Computing with Numbers
Zelle - Chapter 3

Charles Severance - www.dr-chuck.com

Textbook: Python Programming: An Introduction to Computer Science, John Zelle (www.si182.com)

Numbers

- Numeric Data Types and Numeric Operators - 3.1
- Using the Math Library - 3.2
- Type Conversions - 3.6
- Strings and Numbers

What does “Type” Mean?

- In Python variables, literals, and constants have a “type”
- Python knows the difference between an integer number and a string
- For example “+” means “addition” if something is a number and “concatenate” if something is a string

```
>>> ddd = 1 + 4
>>> print ddd
5
>>> eee = "hello " + "there"
>>> print eee
hello there
```

concatenate = put together

Type Matters

- Python knows what "type" everything is
- Some operations are prohibited
- You cannot "add 1" to a string
- We can ask Python what type something is by using the type() function.

```
>>> eee = "hello " + "there"
>>> eee = eee + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
>>> type(eee)
<type 'str'>
>>> type("hello")
<type 'str'>
>>> type(1)
<type 'int'>
>>>
```

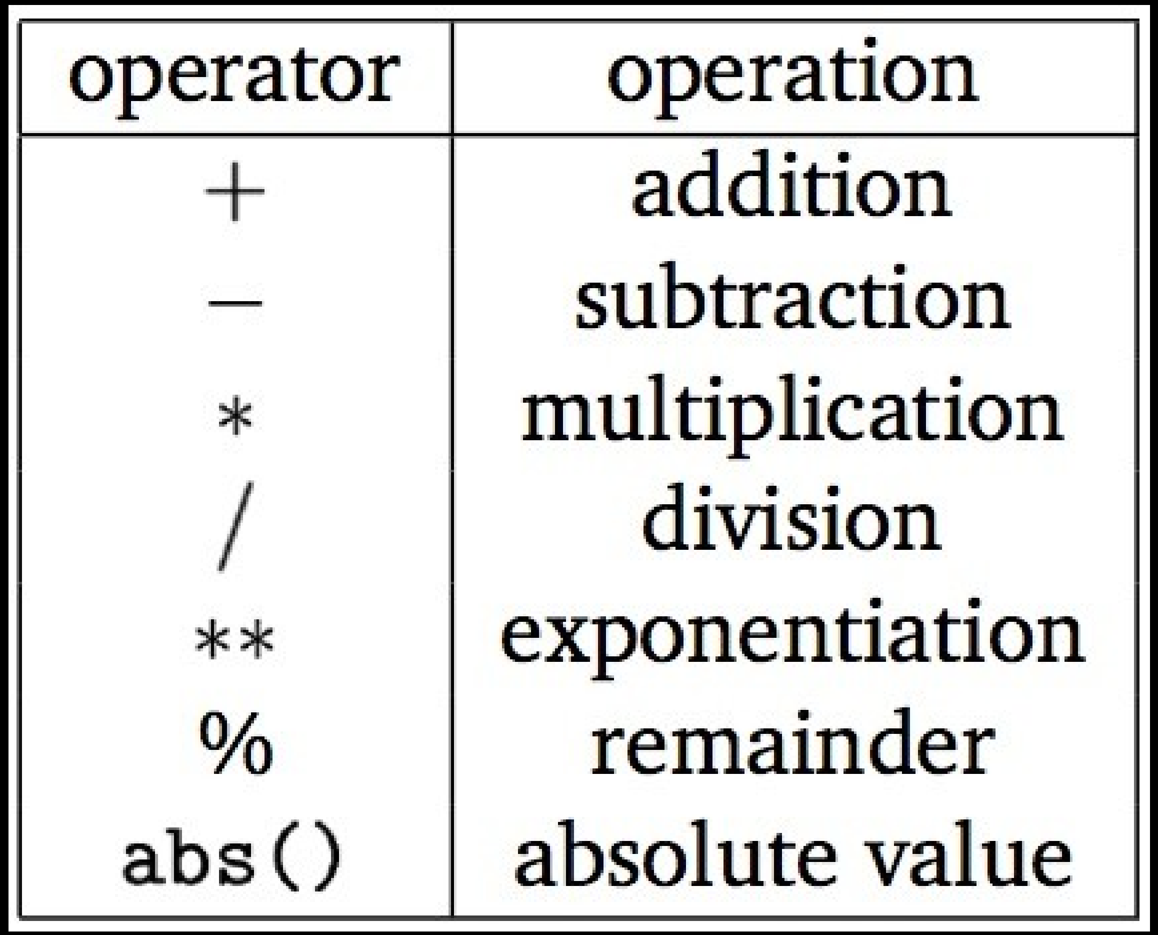
Several Types of Numbers

- Numbers have two main types
  - Integers are whole numbers: -14, -2, 0, 1, 100, 401233
  - Floating Point Numbers have decimal parts: -2.5 , 0.0, 98.6, 14.0
