CSC108: Introduction to Computer Programming

Lecture 3

Wael Aboulsaadat

Acknowledgment: these slides are based on material by: Velian Pandeliev, Diane Horton, Michael Samozi, Jennifer Campbell, and Paul Gries from CS UoT

Announcements

- Assignment Style Guide.
- 24-hour silent period before the assignment due date.
- **Attend the lab you're officially signed** up for on ROSI.
  - Starting next week, you will not receive a mark for a lab unless you're officially on the roster for that lab.
  - In order for any allowances to be made, you have to e-mail the administrative TA (Lan Hui <lanhui@cs.toronto.edu>) with a legitimate reason.

What have we learnt up till now?

- Variables
- Logical & Mathematical Operators
- Assignment Statement
- Types & Type conversion
- if/else Statement
- print
- input & raw_input
- functions

Modules (revisited)

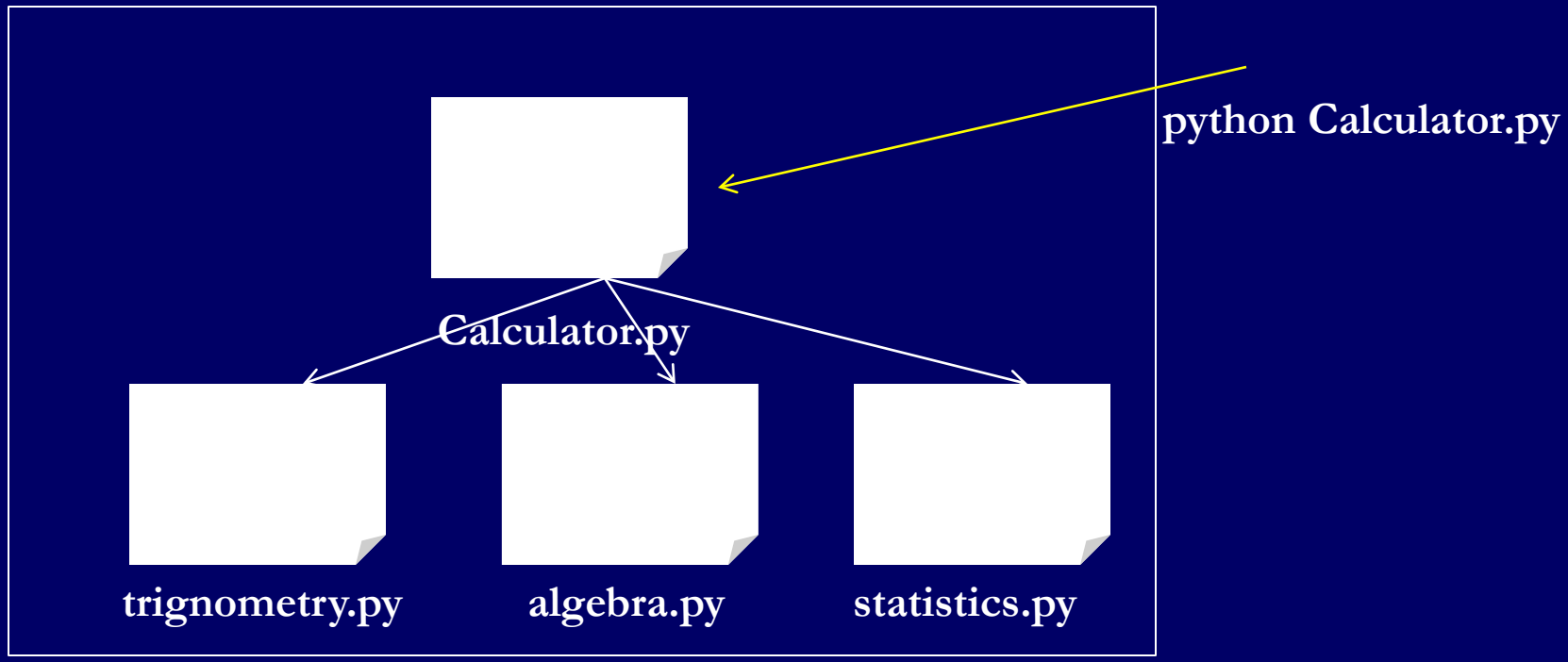
Modules

- A **module is a .py file containing a collection of functions that** have a similar purpose.

