

MCS-177
Introduction to
Computer Science I
- Spring 2014

Louis Yu



Integer and Floating Point

- Integers are the whole numbers you learned in math class. We've seen nothing but integer operations so far (even though some times the result might be floating point)
- Floating point numbers are Python's approximation of what you called real numbers in math class.
- Approximation because floating point numbers cannot have an infinite number of digits following the decimal point

```
>>> 5/3
1.6666666666666667
```

```
>>> 2.0 + 2.5
4.5
>>> 2 + 2.5
4.5
>>> 4.5 - 2.0
2.5
>>> 4.5 - 2
2.5
>>> 3.0 ** 2
9.0
>>> 3 ** 2
9
>>> 3 ** 3.0
27.0
```

Mixing integers and floats

When mixing different types of numbers, you can figure out what the result will be converted to by applying the following rules:

- If either argument is a floating-point number, the other is converted to floating point. Thus result is a floating point
- Only if both arguments are plain integers, then no conversion is needed

Operator	Description
+	Addition - Adds values on either side of the operator
-	Subtraction - Subtracts right hand operand from left hand operand
*	Multiplication - Multiplies values on either side of the operator
/	Division - Divides left hand operand by right hand operand
%	Modulus - Divides left hand operand by right hand operand and returns remainder
**	Exponent - Performs exponential (power) calculation on operators
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.

```
>>> 4 // 2
2
>>> 5 // 2
2
>>> 15 // 2
7
>>> 5.0/2.0
2.0
>>> 4.0//2
2.0
>>> 15.0//2
7.0
>>> 4 / 2
2.0
>>> 5 / 2
2.5
>>> 15 / 2
7.5
>>> 4.0/2
2.0
>>> 5.0/2
2.5
>>> 15.0/2.0
7.5
```

Mixing integers and floats

- If either argument is a floating-point number, the other is converted to floating point. Thus result is a floating point
- If both arguments are plain integers, then no conversion is needed

The expression is evaluated first (integer division), then the result is convert to float/int

Real number division: the result is a real number

Integer division: Floors the result to an integer

Operator	Description
+	Addition - Adds values on either side of the operator
-	Subtraction - Subtracts right hand operand from left hand operand
*	Multiplication - Multiplies values on either side of the operator
/	Division - Divides left hand operand by right hand operand
%	Modulus - Divides left hand operand by right hand operand and returns remainder
**	Exponent - Performs exponential (power) calculation on operators
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.

Casting

- You can also tell Python to explicitly convert a number to either an integer or floating-point by using the int() or float() function. The bracket is used to surround the expression you want to convert

- Caution: if you convert an expression. The expression is evaluated first, then the result (from the evaluation) is convert to float/int**

```
>>> 5          >>> 5.44444
5              5.44444
>>> float(5)  >>> int(5.4444)
5.0           5
```

```
float( 2 )
>>> 4 // 2    >>> float(4//2)
2             2.0
>>> 5 // 2    >>> float(5//2)
2             2.0
>>> 4.0//2    >>> int (4.0//2)
2.0          2
>>> 5.0//2.0 >>> int (5.0//2.0)
2.0          2 int( 2.0 )
>>> 4 / 2     >>> int (4/2)
2.0          2 int( 2.5 )
>>> 5 / 2     >>> int(5/2)
2.5          2
>>> 4.0/2     >>> int(4.0/2)
2.0          2
>>> 5.0/2     >>> int(5.0/2)
2.5          2
```

Casting:

- int(), float()
- Caution: if you convert an expression. The expression is evaluated first, then the result (from the evaluation) is convert to int/float**

Operator	Description
+	Addition - Adds values on either side of the operator
-	Subtraction - Subtracts right hand operand from left hand operand
*	Multiplication - Multiplies values on either side of the operator
/	Division - Divides left hand operand by right hand operand
%	Modulus - Divides left hand operand by right hand operand and returns remainder
**	Exponent - Performs exponential (power) calculation on operators
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.

Naming objects

- Very often we have an object that we would like to remember, if we want to refer to that object later.
- e.g: pi = 3.14159
- In python we can name an object using an "assignment statement"

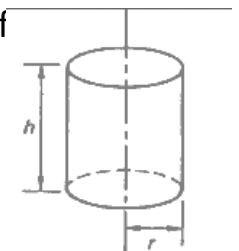
Assignment statement – this particular form of equal sign is used to assign value to a variable

Variable name = python expression

↑
variable Yes, there is another type of equal sign (which we will introduce later)

Example:

- We want to calculate the volume of a cylinder



volume = area of base * height

Let's calculate a case that:

```
r = 8.0      (radius) = 8.0
h = 16      (height) = 16
```

Note: you can pretty much name the variable ANYTHING (except a few name reserved for other purposes which we will mention later). Remember, it's just a name