- There are other number types - they are variations on float and integer

```
>>> xx = 1
>>> type (xx)
<type 'int'>
>>> temp = 98.6
>>> type(temp)
<type 'float'>
>>> type(1)
<type 'int'>
>>> type(1.0)
<type 'float'>
>>>
```

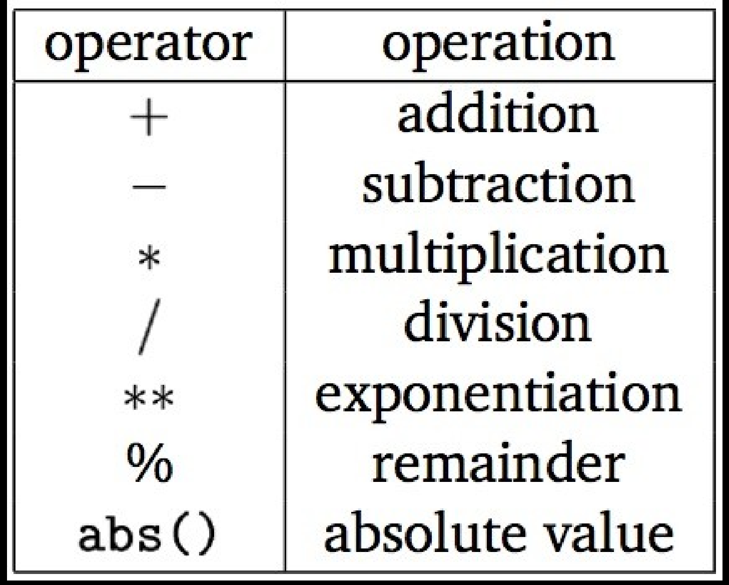
Numeric Expressions

- Because of the lack of mathematical symbols on computer keyboards - we use “computer-speak” to express the classic math operations
- Asterisk is multiplication
- Exponentiation (raise to a power) and absolute value | X | look different from in math.



operator	operation
+	addition
−	subtraction
*	multiplication
/	division
**	exponentiation
%	remainder
abs()	absolute value

Numeric Expressions



operator	operation
+	addition
−	subtraction
*	multiplication
/	division
**	exponentiation
%	remainder
abs()	absolute value

```
>>> xx = 2
>>> xx = xx + 2
>>> print xx
4
>>> yy = 440 * 12
>>> print yy
5280
>>> zz = yy / 1000
>>> print zz
5
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print kk
3
>>> print 4 ** 3
64
>>> print abs(-123.45)
123.45
>>>
```

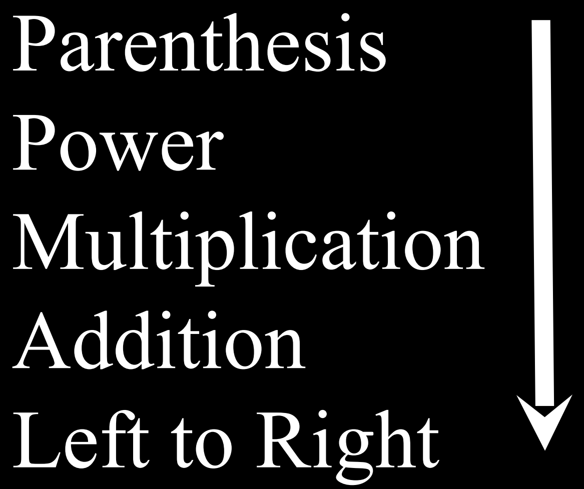
Order of Evaluation

- When we string operators together - Python must know which one to do first
- This is called “operator precedence”
- Which operator “takes precedence” over the others

```
x = 1 + 2 * 3 - 4 / 5 ** 6
```