- By convention, modules should not have any code that doesn't belong to a function definition. Why?
  - Every time Python imports a module, it executes any free-floating code it finds in the file.
  - It degrades the readability of the program.

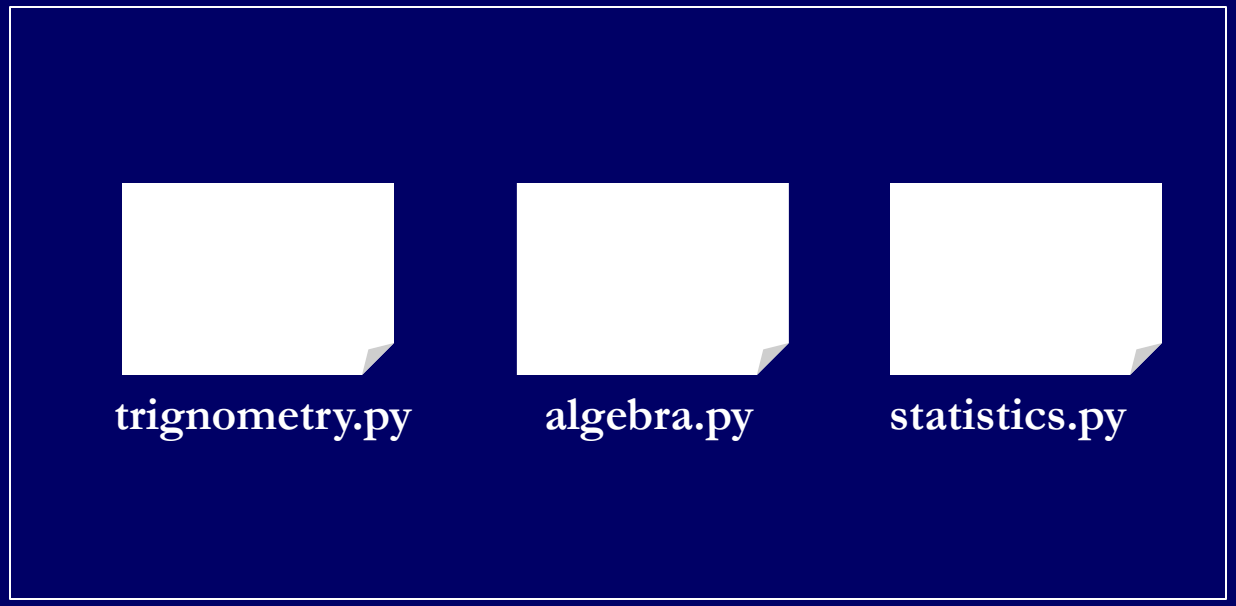
Modules

- A software application, written in Python, consists of a number of modules. One (and only one) of them will be the starting point of the application.



Modules

- You can also write a set of modules for others to use. That will not be called an application. Instead, it is called a library.



Using Existing Modules

- You have to declare that you'd like to use a module's functions by using the import statement:

  import *module_name*

- import statements should always be at the beginning of your code.

