

01204111 Computer and Programming: Lab 1

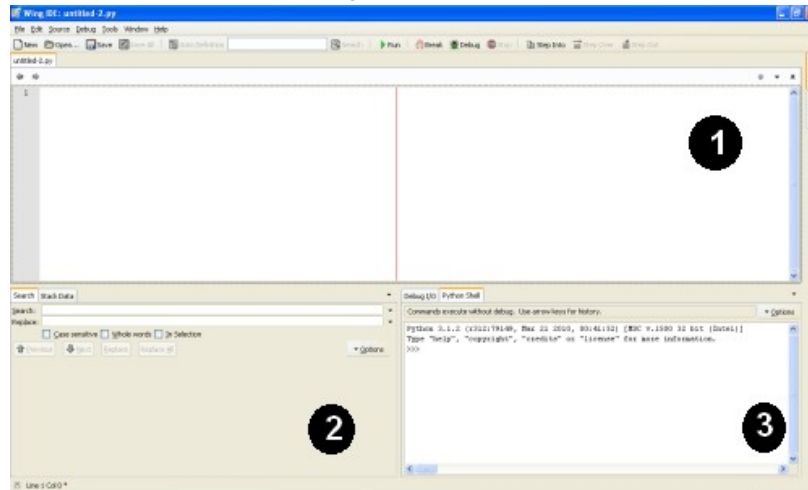
Name _____ ID _____ Section _____

Goals

- To get familiar with Wing IDE and learn common mistakes with programming in Python
- To practice using Python interactively through Python Shell
- To learn Turtle Graphics

Part 1: Welcome to Wing IDE 101

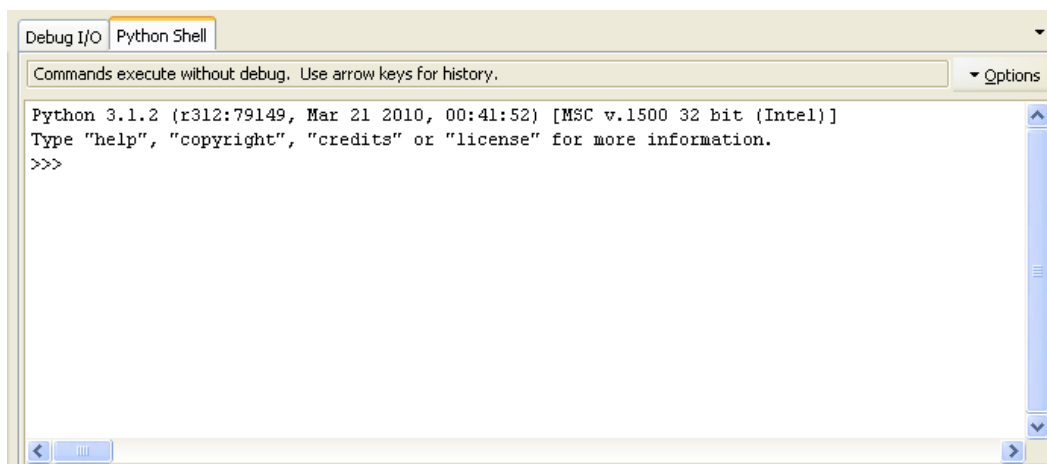
When we start Wing IDE 101, we will see the following window.



The window is divided into 3 areas:

- Area 1 is for typing and editing programs. If this area is empty, you can click the "New" button at the toolbar to start editing a new program.
- Area 2 and 3 are tool areas. It is possible that you only have one tool area. Usually in area 3, there is a tab called **Python Shell**, we will interact with Python on that tab.

The Python Shell is shown below. Note that we choose tab "Python Shell" (shown with orange line), not the "Debug I/O" tab.



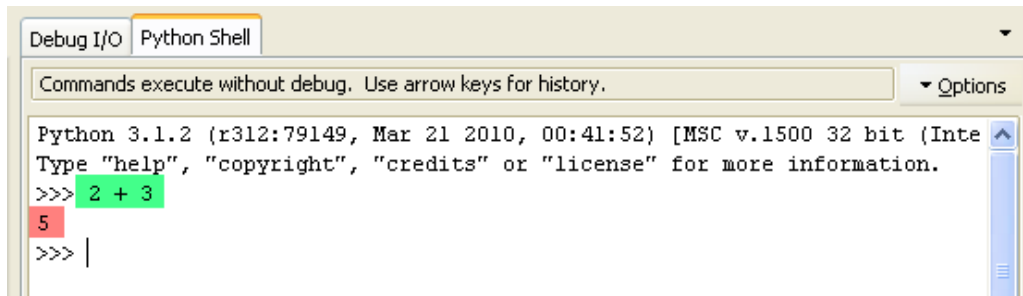
We shall practice Python, mainly by typing commands into this Python Shell. The goal is to get you familiar with the IDE and the syntax of Python. What we try today will be covered in more details later on in the course.