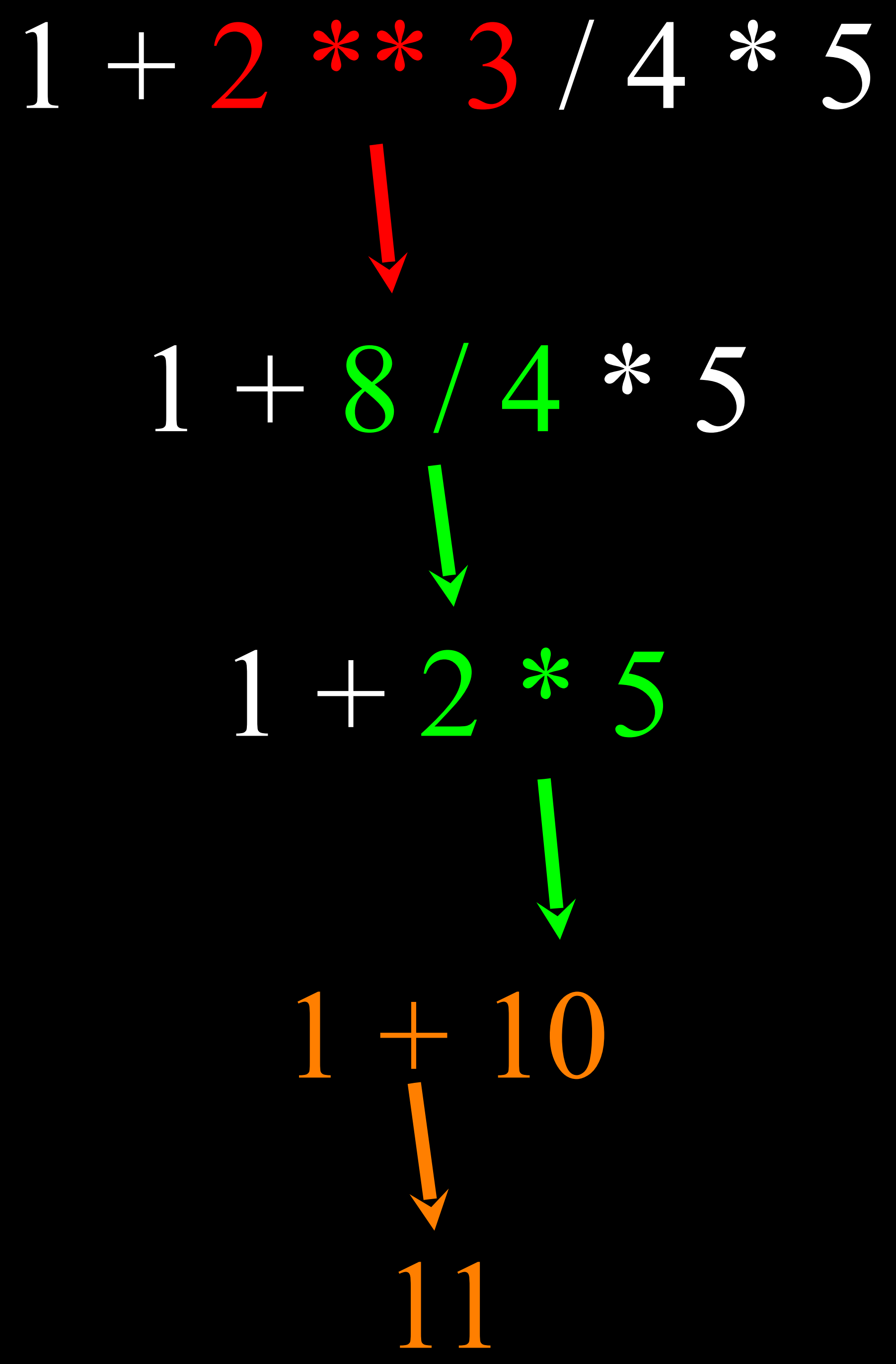
Operator Precedence Rules

- Highest precedence rule to lowest precedence rule
  - Parenthesis are always respected
  - Exponentiation (raise to a power)
  - Multiplication, Division, and Remainder
  - Addition and Subtraction
  - Left to right



