
Introduction to Software Design

P01. Hello World!

Tae Hyun Kim

Spring 2020

Introduction

- Introduction to Python
- Running the Python Interpreter
- Evaluating expressions
- Storing values in variables
- Strings
- Write & run the first program
 - "Hello world"
 - "My Favorite Stuff"

Introduction to Python

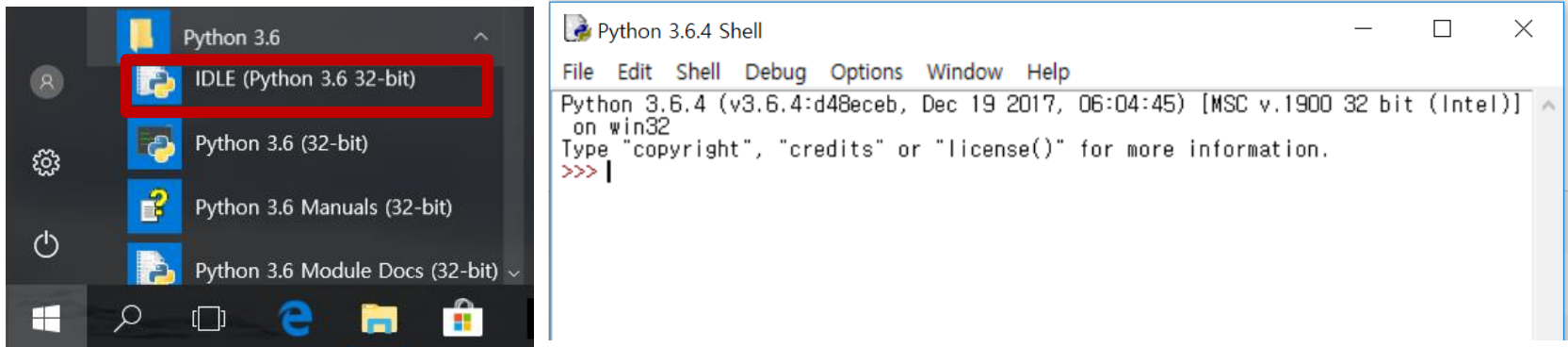
- Python
 - Easier to learn than C.
 - Serious programming language.
 - Popular language in research & scientific community of other areas.
- Python programming
 - Need software called the **Python interpreter**.
 - **Interpreter**
 - : a program that understands the instructions that you'll write in the Python language.

Python 2 & Python 3

- Python 2 is still in active use.
- Python 3 is the future of Python.
 - A lot of very useful features & fixes for well-known problems
 - To do this, **Python 3 breaks backward compatibility.**
- We use **Python 3** in this class.

IDLE

- : Python's **I**ntegrated **D**evelopment and **L**earning **E**nvironment.
 - Provides both interactive & non-interactive mode.
- Windows: Installed with Python interpreter.



- Ubuntu: You need to install IDLE by

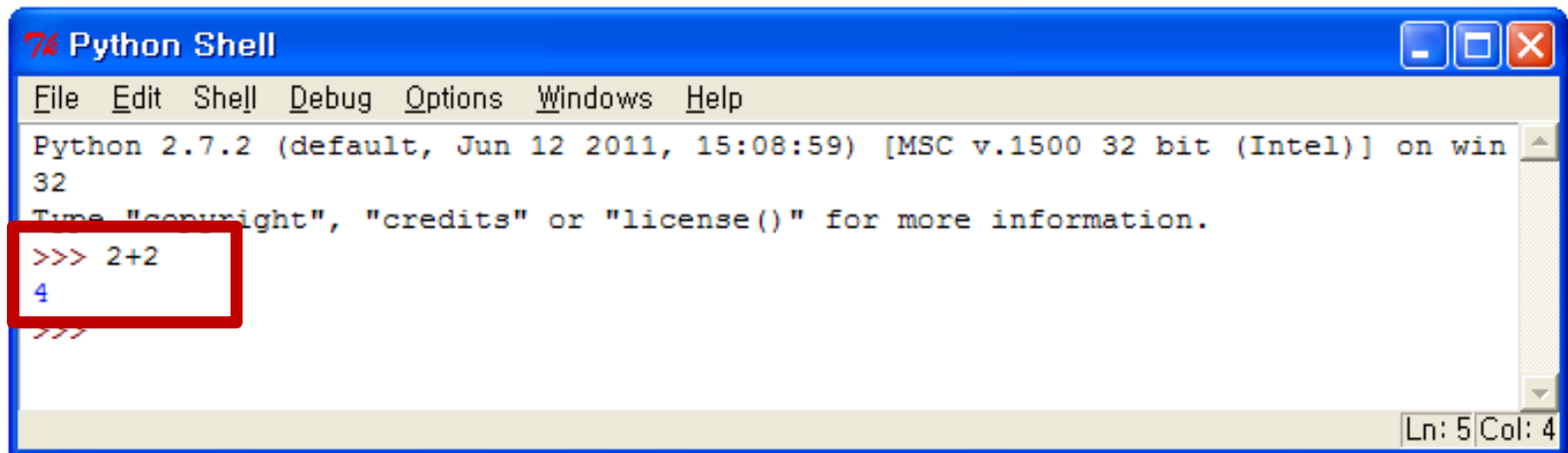
```
$ sudo apt-get install idle-python3.x (자신의 버전에 맞게)
```

