CSC 120
Computer Science for the Sciences

Week 1 Lecture 2

UofT St. George

January 11, 2016

Introduction to Python
&
Foundations of computer Programming

- Variables, DataTypes, Arithmetic Expressions
- Functions
- Control flow
- File I/O
- Modules
- Packages: Numpy
- ....

Arithmetic Expressions

Symbol	Operation	Expression	English description	Value
+	addition	11 + 56	11 plus 56	67
-	subtraction	12 - 2		
*	multiplication	4 * 5		
**	exponentiation	2 ** 5		
/	division	8 / 3		
%	remainder	8 % 3		

Q: In which order does Python evaluate arithmetic (and other) operators ?

Operator Precedence in Python

Operator	Description
lambda	Lambda expression
if — else	Conditional expression
or	Boolean OR
and	Boolean AND
not x	Boolean NOT
in, not in, is, is not, <, <=, >, >=, <>, !=, ==	Comparisons, including membership tests and identity tests
|	Bitwise OR
^	Bitwise XOR
&	Bitwise AND
<<, >>	Shifts
+, -	Addition and subtraction
*, /, //, %	Multiplication, division, remainder [8]
+x, -x, ~x	Positive, negative, bitwise NOT
**	Exponentiation [9]
x[index], x[index:index], x(arguments...), x.attribute	Subscription, slicing, call, attribute reference
(expressions...), [expressions...], {key: value...}, `expressions...`	Binding or tuple display, list display, dictionary display, string conversion

Arithmetic Operators -1

Debug I/O | Python Shell

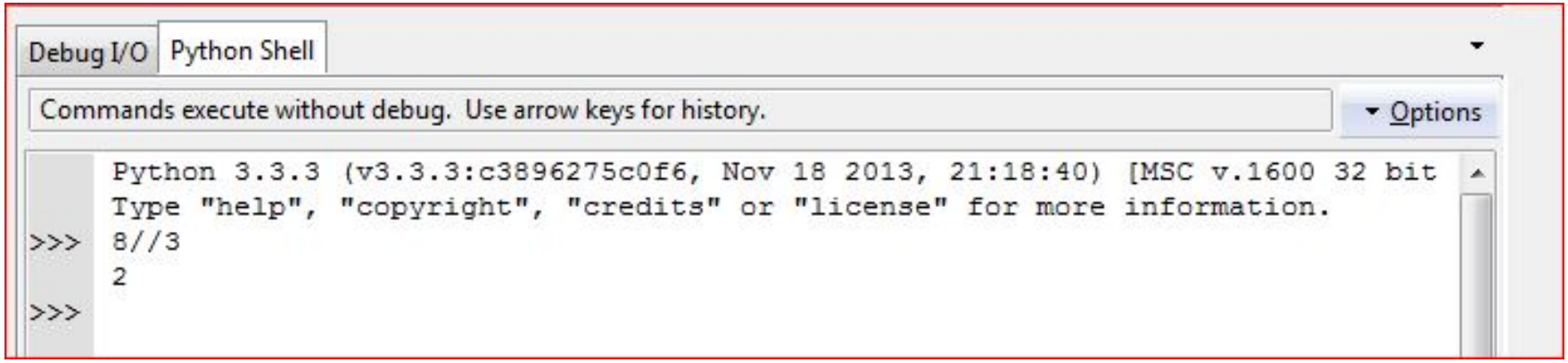
Commands execute without debug. Use arrow keys for history. Options

```
Python 3.3.3 (v3.3.3:c3896275c0f6, Nov 18 2013, 21:18:40) [MSC v.1600 32 bit
Type "help", "copyright", "credits" or "license" for more information.
>>> 8/3
2.6666666666666665
>>> 8%3
2
>>> 8/2
4.0
>>> 8%2
0
>>>
```

With 8 % 3 we get the remainder from “long division“

→ How do we get the quotient?

Arithmetic Operators -2



- What happens if we try to evaluate expressions that would not work (i.e., are undefined)?

Division by 0

Debug I/O | Python Shell

Commands execute without debug. Use arrow keys for history. Options

```
Python 3.3.3 (v3.3.3:c3896275c0f6, Nov 18 2013, 21:18:40) [MSC v.1600 32 bit
Type "help", "copyright", "credits" or "license" for more information.
>>> 8/0
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.ZeroDivisionError: division by zero
>>> 8%0
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.ZeroDivisionError: integer division or modulo by zero
>>> 8//0
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.ZeroDivisionError: integer division or modulo by zero
>>>
```

Mathematical Functions
&
The Math module

- Not all mathematical functions (e.g., sin and cos) are *built-in*.