Parenthesis
Power
Multiplication
Addition
Left to Right

```
>>> x = 1 + 2 ** 3 / 4 * 5>>> print x11>>>
```



1 + 2 ** 3 / 4 * 5

1 + 8 / 4 * 5

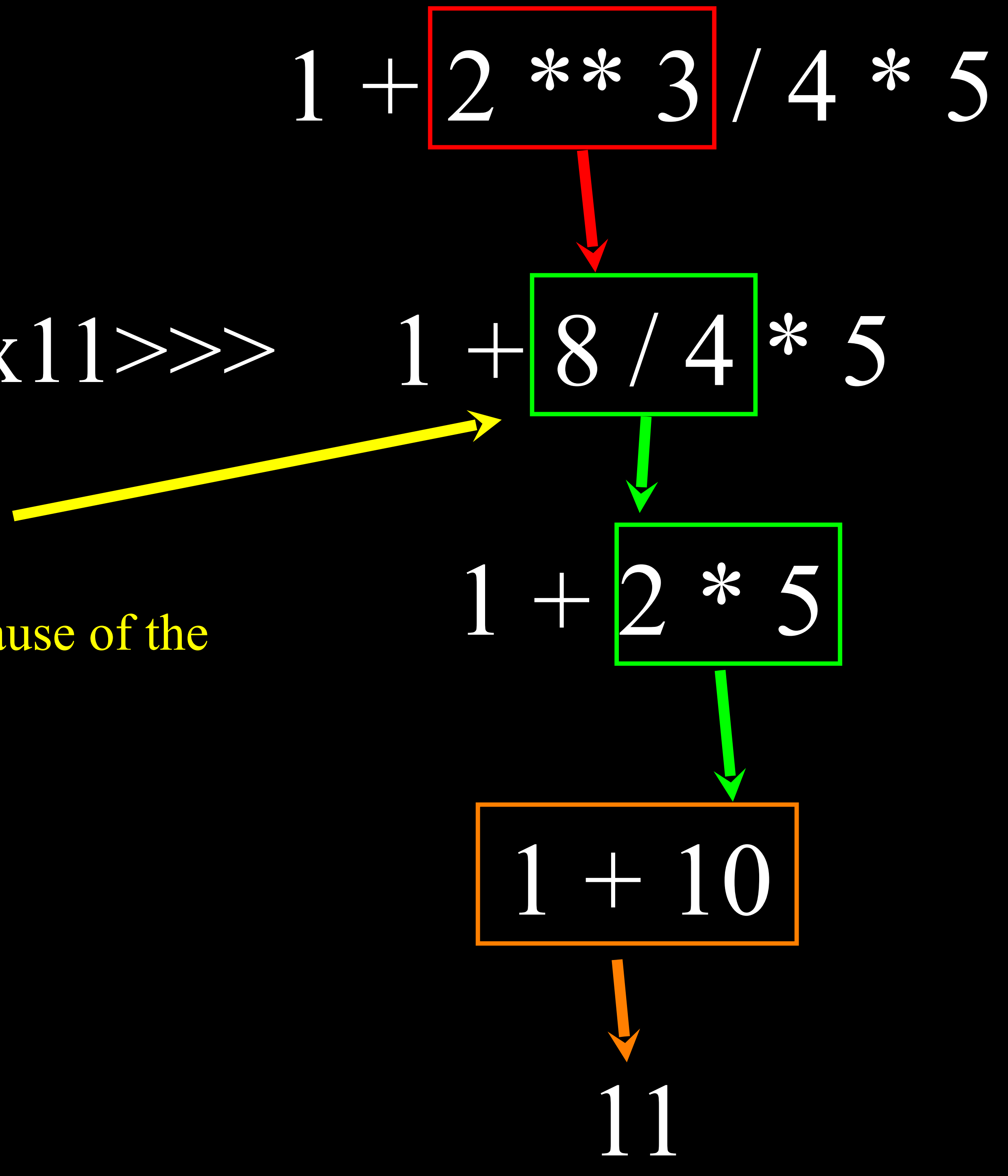
1 + 2 * 5

1 + 10

11

Parenthesis
Power
Multiplication
Addition
Left to Right

```
>>> x = 1 + 2 ** 3 / 4 * 5>>> print x11>>>
```



$1 + \boxed{2 ** 3} / 4 * 5$

$1 + \boxed{8 / 4} * 5$

$1 + \boxed{2 * 5}$

$\boxed{1 + 10}$

11

Note 8/4 goes before 4*5 because of the left-right rule.

Parenthesis
Power
Multiplication
Addition
Left to Right

Operator Precedence

Parenthesis
Power
Multiplication
Addition
Left to Right

- Remember the rules top to bottom
- When writing code - use parenthesis
- When writing code - keep mathematical expressions simple enough that they are easy to understand
- Break long series of mathematical operations up to make them more clear

Exam Question: x = 1 + 2 * 3 - 4 / 5

Integer Division

- Integer division truncates
- Floating point division produces floating point numbers

```
>>> print 10/2
5
>>> print 9/2
4
>>> print 99/100
0
>>> print 10.0 / 2.0
5.0
>>> print 99.0 / 100.0
0.99
```

Mixing Integer and Floating

- When you perform an operation where one operand is an integer and the other operand is a floating point the result is a floating point

- The integer is converted to a floating point before the operation

```
>>> print 99 / 100
0
>>> print 99 / 100.0
0.99
>>> print 99.0 / 100
0.99
>>> print 1 + 2 * 3 / 4.0 - 5
-2.5
>>>
```

Type Conversions

- When you put an integer and floating point in an expression the integer is implicitly converted to a float
- You can control this with the built in functions int() and float()

```
>>> print float(99) / 100
0.99
>>> i = 42
>>> type(i)
<type 'int'>