1.1 Python, interactively

Type the following command into the Python Shell (after the ">>>" sign).

2 + 3

After you press Enter, you will see the following output. (What you typed is shown in green; the output is shown in red.)



We will follow this convention when showing interactions with Python: the line that you enter is shown beginning with ">>>" in bold types, and the output from Python is shown in italics. See example below.

```
>>> 2 + 3          # This is the line that you typed.  
5                # This is the output from Python.
```

When using Python Shell, the Python interpreter reads what we type and then *evaluates* it. If what we type is an expression, the interpreter will calculate the value and output the result.

The expression can be just a constant:

```
>>> 100  
100
```

or can be something more involved:

```
>>> 1+2+3+4+5+6+7*100-30  
691
```

or even:

```
>>> 1/2 + 1/4 + 1/8 + 1/16  
0.9375
```

In the next section, you should experiment with Python Shell by typing the commands specified, in the given order.

1.1.1 Parentheses

We can use Python as a calculator. Try this:

```
2 + 3 * 5
```

Or this:

```
(2 + 3) * 5
```

Can you tell the reason why the two expressions evaluate to different results?

Consider the following expression. What should it evaluate to? Try to guess the value before typing into Python Shell.

```
2 + (3 * 5)
```

1.1.2 Variables

After an expression is evaluated, we can *reuse* its value again without having to type it in again by assigning the value to a *variable*.

We can assign a value to a variable using operator =, for example:

```
g = 9.81
```

When we assign a value to a variable, we tell the variable to *refer* to that value. We can illustrate this with the figure below.



Notes: You do not need spaces between variable and the expression, or between the variable and the operator. However, we suggest you do so for readability.

After assigned, the variable can be used as follows:

```
>>> g
9.81

>>> 10 * g
98.10000000000001
```

Notes: Operator * is for multiplication. The precision of numbers on a computer is limited; therefore, you can see small errors on the output.

We call a variable "variable" because we can vary (change) its values.

```
>>> g
9.81
>>> g + 5
14.81
>>> g = 100
>>> g + 5
105
```

Consider the following interactions. (Please experiment as well.)

```
>>> a = 100
>>> b = a + 10
>>> b
110
>>> a = 200
>>> a + 10
210
>>> b
110
```

What can you conclude from this experiment. How does b change in terms of a?

Try the following 3 commands.

```
>>> x
>>> x = 10
>>> x
```

Explain why you got that result.

Try the following commands:

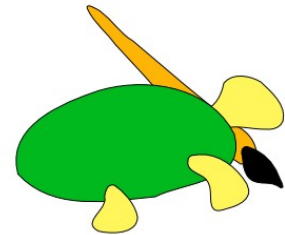
```
x * 5
y = 20
y * x
y = x
y * x
y = y + 1
y * x
```

Explain what the command "y = y + 1" does.

Part 2: Turtle Graphics

In this part, we shall experiment with Python by writing a program to draw pictures. We shall program a Turtle to walk around on the canvas. This turtle carries a pen while walking, so a picture is drawn as we tell it to move.

The most important thing for this part of the lab is to be brave! Don't be afraid of making mistake.



2.1 Here comes the turtle

We work mainly on the Python Shell. However, this tab is quite small, so before getting started, you should enlarge that area by dragging its borders so that the Wing IDE looks like the following figure. After adjusting the tab size, you should resize the whole Wing IDE windows to roughly 1/2 of the screen. (If the Wing IDE windows is "maximized," don't forget to "restore" it before trying to resize the window. See example on the right.

We will start using Turtle graphics by importing the `turtle` module. Type the following to the Python Shell.