Actually, IDLE is not mandatory

- You can use instead of IDLE...
 - Running Python interpreter in cmd or terminal window directly (for interactive mode).
 - Any editors (for non-interactive mode) - Vim, Notepad++, Sublime Text, Atom, Notepad, gedit...
- Following slides are based on IDLE, just as an example.

IDLE - Interactive Mode

- Some Simple Math Stuff
 - Type **2+2** into the shell and press the **Enter** key.
 - Computer should respond with the number **4**.
: the sum of **2+2**



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>>
```

Math Operators

- Some Simple Math Stuff
 - The various math operators in Python

2+2	Addition
2-2	Subtraction
2*2	Multiplication
2/2	Division

- +, -, *, and / are called **operators**.
- * sign is called an **asterisk** (and also called a “star”)

Numbers

- Integers and Floating Point Numbers
 - **Integers**
 - Whole numbers (like 4, 0, and 99)
 - **Floating point numbers**
 - Numbers with a decimal point (like 5.0)
 - **In Python**
 - The number 5 is an **integer**.
 - But if we wrote it as 5.0 it would **not be an integer**.

```
>>> 2.0/3.0  
0.66666
```

Warning: Integer division

In Python 2:

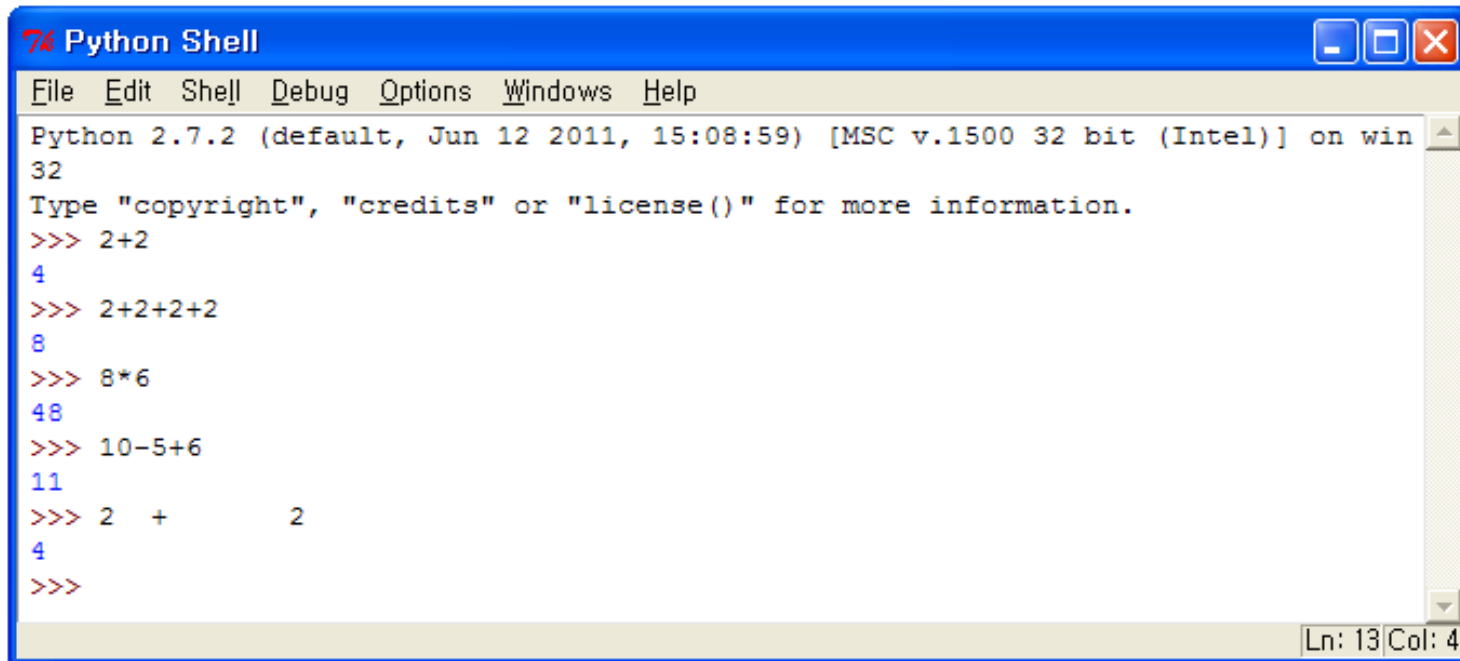
```
>>> 3 / 2  
1
```

In Python 3:

```
>>> 3 / 2  
1.5
```

Evaluating Expressions

- Expressions



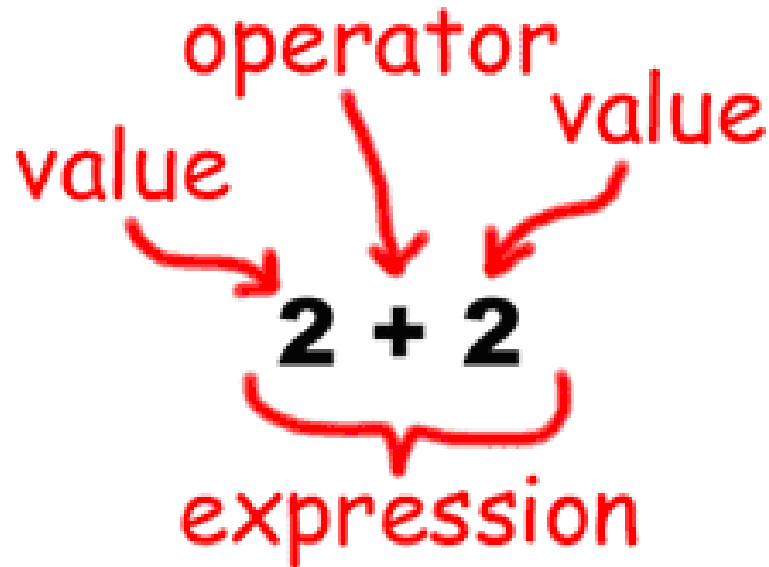
The screenshot shows a 'Python Shell' window with a menu bar (File, Edit, Shell, Debug, Options, Windows, Help) and a status bar (Ln: 13 Col: 4). The main text area contains the following text:

```
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 2+2+2+2
8
>>> 8*6
48
>>> 10-5+6
11
>>> 2 + 2
4
>>>
```

- These math problems are called **expressions**.
- These integers are also called **values**.

Evaluating Expressions

- Expressions
 - An expression is made up of **values** and **operators**.

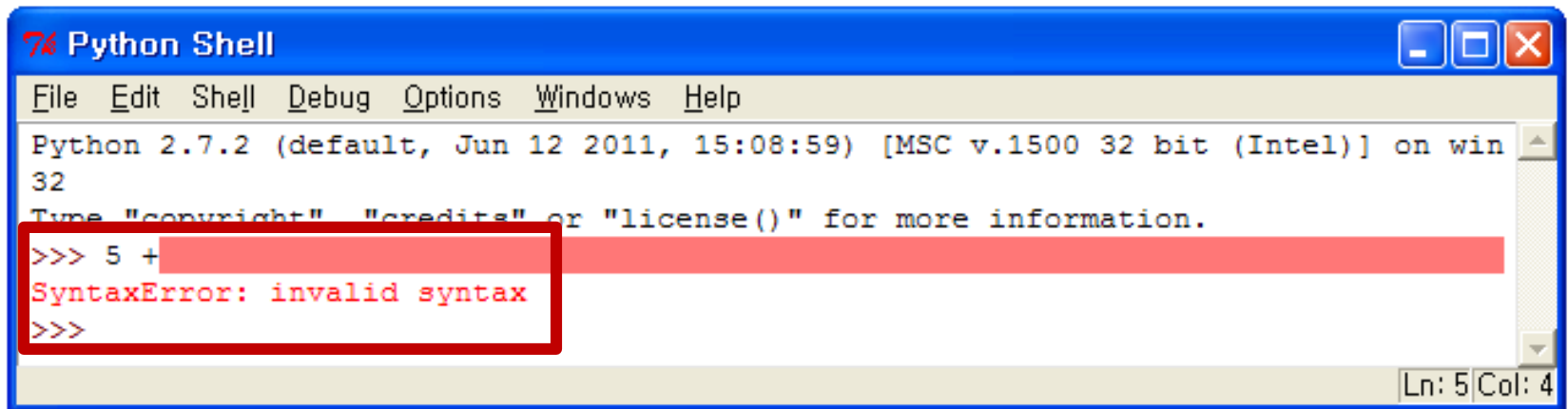


Evaluating Expressions

- Computer solves the **expression** $10 + 5$ and outputs the **value** 15.
- The expressions $10 + 5$ and $10 + 3 + 2$
 - have the **same value**.
 - they both **evaluate to 15**.
- Single values are considered expressions.
 - The expression 15 evaluates to the value 15.

Evaluating Expressions

- If you type in just “5 +”,
 - you will get an **error message**.



The screenshot shows a Python Shell window titled "Python Shell" with a menu bar (File, Edit, Shell, Debug, Options, Windows, Help). The status bar at the bottom indicates "Ln: 5 Col: 4". The main text area shows the following text:

```
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> 5 +
SyntaxError: invalid syntax
>>>
```

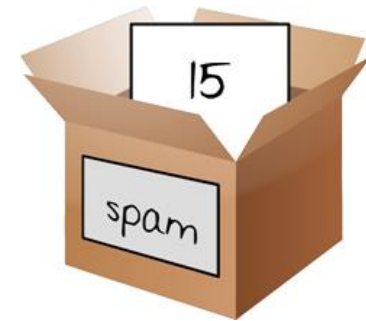
A red rectangular box highlights the input "5 +" and the resulting "SyntaxError: invalid syntax" message.

- It is because **5 +** is **not an expression**.
- Expressions have values connected by operators.
- The binary **+** **operator** always expects to **connect two things** in Python.

Storing Values in Variables

- Variables

- A box that can hold values.
- We can **store values** in the variables.
 - with the **= sign** (called the **assignment operator**)



- For example, to store the value **15** in a variable named “spam”,

```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> spam = 15
>>> spam
15
Ln: 6|Col: 4
```

Storing Values in Variables

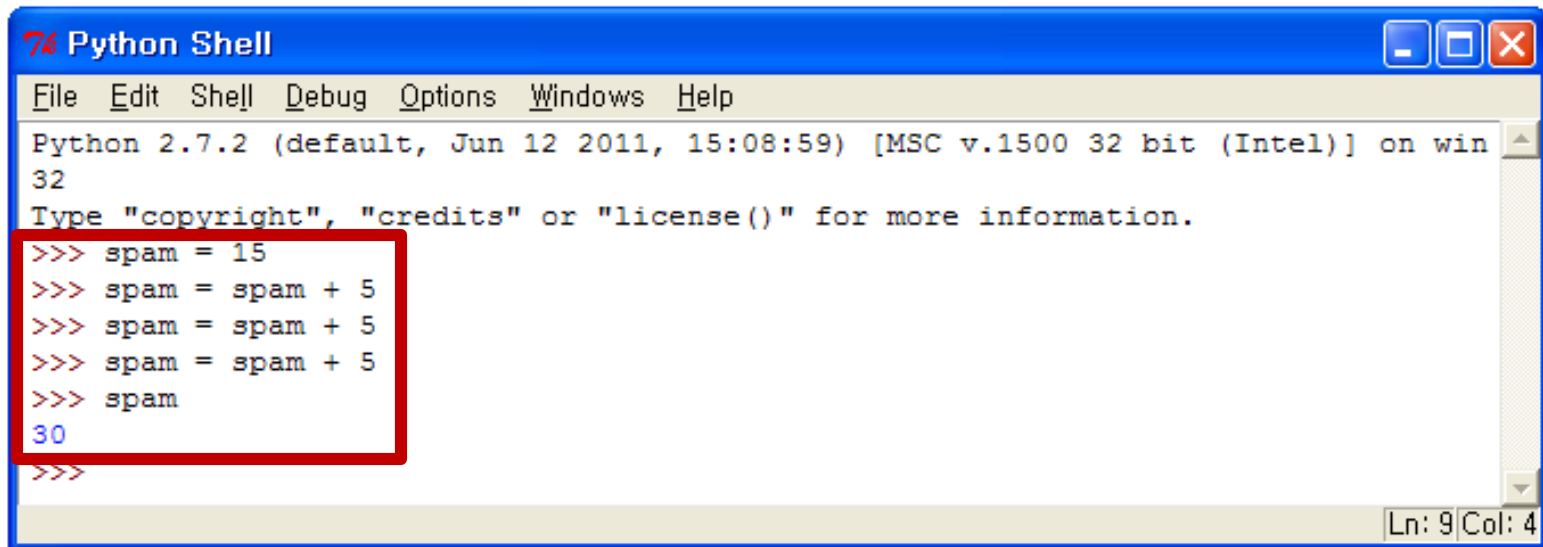