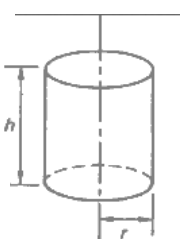
Variable name = python expression

```

Python 3.3.2 (v3.3.2:d047928ae3f6, May 13 2013, 13:52:24)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.7) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

>>> radius = 8.0
>>> height = 16
>>> pi = 3.1415926
>>> baseArea = pi * radius ** 2
>>> baseArea
201.0619264
>>> CylinderVolume = baseArea * height
>>> CylinderVolume
3216.9908224

```



$volume = \pi r^2 h$
 $volume = area\ of\ base * height$

```

>>> radius = 4.0
>>> CylinderVolume
3216.9908224 ??

```

```

Python 3.3.2 (v3.3.2:d047928ae3f6, May 13 2013, 13:52:24)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.7) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

>>> radius = 8.0
>>> height = 16
>>> pi = 3.1415926
>>> baseArea = pi * radius ** 2
>>> baseArea
201.0619264
>>> CylinderVolume =
    baseArea * height
>>> CylinderVolume
3216.9908224

```

Think Sticky Notes

Namespace	Object space
pi	3.1415926
radius	4.0
	8.0
height	16
baseArea	201.0619264
CylinderVolume	3216.9908224

Radius sticky note stick to another value, but the rest unchanged

```

>>> radius = 4.0
>>> CylinderVolume
3216.9908224

```

```

Python 3.3.2 (v3.3.2:d047928ae3f6, May 13 2013, 13:52:24)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.7) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

>>> radius = 8.0
>>> height = 16
>>> pi = 3.1415926
>>> baseArea = pi * radius ** 2
>>> baseArea
201.0619264
>>> CylinderVolume =
    baseArea * height
>>> CylinderVolume
3216.9908224
>>> x = 201.0619264
>>> CylinderVolume = x * height
>>> CylinderVolume
3216.9908224

```

Multiple "sticky notes" can stick to the same object

Namespace	Object space
pi	3.1415926
radius	8.0
height	16
baseArea	201.0619264
x	201.0619264
CylinderVolume	3216.9908224

```

Python 3.3.2 (v3.3.2:d047928ae3f6, May 13 2013, 13:52:24)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.7) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

>>> a = 10
>>> b = 20
>>> a
10
>>> b
20
>>> a = b
>>> a
20
>>> b
20

```

Think Sticky Notes

Namespace	Object space
a	10
b	20

Rules about naming things in python

- Name must start with either a letter or a _
- Python is case sensitive
 - baseArea, basearea, BaseArea are all different
- Some names are reserved by python for its own use.

```
and      del      from     not      while
as       elif    global  or       with
assert   else    if       or       with
break    except  import  print
class    exec    in       raise
continue finally is       return
def      for     lambda  try
```

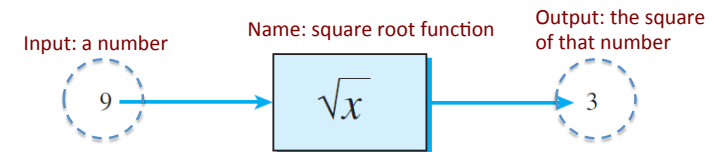
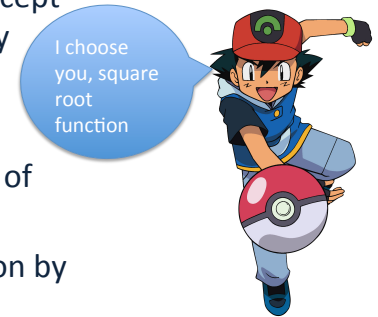
13

Module

- Many of the additional parts of Python functionality are found in **modules** – an optional part of Python that implements an abstraction that is designed to make programming easier.
- We illustrate this using the cTurtle module which allows the user to create a turtle and control it on the computer screen

Abstraction and Functions

- Abstraction is defined as a concept or idea not associated with any specific instance
- Think of it as a black box
- It's a container for a sequence of actions
- To use, you just call the function by name



The cTurtle Module

- In general, the statement you need to use to load a module is **import**
- When you import a module, an object is created inside Python, That object has the **type module** and has a name attached to it that matches the name you used on the import line

```
>>> import cTurtle
>>> cTurtle
<module 'cTurtle' from '/Library/Frameworks/Python
.framework/Versions/3.3/lib/python3.3/site-package
s/cTurtle.py'>
```

Module

- Every object in Python has 3 important characteristics:
 - A name `cTurtle`
 - A type `module`
 - A value
- Some Python objects have special values called attributes
 - Think of it as “what is the current state of the object”
- Some Python objects have methods/functions
 - You tell the object to do a particular thing
 - Go up
 - Go down
 - Calculate ____ for me