>>> f = float(i)
>>> print f
42.0
>>> type(f)
<type 'float'>
>>> print 1 + 2 * float(3) / 4 - 5
-2.5
>>>
```

String Conversions

- You can also use int() and float() to convert between strings and integers

- You will get an error if the string does not contain numeric characters

```
>>> sval = "123"
>>> type(sval)
<type 'str'>
>>> print sval + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int'
>>> ival = int(sval)
>>> type(ival)
<type 'int'>
>>> print ival + 1
124
>>> nsv = "hello bob"
>>> niv = int(nsv)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
```

Sneak Peek: Error Recovery

- Are you tired of seeing trace back errors?
- Do you want to do something about it?

```
>>> niv = int(nsv)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
```

- Do you want to take control of error recovery?
- Then you should take advantage of the try/accept capability in Python!

The try / except Structure

- You surround a dangerous section of code with try and except.
- If the code in the try works - the except is skipped
- If the code in the try fails - it jumps to the except section

```
$ cat notry.py astr = "Hello Bob"istr = int(astr)print "First",
istrastr = "123"istr = int(astr)print "Second", istr
```

The program stops here

```
$ python notry.py Traceback (most recent call last):  File
"notry.py", line 6, in <module>    istr = int(astr)ValueError:
invalid literal for int() with base 10: 'Hello Bob'
```

All Done

```
$ cat tryexcept.py
astr = "Hello Bob"
try:
    istr = int(astr)
except:
    istr = -1

print "First", istr

astr = "123"
try:
    istr = int(astr)
except:
    istr = -1

print "Second", istr
```

When the first conversion fails - it just drops into the except: clause and the program continues.

