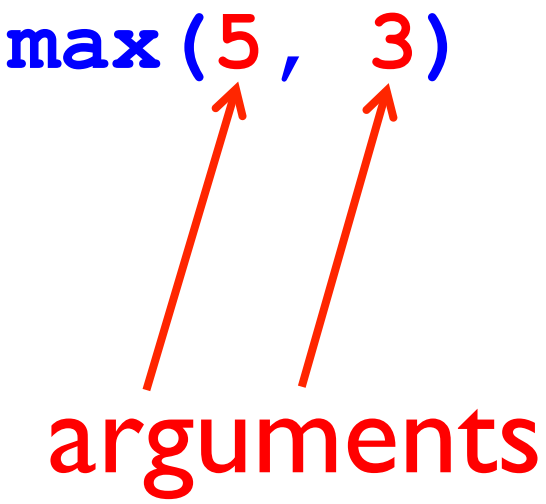
- Anatomy of a function call:

```
max(5, 3)
```

commas separate arguments

Functions

- Anatomy of a function call:



Functions

- Can have expressions as arguments:

```
>>> max(5 + 3, 8 - 6)
8
```

- Evaluation rule:
  1. Evaluate all argument expressions to get values
  2. Then evaluate the function using those values

Functions

- Can have expressions as arguments:
- `max(5 + 3, 8 - 6)`
- → `max(8, 2)`
- → `8`

Functions

- Can have function calls in expressions:
- `2 * max(5 + 3, 8 - 6) - 4`
- → `2 * max(8, 2) - 4`
- → `2 * 8 - 4`
- → `16 - 4`
- → `12`

Functions

- Can have function calls as arguments to other functions:

```
>>> max(max(5, 3), min(8, 6))
6
>>> min(2 + max(5, 3), 10)
7
```

- Evaluation rule:
  - same as before!

Function definitions

- A function *call* is done when you want to compute a particular value using that function
- If the function doesn't exist yet, you have to *define* it
- Python has a particular *syntax* to define functions
  - "syntax" means the way the language is written

Function definitions

- Example function definition in Python:

```
def double(x):
    return x * 2
```

Function definitions

- Example function definition in Python:

```
def double(x):
    return x * 2
```

- **`def`** is a *keyword* (reserved word) that introduces a function definition

Function definitions

- Example function definition in Python:

```
def double(x):
    return x * 2
```

- `double` is the name of the function we are defining

Function definitions

- Example function definition in Python:

```
def double(x):
    return x * 2
```

- Parentheses enclose the list of *formal arguments* to the function
  - Here, there is just one: **x**
- A colon (:) *must* follow the argument list!

Function definitions

- Example function definition in Python:

```
def double(x):
    return x * 2
```

- Indented lines below the `def` are the *body* of the function
  - can be just one line, or many
  - indenting is *not* optional!

Function definitions

- Example function definition in Python:

```
def double(x):
    return x * 2
```

- `return` statement is used to return the result of the function to the caller
  - `return` is another keyword in Python
  - here, `x * 2` is evaluated and returned as the result of the call to `double`

Function definitions

- Using our function definition:

```
def double(x):
    return x * 2
>>> double(42)
84
```

Function definitions

- When entering function definitions interactively into python, it looks like this:

```
>>> def double(x):
...     return x * 2
>>> double(42)
84
```

- Usually, function definitions are written directly into a file instead

Function definitions

- When entering function definitions interactively into python, it looks like this:

```
>>> def double(x):
...     return x * 2
```

- `...` is Python's *secondary prompt*
- Indicates that you're writing a function body
- Goes back to regular prompt when you're done

More function definitions

- Functions can have more than one formal argument:

```
def f(x, y):
    return (x * x + y * y)

>>>  f(2, 3)
13
>>>  f(2 + 1, 8 - 2)
45
```

More function definitions

- Functions can have *local variables*:

```
def f2(x, y):
    v1 = x * x
    v2 = y * y
    res = v1 + v2
    return res
```

- Here, **v1**, **v2**, and **res** are all local variables

More function definitions

- Functions can have *local variables*:

```
def f2(x, y):
    v1 = x * x
    v2 = y * y
    res = v1 + v2
    return res
```

- Local variables only exist for the duration of the function

Comments

- Comments are lines in a source code file that are "notes to the reader"
  - Python just ignores them
- Comments start with a **#** and go to the end of the line:

```
# This is a comment.
```

- Comments are one way to document your code
  - we'll see others as we go along

LEAD
Leadership Education And Development

Next lectures

- Strings and string processing
- Lists
- Loops
- Making decisions