```
>>> spam = 15
>>> spam + 5
20
```

```
>>> spam = 15
>>> spam + 5
20
>>> spam = 3
>>> spam + 5
8
```

Storing Values in Variables

- Write Expressions with Variables



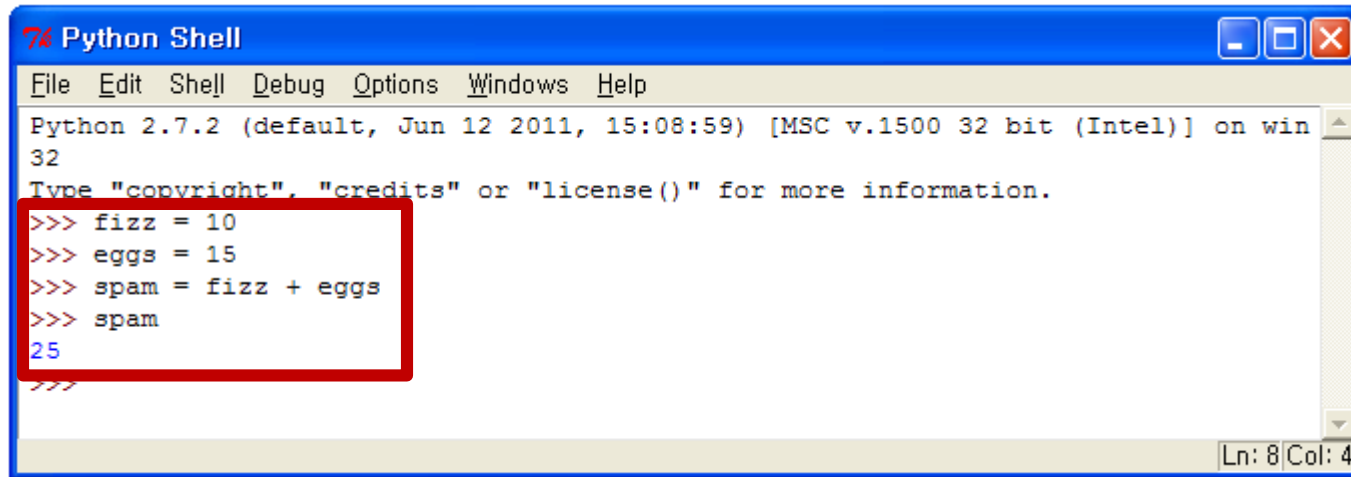
The screenshot shows a Python Shell window with a blue title bar and standard Windows window controls. The menu bar includes File, Edit, Shell, Debug, Options, Windows, and Help. The main text area displays the Python version and environment information, followed by a prompt to type 'copyright', 'credits', or 'license()'. A red rectangular box highlights a series of five lines of code: three lines incrementing the variable 'spam' by 5, and one line printing the final value of 'spam'. The output '30' is shown below the last line of code. The status bar at the bottom right indicates 'Ln: 9 | Col: 4'.

```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> spam = 15
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam
30
>>>
```

- The value **15** was **overwritten**.

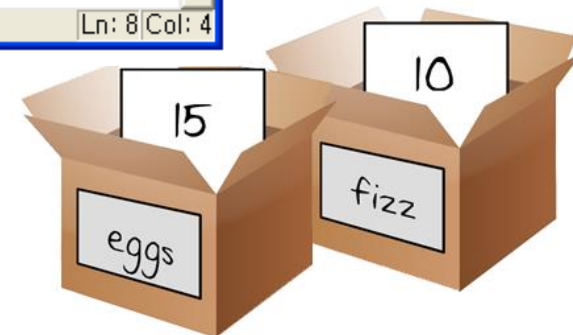
Storing Values in Variables

- Using More Than One Variable
 - Often we'll need to use **multiple variables**.
 - The "**fizz**" and "**eggs**" variables have values stored in them.



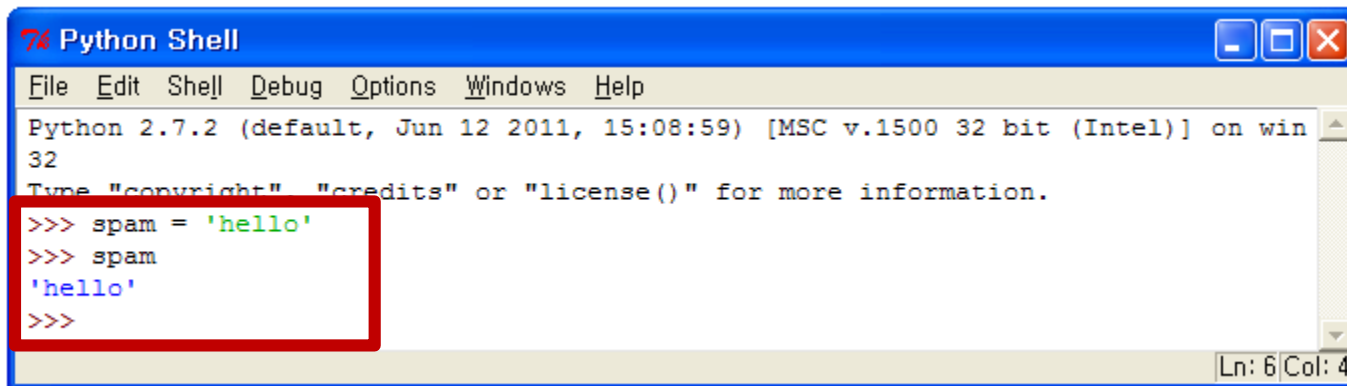
```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> fizz = 10
>>> eggs = 15
>>> spam = fizz + eggs
>>> spam
25
>>>
```

The screenshot shows a Python Shell window with a blue title bar and a menu bar. The main text area displays the Python version and environment information. A red rectangle highlights the four lines of code where variables are assigned and a calculation is performed. The output of the calculation is shown on the next line.



Strings

- Strings
 - Little chunks of text.
 - Can store string values inside variables.
 - Put them in between **two single quotes (')**.



A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following text: "Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win32" and "Type 'copyright', 'credits' or 'license()' for more information." Below this, a red rectangle highlights the following code: ">>> spam = 'hello'", ">>> spam", and "'hello'". The status bar at the bottom right shows "Ln: 6 | Col: 4".