  → to use them, we need to "import" them from the math ***module*** using **`import math`**

```
Debug I/O   Python Shell
Commands execute without debug.  Use arrow keys for history.        Options

    Python 3.3.3 (v3.3.3:c3896275c0f6, Nov 18 2013, 21:18:40) [MSC v.1600 32 bit
    Type "help", "copyright", "credits" or "license" for more information.
>>> import math
>>> math.sin(math.pi)
    1.2246467991473532e-16
>>> math.sin(math.pi/2)
    1.0
>>> math.sin(2*math.pi/2)
    1.2246467991473532e-16
>>> math.sin(2*math.pi)
    -2.4492935982947064e-16
>>>
```

Try it at home

- `help(math)`
- `help(math.sin)`
- `math.sin(90)`
- `math.pi`
- `math.sin(math.pi / 2)`
- `math.degrees(math.pi)`
- `math.radians(90)`
- `math.cos( math.radians(180) )`
- `math.fabs(-2)`
- `math.log(10)`
- `math.e`
- `math.log(math.e)`
- `math.log(10, 10)`

Side Note: floating point error

- Consider the expression: **`math.sin(math.pi)`**

- What would we expect this to evaluate to? What does it actually evaluate to?

- **Rounding error**: e.g., solving a question with a different number of decimal points than a friend, and getting a slightly different result.

- **Cause**: numbers are stored in the memory of computers in base 2 (binary), and sometimes the conversion into base 10 (decimal) adds a numerical error to the number.

  → This gets exacerbated when the number is used in calculations further on.

Python Types

- Every value has a type in Python

```
type(4.4)  <class 'float'>
type(4)    <class 'int'>
type(4.0)  <class 'float'>
type("4")  <class 'str'>

type(True) <class 'bool'>

type(4 > 6)<class 'bool'>
```

Python Casting

- There are also functions that take a value of one type and convert it to another. This is called casting:

```
int(4.3)
int(4.4)
float(4)
int("8")
str(8.5)
int(True)
bool(1)
bool(3)
```

Variables

- A **variable** is a name that refers to a value.

- Variable assignment: creates new variables and also gives it a value to refer to.

- Form of an assignment statement `variable = expression`

- How it is executed:

  1. Evaluate the expression on the right-hand side.
  2. Associate the result with the variable on the left-hand side.

Differences between Python variables and math variables

➔ Python variables look like math variables.

This could be Python or math:

```
p = 5
q = p * 7
```

➔ There are important differences!

In mathematics, equations are descriptions that are simultaneously true. In Python, assignment statements look like equations but specify a sequence of steps

Changeability

- In math, the following are inconsistent:

  `p = 5`

  `q = 7`

  `p = q + 10`

  → p cannot be both 5 and 17 !

- In Python, and other programming languages, it makes perfect sense.

  →P starts out referring to 5, then changes to refer to 17

You can change a variable's value as many times as you want, and that may change its type too.

Equality vs Assignment

- In math, p = q + 10 states a fact about the value of p and that of q + 10

  → that they are *equal*

- In Python, `p = q+10` means something completely different!!!

- This is reasonable, and common, in programming

  `x = x + 1` (*makes no sense in math*!)

  → We say that `x` is assigned `x + 1` or x gets the value of x + 1

- Programming languages usually have different symbols for assignment and equality.

  → in Python, the symbol for equality is ==

Cannot tie two variables

Q: What does this do?

```
x = 37
y = x + 2   # y is now 39
x = 20      # Is y now 22?
```

- ***One cannot use assignment to tie the values of two variables together permanently!!!***

Assignment is not symmetric

	Math	Python
`sum = a + b`	Legal	Legal
`a + b = sum`	Legal	Illegal

Naming variables

→ Rules about variable names:

- must start with a letter (or underscore): `x`, `my_average`
- can include letters, digits, and underscores, but nothing else
- case matters:

```
age = 11
aGe # Error! This is not defined.
```

→ Conventions about choosing names:

- choose meaningful names, e.g., if you are adding something up, `Sum` is better than `X` .
- for names that include multiple words, use underscore: e.g., `average_grade`

Expressions vs. Statements

- Expressions refer/evaluate to a value. Statements are a command to do something.

- Example Python expressions: (x+3)%4, not True

- Example Python statements: print(x), x = 3

- In Python, you normally write statements that either produce output or change the value of at least one variable. In the shell, you can type \statements" that are really just expressions; the shell prints the values of these expressions

Problems -1

- How can I make a variable named temp that has the value 22.3?
- How can I then cast temp into an integer?
- How can I make a variable named `Mins_in_day` that has the value of how many minutes there are in a day?

Problems -2

- What will x be, if I enter this code?

```
x= float(math.log(2,2)) + int(math.fabs(-6.7))*(8%5)
```

- What are the values of p and x after this code?

```
x = 3
p = 2
x = p ** 3
p = x // 5
```