- Then, we can call a function from that module:

  module_name.function_name()

Using Existing Module example

```
import random

def roll():
    return random.randint(1,6)
```

Modules and __name__

- Python has a way of recognizing whether a module is being imported into another program or is being executed directly (as a starting point of some application)

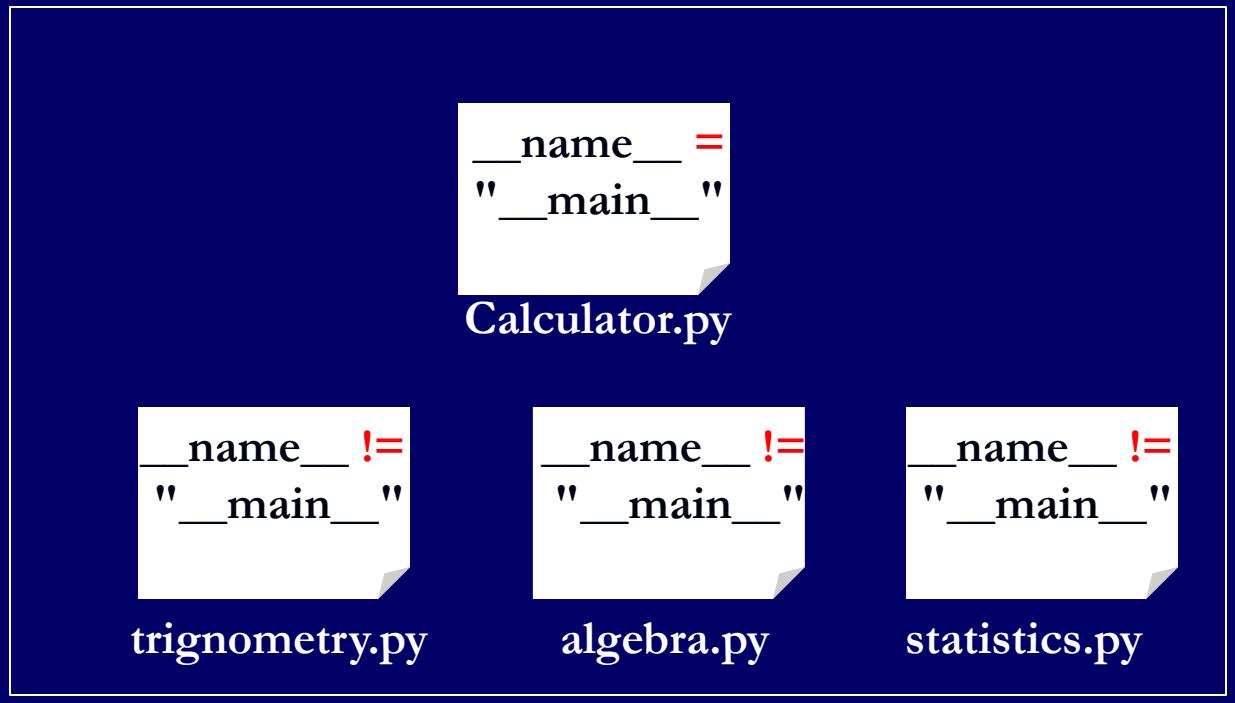
- This enables programmers to write programs both for standalone use and as reusable modules.

Modules and __name__

```
if __name__ == "__main__":
    statement1
    statement2
    statement3
```