```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> spam = 'hello'
>>> spam
'hello'
>>>
```

Ln: 6 | Col: 4

Strings

- Strings
 - Strings can have **spaces** and **numbers** as well.
 - Examples of strings

```
'hello'
```

```
'Hi there!'
```

```
'KITTENS'
```

```
'7 apples, 14 oranges, 3 lemons'
```

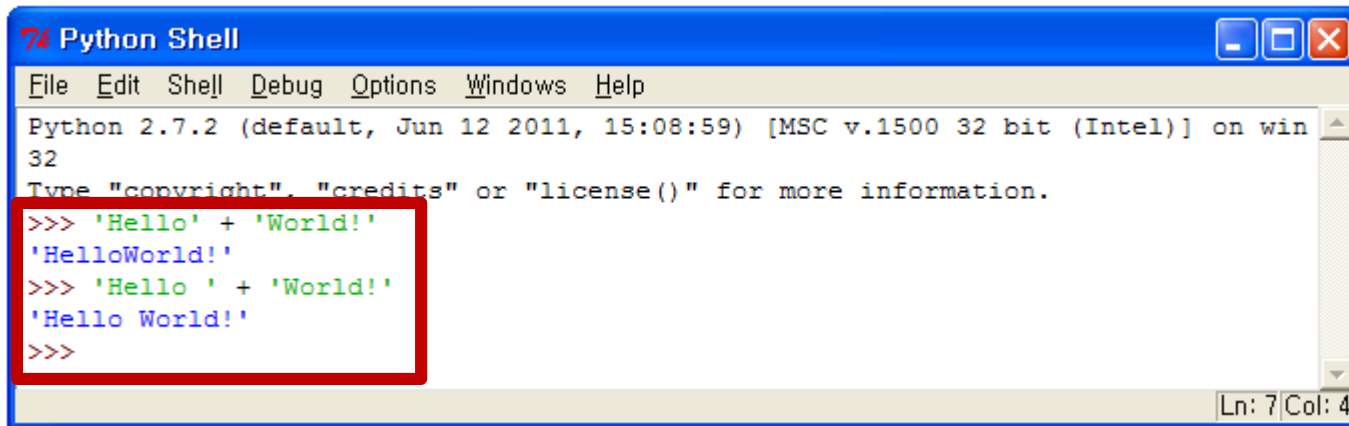
```
'Anything not pertaining to elephants is irrelephant.'
```

```
'A long time ago in a galaxy far, far away...'
```

```
'O*&#wY%*&OCfsdYO*&gfC%YO*&%3yc8r2'
```

Strings

- Strings Concatenation
 - Can **add one string** to the end of another by **using the + operator**.
 - Ex) Focus on **a space** at the end of the 'Hello' string.



A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following text:

```
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.

>>> 'Hello' + 'World!'
'HelloWorld!'
>>> 'Hello ' + 'World!'
'Hello World!'
>>>
```

The last three lines of code and their output are highlighted with a red rectangular box. The status bar at the bottom right shows "Ln: 7 | Col: 4".

Statement & Expression

- Expression

- A statement that evaluates to a **value**.

- e.g.