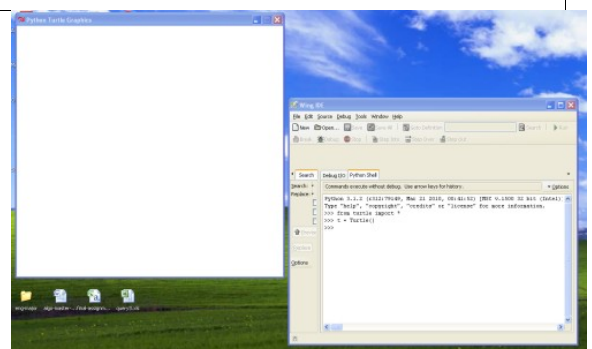
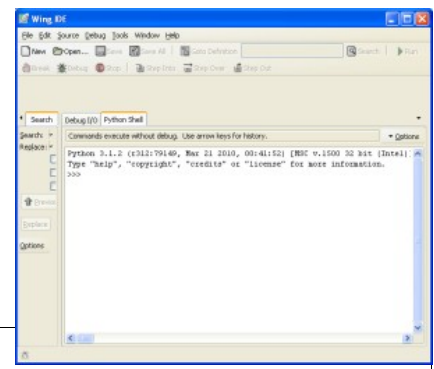
```
from turtle import *
```

We then *create* the turtle and assign it to variable `t` using the following command.

```
t = Turtle()
```

After that, a window titled, "Python Turtle Graphics," would appear. It may takes some time before its appearance. It may appear behind other windows; if that is the case, select the window so that it moves to the front.

Then, try to resize both windows (Turtle Graphics and Wing IDE) so that you can see both windows. See example on the right.



Try the following command in the Python Shell.

```
t.forward(100)
```

If you see a line to the right (a result from the turtle move) with a small arrow (representing the position and the direction of the turtle).

Note: If you have to close Wing IDE or restart the machine, you should type the following two commands to open the Turtle windows before you can continue working.

```
from turtle import *  
t = Turtle()
```

2.2 Experiments with turtle command

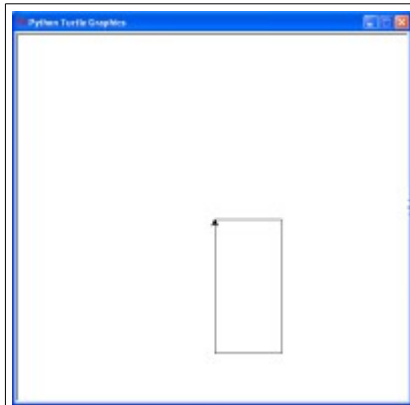
Try the following commands one by one and observe how the turtle works.

```
t.right(90)  
t.forward(200)
```

What do commands `forward` and `right` do?

Commands **forward** and **right** are *messages* that we send to the turtle. When we mention these messages we usually leave the `t.` part from the names of the message. However, when we issue the command, we have to refer to the exact turtle we are sending the methods to; thus, we have to prefix the command with `t.`

Try to tell the turtle to move around so that a rectangle of size 100 x 200 is drawn on the canvas.



Write the commands that you use to draw here.

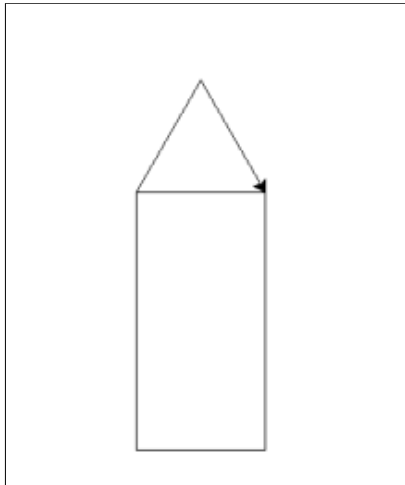
If you make a mistake: Don't be afraid that you may get extra lines. In this lab, having more lines is better than having incomplete pictures.

If we tell the turtle to `forward` too far, we can use the method `backward(distance)` to tell the turtle to move backwards. Similarly, we can use method `left(angle)` to tell the turtle to turn left. If you want to start over, you can use method `clear()` to delete everything. You can also use method `home()` to tell the turtle to move to the origin position.

Don't forget that to use these method, we have to refer to the turtle, i.e., `t.backward(100)`.

Tell the turtle to draw a house as in the figure below. (Note: it took some time before the lab author can get it drawn and the author has to start over many times. Don't forget that it is okay to have extra lines.)

Hints: an equilateral triangle has inside angle of 60 degrees. (But you do not need to make an equilateral triangle.)