17

Turtle Methods

Name	Parameter(s)	Description
<code>Turtle</code>	None	Creates and returns a new turtle object
<code>forward</code>	Distance	Moves the turtle forward
<code>backward</code>	Distance	Moves the turtle backward
<code>right</code>	Angle	Turns the turtle clockwise
<code>left</code>	Angle	Turns the turtle counterclockwise
<code>up</code>	None	Picks up the turtle's tail
<code>down</code>	None	Puts down the turtle's tail
<code>color</code>	Color name	Changes the color of the turtle's tail
<code>fillcolor</code>	Color name	Changes the color that the turtle will use to fill a polygon
<code>heading</code>	None	Returns the current heading
<code>position</code>	None	Returns the current position
<code>goto</code>	x, y	Moves the turtle to position x, y
<code>begin_fill</code>	None	Remembers the starting point for a filled polygon
<code>end_fill</code>	None	Closes the polygon and fills it with the current fill color
<code>dot</code>	None	Leaves a dot at the current position

Table 1.3 Summary of simple turtle methods

18

How to Use a Function From a Module

- The general form is:

The module you imported

• The function you want to use from the module

(The parameter you provide for the function. It could be nothing)

The dot operator

Sometimes when you call a function, the function needs some values from you in order to do its job. The value(s) you put in is called a parameter/parameters

For example

- If you tell the turtle to move, then you need to say how far it should move
- If you tell the turtle to turn right, how many degrees it should turn

```
>>> cTurtle.Turtle()
<cTurtle.Turtle object at 0x10327a990>
```

The name of the module we imported

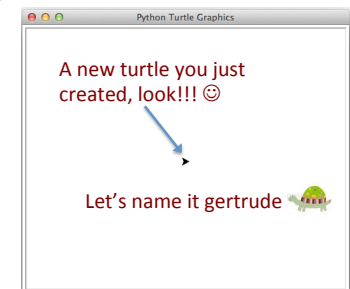
One specific function we used within that module
In this case we use a constructor to construct a turtle

```
>>> gertrude = cTurtle.Turtle()
```

The name of one specific turtle you just created (note that you can name it anything)

Definition: constructor
- Recall that the Turtle function is used to create a new turtle object

```
>>> gertrude
<cTurtle.Turtle object at 0x10174c510>
```



20

```

>>> import cTurtle
>>> turtle1 = cTurtle.Turtle()
>>> turtle2 = cTurtle.Turtle()
>>> turtle3 = cTurtle.Turtle()
>>> turtle4 = cTurtle.Turtle()
>>> turtle1
<cTurtle.Turtle object at 0x101010750>
>>> turtle2
<cTurtle.Turtle object at 0x1010107d0>
>>> turtle3
<cTurtle.Turtle object at 0x1010108d0>
>>> turtle4
<cTurtle.Turtle object at 0x101010950>

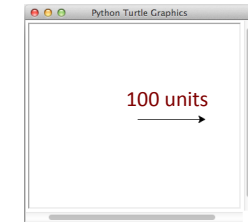
```



```
>>> gertrude.forward(100)
```

Calling a function

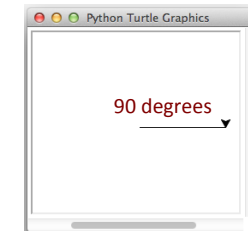
Have to provide a parameter (how far it should move)



```
>>> gertrude.right(90)
```

Calling a function

Have to provide a parameter (how many degrees it should turn right)



22

```
>>> gertrude.position()
(100.00, 0.00)
```

Returns a value

No parameter (don't forget the bracket)

```
>>> gertrude.heading()
270.0
```

- What does gertrude.forward(100) returns?

It returns no value, it simply moves the turtle around.

Important lesson learned: when calling a function, it is important to ask:

- 1) What values do we have to pass in for the function (what are the parameters)
- 2) What does the function return

You should ask that every time you call a function (practice asking yourself that often). You will need to use this skill a little later

Example – Let's Run This

```

Python 3.3.2 Shell
Python 3.3.2 (v3.3.2:d047928ae3f6, May 13 2013, 13:52:24)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.7) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

>>> import cTurtle
>>> cTurtle
<module 'cTurtle' from '/Library/Frameworks/Python.framework/Versions/3.3/lib/python3.3/site-packages/cTurtle.py'>
>>> gertrude = cTurtle.Turtle()
>>> gertrude
<cTurtle.Turtle object at 0x10317c810>
>>> gertrude.forward(100)
>>> gertrude.right(90)
>>> gertrude.forward(50)
>>> gertrude.position()
(100.00, -50.00)
>>> gertrude.heading()
270.0
>>> |

```