```
3 + 7  
min(2, 22)  
'foo'  
'foo'+'bar'
```

- Statement

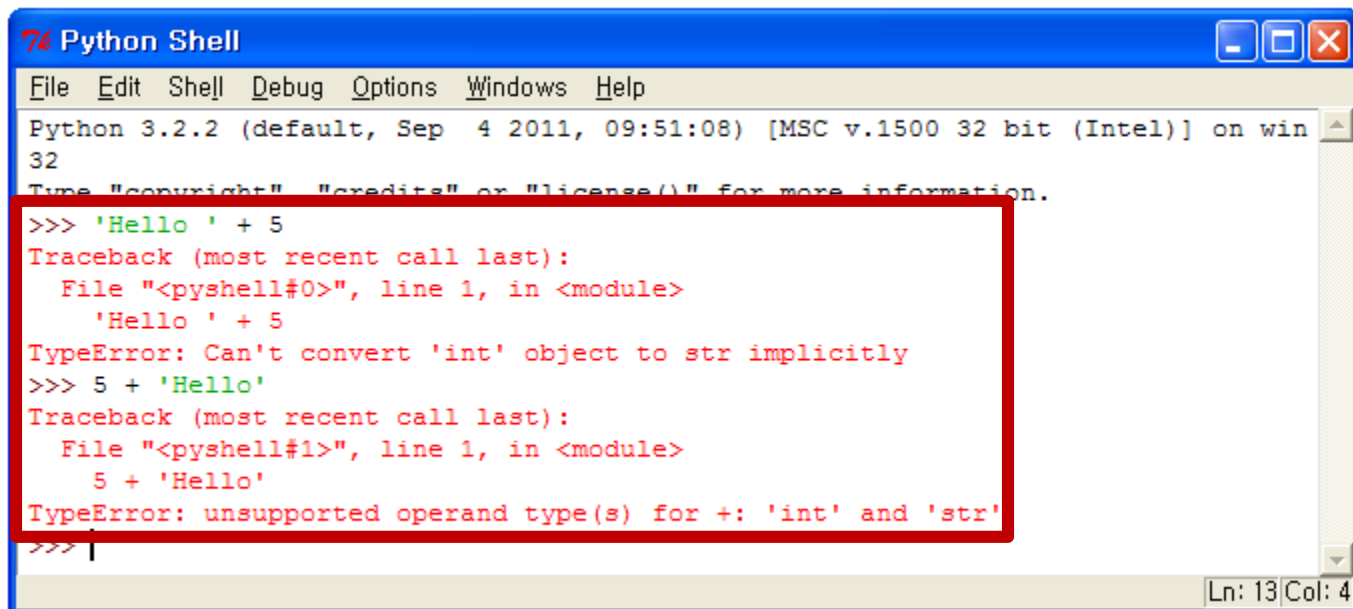
- A unit that expresses some action to be carried out.

- e.g.

```
3 + 7  
if x:  
    return  
a = 7
```

Strings

- Data Types
 - Can't add a **string** to an **integer**, or an **integer** number to a **string**.
 - Because a **string** and an **integer** are different data types.



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.2 (default, Sep  4 2011, 09:51:08) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> 'Hello ' + 5
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    'Hello ' + 5
TypeError: Can't convert 'int' object to str implicitly
>>> 5 + 'Hello'
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    5 + 'Hello'
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> |
```

Ln: 13 Col: 4

Python 3 Data Types

- **Numeric**

- `int / float / complex`
- Types that describe numeric content

- **Boolean**

- `bool`
- Types that define true/false relationships
- *Logical operators: and / or / not*

- **Text**

- `str`
- Immutable string objects
- 1,2, or 3 quotes

- **Sequence**

- **Lists**

- Typically homogeneous sequences of objects
 - An mutable, ordered array
 - Square Brackets

- **Tuple**

- Typically heterogeneous sequences of objects
 - An immutable collection
 - Parenthesis

- **Mapping**

- **Dictionaries**

- A mutable, unordered, associative array
 - Curly Brackets

int, float, str

- `int`
 - Integer numbers
 - e.g. 1, 5, 20, ...
- `float`
 - Floating point numbers
 - e.g. 1.0, 3.14., 1.111111111..., ...
- `str`
 - Strings
 - e.g. 'hello world', 'asdf', ' ', '24', ...

Type Casting

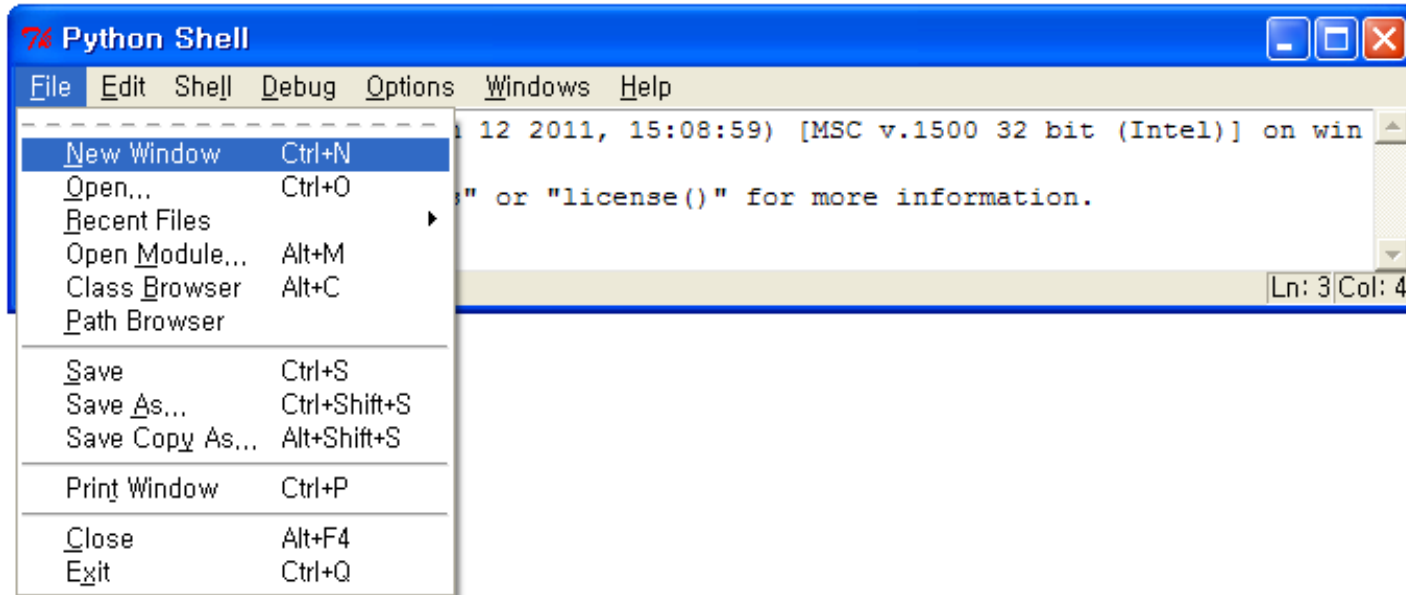
- `int(x)`
 - Converts a number or string `x` to an integer number
 - Q: Any string?
- `float(x)`
 - Converts a number or string `x` to a floating point number
- `str(x)`
 - Converts any object `x` to a string