Write the commands that you use to draw here.

2.2.1 Summary of methods

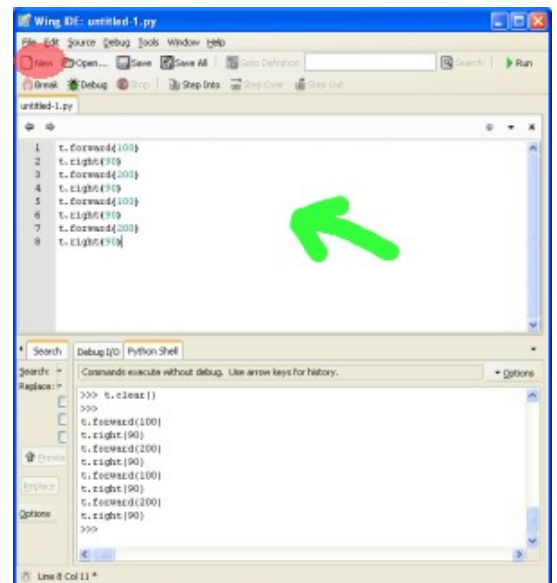
- `forward(distance)` and `backward(distance)` ---- tell the turtle to move forward or backward.
- `left(angle)` and `right(angle)` ---- tell the turtle to turn left or right.
- `home()` ---- tell the turtle to go home (the original position with the original head direction).
- `clear()` ---- clear all drawing.

2.3 A better way to enter long list of commands

Issuing a long list of commands in Python Shell is fairly difficult because it is easy to make mistakes and correcting them is very hard. It is better to type in commands into the editor and then copy them to the Python Shell later. When we make mistakes in this case, we can edit the commands with out having to type everything all over again and again.

Click the New button at the toolbar (shown in red circle), the enlarge the edit area so that it looks like the figure on the right.

The top area (containing the green arrow) is an editor where you can type in lots of commands. The commands typed here are not executed as you type; when you finish writing the program, you can Copy (with Ctrl-C) and Paste (with Ctrl-V) the program to the Python Shell to run that commands.



2.4 Let's draw!

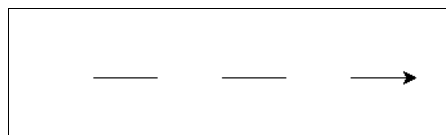
Before we can draw beautiful pictures, we must learn another two turtle commands:

- `penup()` --- tell the turtle to pull the pen up.
- `pendown()` --- tell the turtle to pull the pen down.

Start over by typing the commands:

```
t.home()
t.clear()
```

then tell the turtle to draw the following figure.



Write down your commands here:

2.5 Teach the turtle new commands

What if we can teach the turtle new commands...

We can teach Python new commands. The idea is that we can take a list of commands and define a new command based on that list.

Type the following program in an Editor. Then copy to the Python Shell.

```
def square():
    t.forward(100)
    t.right(90)
    t.forward(100)
    t.right(90)
    t.forward(100)
    t.right(90)
    t.forward(100)
    t.right(90)
```

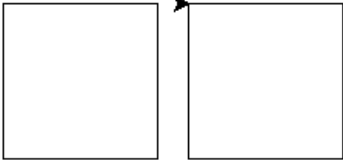
Be careful, especially the symbols `:` at the end of the first line, and the indentation. Note that when we hit Enter on the first line, Wing IDE tries to indent the next line for us automatically.

Copy that program segment to the Python Shell. If you make no mistake, you have already defined a new command `square()`. Try it by typing the following command to the Python Shell.

`square()`

The turtle would draw a rectangle of size 100 x 100.

Write a program using function `square()` to draw the following figure.

	<p><u>Write your commands below:</u></p> <div style="border: 1px solid black; height: 100px;"></div>
---	--

Function `square` that we have just defined does the same work for 4 times. In Python, we can make it more concise as follows.

```
def square():
    for x in range(4):
        t.forward(100)
        t.right(90)
```

Again, be careful on the indentation and the symbols `":"`.

Again, type the program on an editor, copy to the Python Shell, and try calling function `square()`.

Note2: In Python Shell, we can use up arrow to bring back old commands.

From that experiment, we can write function `circle` as follows.

```
def circle():
    for x in range(36):
        t.forward(20)
        t.right(10)
```

Part 3 Number Guessing Game

In this part, we shall develop a program for number guessing game. We will write a program in the editor then we will run the program directly (without having the copy-and-paste to the Python Shell). The goal for this part is to get you familiar with Wing IDE 101 and the Python Programming Language so that you can avoid common mistakes.