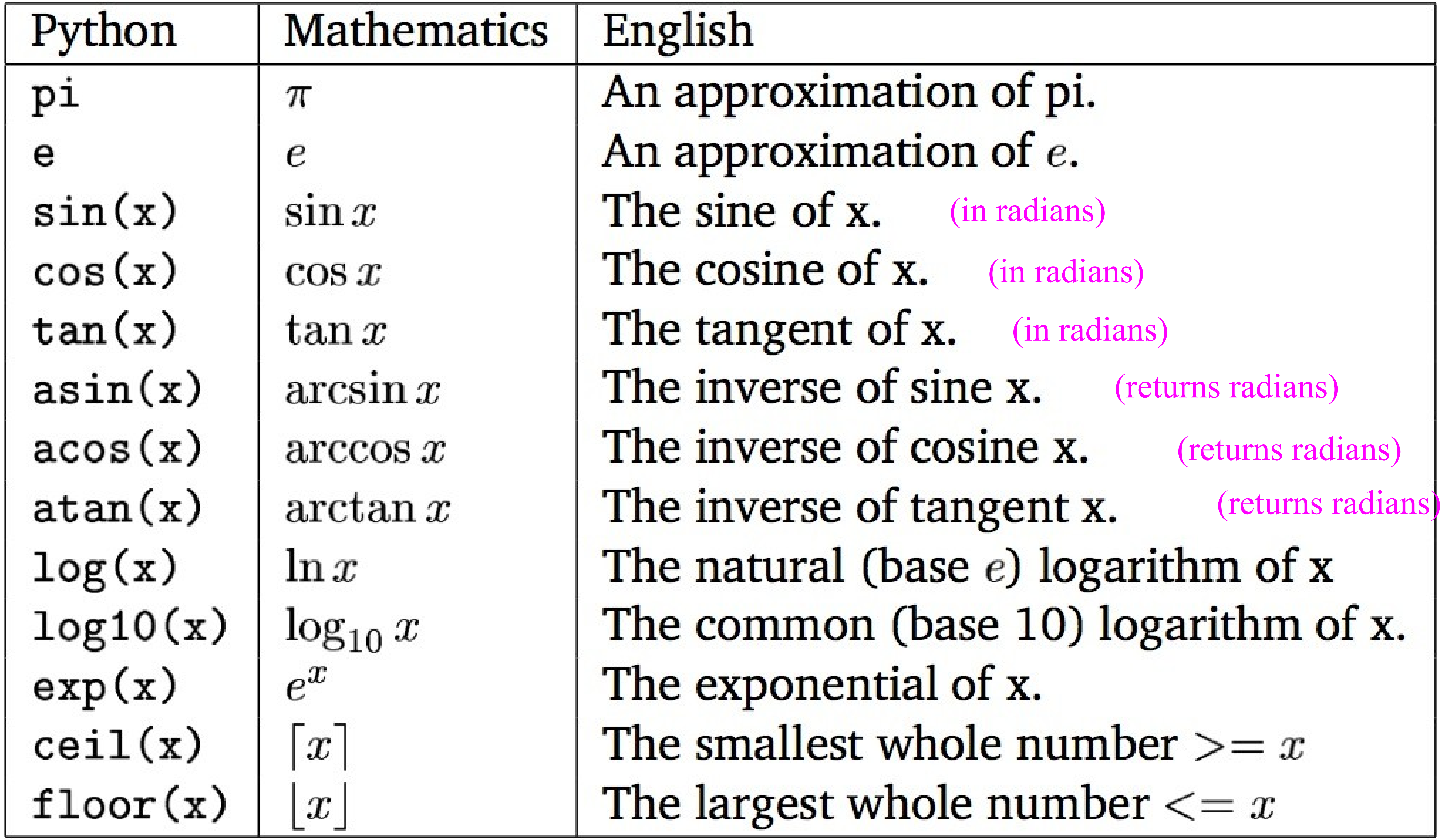
```
$ python tryexcept.py
First -1
Second 123
```

When the second conversion succeeds - it just skips the except: clause and the program continues.