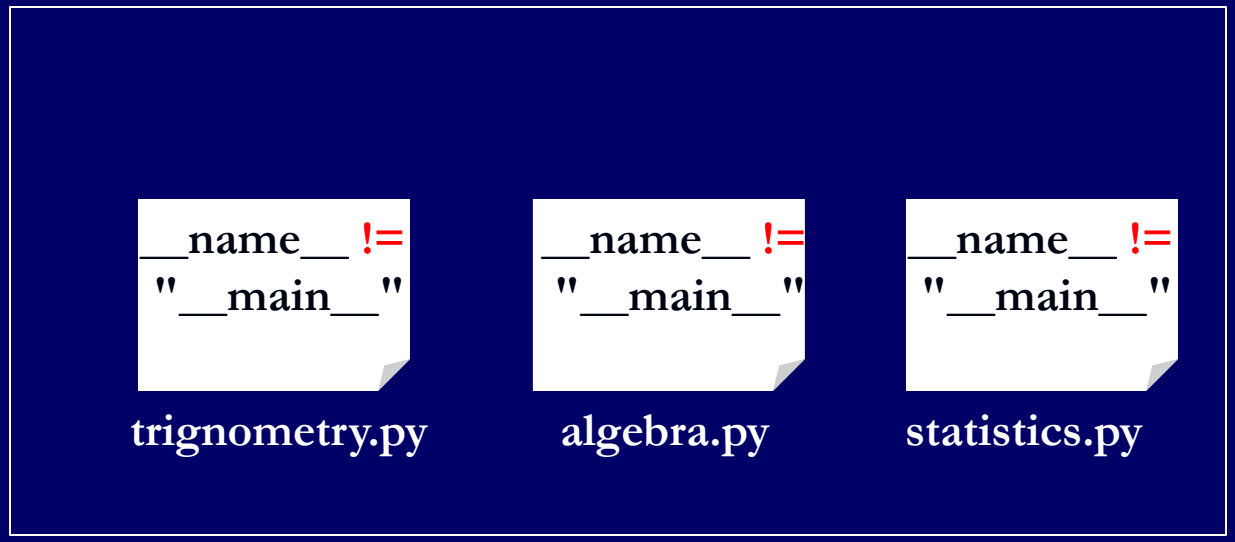
- The name of an execution instance will only be "__main__" if the module is being executed directly.

Application example



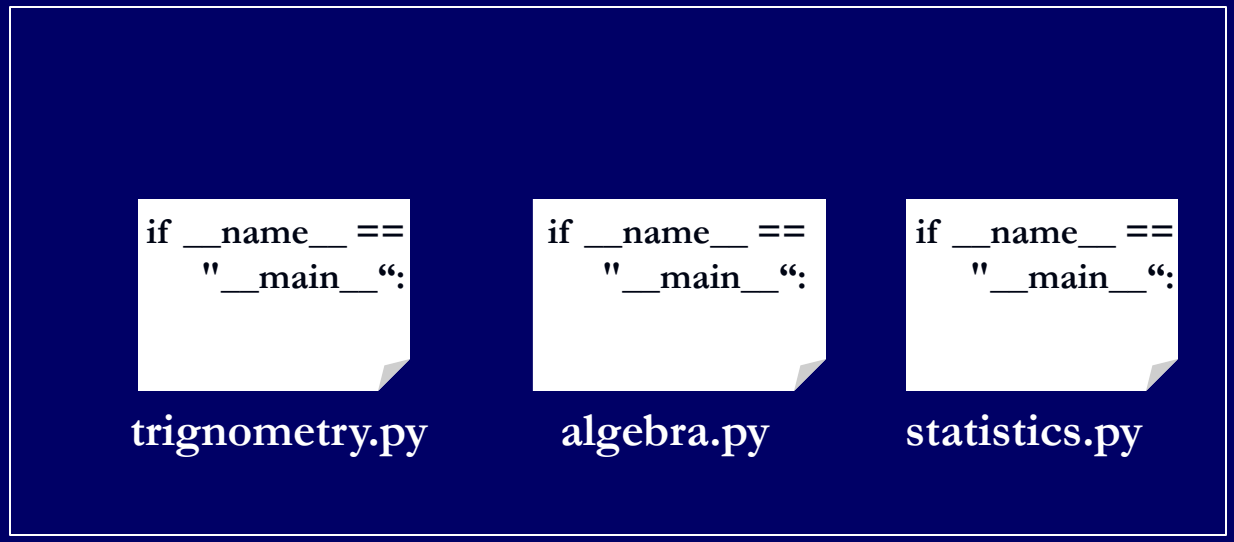
python Calculator.py

Library example



How many if __name__ == "__main__" should we have in a program?

- You guessed wrong! We are going to have it in every file!



- Why?