The game proceeds as follows:

- A number between 1 to 100 is randomly chosen,
- The user make a guess; the game gives hints by telling if the guess is too high or too low,
- The game ends when the user guess correctly.

We will develop the game iteratively, adding more features as we go.

3.1 Getting Started: Higher or Lower

We start entering the program into the editor by clicking the "New" button on the tool bar. Enter the following code.

```
answer = 53
guess = int(input("Guess the number between 1-100: "))
if guess > answer:
    print("Too high")
if guess < answer:
    print("Too low")
if guess == answer:
    print("Correct answer!")
```

Start the program by clicking the "Run" button on the tool bar (a green "play" triangle). The program will start in the Python Shell. After it ends, the prompt `>>>` would appear to take further commands. If you want to restart, click "Run" again.

Be careful on the symbols : , parentheses and quotes (")

Consider the program and experiment with it to answer the following questions:

<u>Question</u>	<u>Answer</u>
What should you guess to get "Correct answer!"?	
What should you guess to get "Too high"?	
What should you guess to get "Too low"?	
What does if-statement do?	

Note that we do not know how to random numbers; therefore, we use 53 as our random number temporarily.

3.2 Keep guessing

The previous version of our game asks the user once. Now we shall make it asks repeatedly using the `while`-statement.

Edit the program into the following code. Be careful on the indentation, symbols, and the capitalized T in "True".

```
answer = 53
while True:
    guess = int(input("Guess the number between 1-100: "))
    if guess > answer:
        print("Too high")
    if guess < answer:
        print("Too low")
    if guess == answer:
        print("Correct answer!")
```

Observe that the program follows a certain structure created by indentation. The levels of indentation determines the scope of the statement containing that block, in this case, the `while`-statement.

We shall learn more about the `while`-statement. Right now, it is enough to know that if the condition following the keyword `while` is true, the commands inside the block will be executed repeatedly. In this case, a constant `True` is always true.

Run the program and experiment with various guess.

Note that the program keeps running, even though we enter the correct guess.

We can use `break`-statement together with the `while`-statement to "break" out of the loop. Edit the program as shown below and experiment with the new version.

```
answer = 53
while True:
    guess = int(input("Guess the number between 1-100: "))
    if guess > answer:
        print("Too high")
    if guess < answer:
        print("Too low")
    if guess == answer:
        print("Correct answer!")
        break
```

Be careful on the indentation of the `break` statement.

In Python, correct indentation is very important. Try to change the indentation of the `break` statement as follows, and run it. Try to guess with various values.

```
answer = 53
while True:
    guess = int(input("Guess the number between 1-100: "))
    if guess > answer:
        print("Too high")
    if guess < answer:
        print("Too low")
    if guess == answer:
        print("Correct answer!")
    break
```

How is the new program perform? Can you explain why?

Don't forget to change the program back to the correct one before continue.

3.3 Pick a number at random

We are left with the last piece of our puzzle: to random a number. We shall use function `randint` from module `random`. To do so, we have to announce at the beginning of our program:

```
import random
```

after that line, we can refer to every function from module `random` in using syntax of the form `random.functionname`.

Function `randint` takes two arguments, which are the minimum and the maximum random values that we need. We can edit our program to use that function as follows.

```
import random

answer = random.randint(1,100)
while True:
    guess = int(input("Guess the number between 1-100: "))
    if guess > answer:
        print("Too high")
    if guess < answer:
        print("Too low")
    if guess == answer:
        print("Correct answer!")
        break
```

3.4 Count the number of guesses

We can use variable `count` to keep tracks of the number of times the user has guessed. This variable is initially zero and will increase as the user plays the game.

```
import random

answer = random.randint(1,100)
count = 0
while True:
    count = count + 1
    guess = int(input("Guess the number between 1-100: "))
    if guess > answer:
        print("Too high")
    if guess < answer:
        print("Too low")
    if guess == answer:
        print("Correct answer!")
        break
print("You guessed",count,"times.")
```

The line `count = count + 1` can be written more concisely as:

```
count += 1
```

(Extra credit, not required) If we want the program to print "You did very well." when the user guesses correctly with in 5 guesses, what program should we add at the end of the last program.