Math Library

- Python also includes common math functions
- You must import math to use these

```
>>> import math
>>> print math.sqrt(25.0)
5.0
```

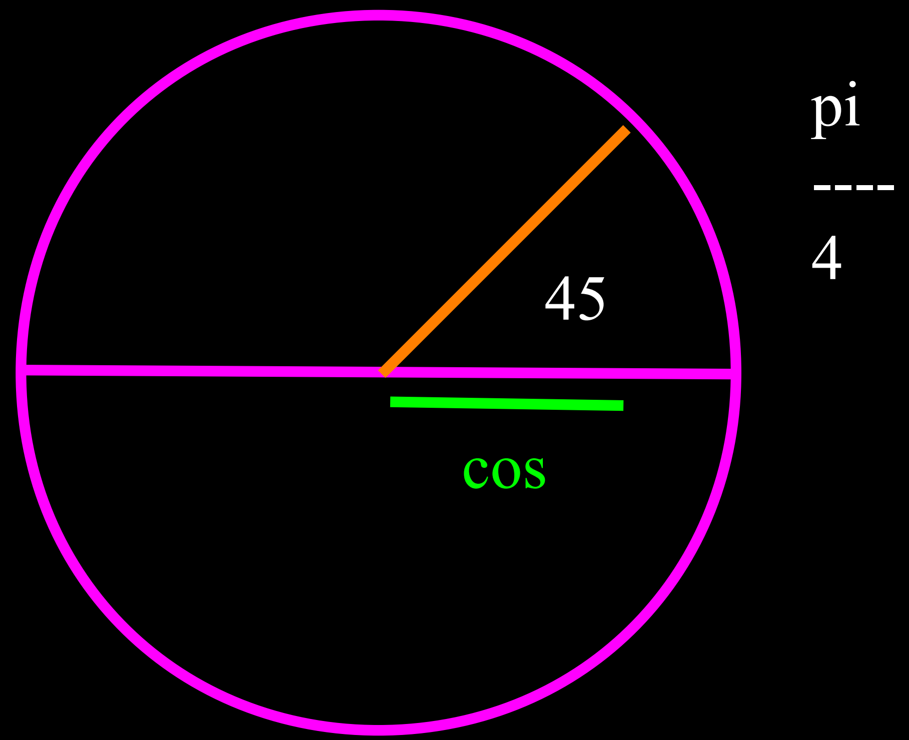


Python	Mathematics	English
`pi`	π	An approximation of pi.
`e`	e	An approximation of e.
`sin(x)`	$\sin x$	The sine of x. (in radians)
`cos(x)`	$\cos x$	The cosine of x. (in radians)
`tan(x)`	$\tan x$	The tangent of x. (in radians)
`asin(x)`	$\arcsin x$	The inverse of sine x. (returns radians)
`acos(x)`	$\arccos x$	The inverse of cosine x. (returns radians)
`atan(x)`	$\arctan x$	The inverse of tangent x. (returns radians)
`log(x)`	$\ln x$	The natural (base e) logarithm of x
`log10(x)`	$\log_{10} x$	The common (base 10) logarithm of x.
`exp(x)`	e^x	The exponential of x.
`ceil(x)`	$\lceil x \rceil$	The smallest whole number $>= x$
`floor(x)`	$\lfloor x \rfloor$	The largest whole number $<= x$

Table 3.2: Some math library functions.

Trigonometry Review

- Radians represent the length of an arc described by an angle in the unit circle (radius 1.0)
- So 45 degrees is pi / 4 or 1/8 the way around the entire unit circle (2 * pi)



```
>>> import math
>>> print math.pi
3.14159265359
>>> print math.pi / 4
0.785398163397
>>> print math.cos(math.pi / 4)
0.707106781187
```

Math Function Summary

- The math functions are there when you need them
- Unless we are solving complex trigonometry problems or statistics problems - pretty much all we use is the square root

```
>>> import math
>>> print math.sqrt(25.0)
5.0
```

Summary

- Variables, Literals, and constants have a type
- Python knows what type each object is
- Operations may work differently between types
- The common number types are floating point and integer
- We use functions to convert between strings, integers, and floats
- Peek Ahead Page 216 - We can use try / except blocks to keep our program from blowing up with bad data
- Python has rich support for common mathematical functions
- These functions are mostly useful for statistics and trigonometry
- Games use lots of trigonometry