How many if __name__ == "__main__" should we have in a program?

- Quality Assurance (QA) or simply testing is an integral part of software development.

- In every .py file we write, we will include test cases to test the functions in that file (more about QA later…)

```
def function1
        .....
def function2
        .....
if __name__ == "__main__":
        testfunction1
        testfunction1
```

Docstrings

Docstring

- A docstring (*documentation string*) is a description of what a function does.

- Docstrings are typically specified using triple-nested single quotes:

```
'''This is a docstring'''
```

- **Every function you write should have a** docstring.

Docstring example

- A function that returns the result of a 6-sided dice roll.

```
import random

def roll():
'''Return a random integer between 1 and 6'''
    return random.randint(1,6)
```

1. Describe precisely what the function does

```
def vowels(word):
'''Returns whether the word has vowels.'''
```

True if there are vowels or False?

```
def remove_duplicates(s):
'''Removes duplicate characters from s.'''
```

Does it remove extra occurrences or all?

1. Describe precisely what the function does

```
def vowels(word):
    '''Return True iff the string word has vowels. Do not treat
    Y as a vowel.'''
```

```
def remove_duplicates(s):
    '''Return the string s, except only the first occurrence of
    each character in s is kept.'''
```

2. Do not reveal *how the function* does it.

```
def add_border(pic, c):
    '''Add a border to pic by adding 4 overlapping rectangles.'''
```

3. Make the purpose of every parameter clear.

```
def add_border(pic, c):
    '''Add a border to pic.'''
```

```
def add_border(pic, c):
    '''Add a 20-pixel wide border of colour c to picture pic.'''
```

4. Refer to every parameter by name.

```
def average_red(pic):
    '''Compute the average amount of red in a picture.'''
```

```
def average_red(pic):
    '''Compute the average amount of red in picture pic.'''
```

5. Be clear about whether the function returns a value (and if so, what)

```
def average_red(pic):
    '''Compute the average amount of red in picture pic'''
```

```
def average_red(pic):
    '''Return the average amount of red (a float) in the pixels
    of picture pic.'''
```

6. Explain any conditions that the function assumes are true.

```
def speed(d, t):
'''Return a car's speed.'''
```

Does the function assume that time must not be zero? Negative?

```
def speed(d, t):
'''Return the speed (a float) of an object that travels
distance d in time t. d and t are ints. t is non-zero. '''
```

7. Write as a command rather than a statement.

```
def cube(x):
'''Returns the cube of x'''
```

```
def cube(x):
'''Return the cube of x'''
```

8. Use iff in boolean functions for precision

- Boolean functions are ones that return True or False.

- E.g.

```
def is_odd(n):
'''Return True if and only if integer is odd'''
```

how does Python know what a function does when help() is called?

- Python looks for the first free-floating docstring after the function definition and makes that the output of help().

- This is how programmers communicate information about the code.

Iteration: while loop

The while Loop

- Executes a group of statements as long as a condition is True.

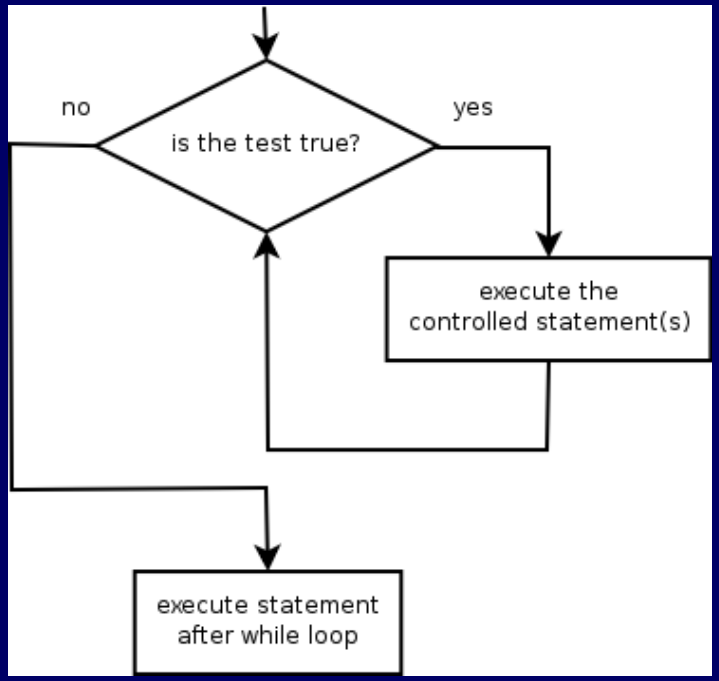
- Syntax:

  while ***condition*** :

  statements

- Example:

```
number = 1
while number < 200:
    print number,
    number = number * 2
```



The while Loop: Uses

1) To repeat an action indefinitely

2) To repeat an action as many times as is necessary for some condition to be met

3) To repeat an action a set number of times Modules

The while Loop: endless repition

- To repeat an action forever, set the condition of the while loop to something that will always be True:

```
x = 0
while True:
    x  = x + 1
      print "Cycle", x
```

- **Examples:**
  - Screen refresh

The while Loop: conditional repetition

- To repeat an action until a desired condition is reached, make sure the outcome of the condition has an opportunity to change while the body of the while loop executes:

```
while x < 1 or x > 1000:
        print "Number out of range"
        x = int(raw_input("Number: "))
```

- **Example:**
  Password prompts

The while Loop: bounded repetition

- To repeat an action a predetermined number of times, have the condition check a variable's value and increment/decrement it by 1 inside the while body:

```
count = 1
while count <= 10:
        print count
        count += 1
```

- **Example:**
  Mathematical calculation

The while Loop: 3 things to remember

1) to initialize the loop variable before the loop

2) Check on the loop variable in the while condition

3) Increment/decrement the loop variable in the body of the loop

Nested while Loops

- Example:

```
x = 1
y = 1
while x < 10:
    print x
    x = x * 2
    y = 1
    while y < 5:
        print y
        y = y + 1
```

Iteration: for loop

The for loop

- **for loop**: Repeats a set of statements over a group of values.

- Syntax:

  for *variableName* in *groupOfValues*:

  statements

  - ***variableName*** gives a name to each value, so you can refer to it in the ***statements***.
  - ***groupOfValues*** can be a range of integers, specified with the `range` function.

```
myList = [1,2,3,5,6]
for x in myList:
    print x
```

Iteration: Strings

Strings

- A sequence of text characters in a program.
  - Strings start and end with quotation mark " or apostrophe ' characters.
  - Examples:

    "hello"
    "This is a string"
    "This, too, is a string. It can be very long!"

- A string may not span across multiple lines or contain a " character.

  "This is not
  a legal String."

  "This is not a "legal" String either."

Strings

- A string can represent characters by preceding them with a backslash.
  - \t tab character
  - \n new line character
  - \" quotation mark character
  - \\ backslash character
  - Example: "Hello\tthere\nHow are you?"

String Index

- Characters in a string are numbered with *indexes* starting at 0:
  - Example:

```
name = "P. Diddy"
```

index	0	1	2	3	4	5	6	7
character	P	.		D	i	d	d	y

- Accessing an individual character of a string:

variableName `[` ***index*** `]`

```
print name, "starts with", name[0]
```

String Index – negative values

- Negative indices count backward from the end of the string
  - x[-1] is the last character
  - x[-2] is the second-last character

- Example:

```
val = "carbon"
print val[-2], val[-4], val[-6]
```

String Example

What have we learnt today?

- Modules
- Docstrings
- While/for loops
- Strings

This Week's To Do List

- Go through lecture slides – make sure you try the code snippets
- Try the lecture's programs posted on course website
- Start working on the assignment if you haven't already!...