Type Castings

```
>>> 2 + 1.1
3.1
>>> 2 + '1.1'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> 2 + float('1.1')
3.1
>>> 2 + int('1.1')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '1.1'
>>> 2 + int('1')
3
>>> str(2) + '1.1'
'21.1'
>>> str(2) + 'hello'
'2hello'
>>> fstr = '1.1'
>>> 2 + float(fstr)
3.1
>>>
```

IDLE - Non-Interactive Mode

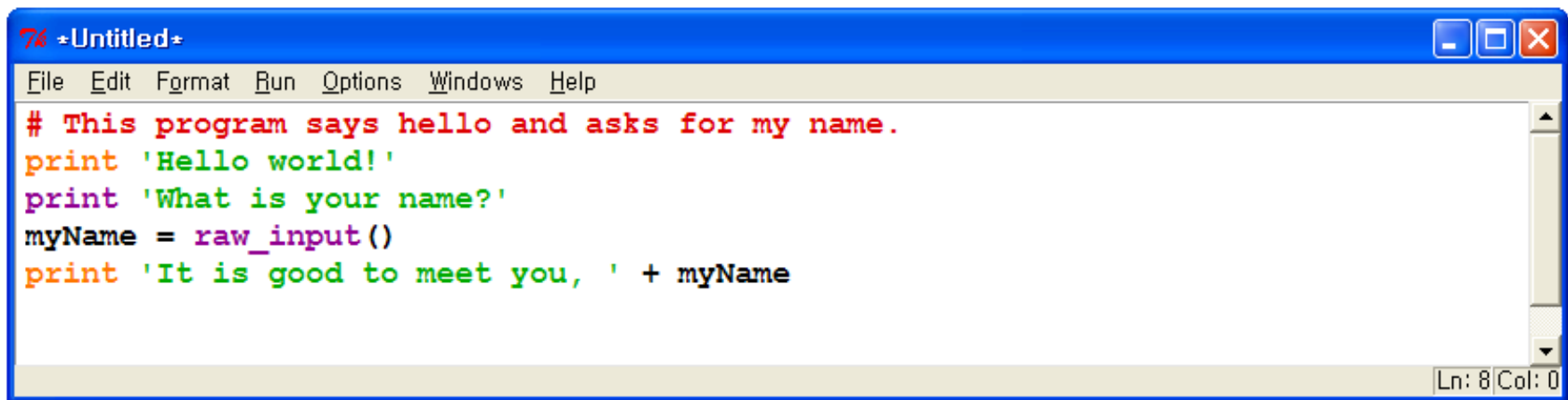
- Program “Hello World!”



- You will see “New File” instead of “New Window” from Python 3.6.4.
- Of course, you can use any other editors instead of IDLE.

Write the First Program “Hello World!”

- Program “Hello World!”
 - We call this text the **source code** of the program.



```
# This program says hello and asks for my name.  
print 'Hello world!'  
print 'What is your name?'  
myName = raw_input()  
print 'It is good to meet you, ' + myName
```

The screenshot shows a window titled '*Untitled*' with a menu bar (File, Edit, Format, Run, Options, Windows, Help). The code is written in a monospaced font with syntax highlighting: comments are red, strings are green, and keywords/variables are purple. The status bar at the bottom right indicates 'Ln: 8 Col: 0'.

- *raw_input()* is for Python 2, whereas *input()* is used for Python 3.
- With Python 2, we have *print x* (no parentheses), which becomes *print(x)* in Python 3.

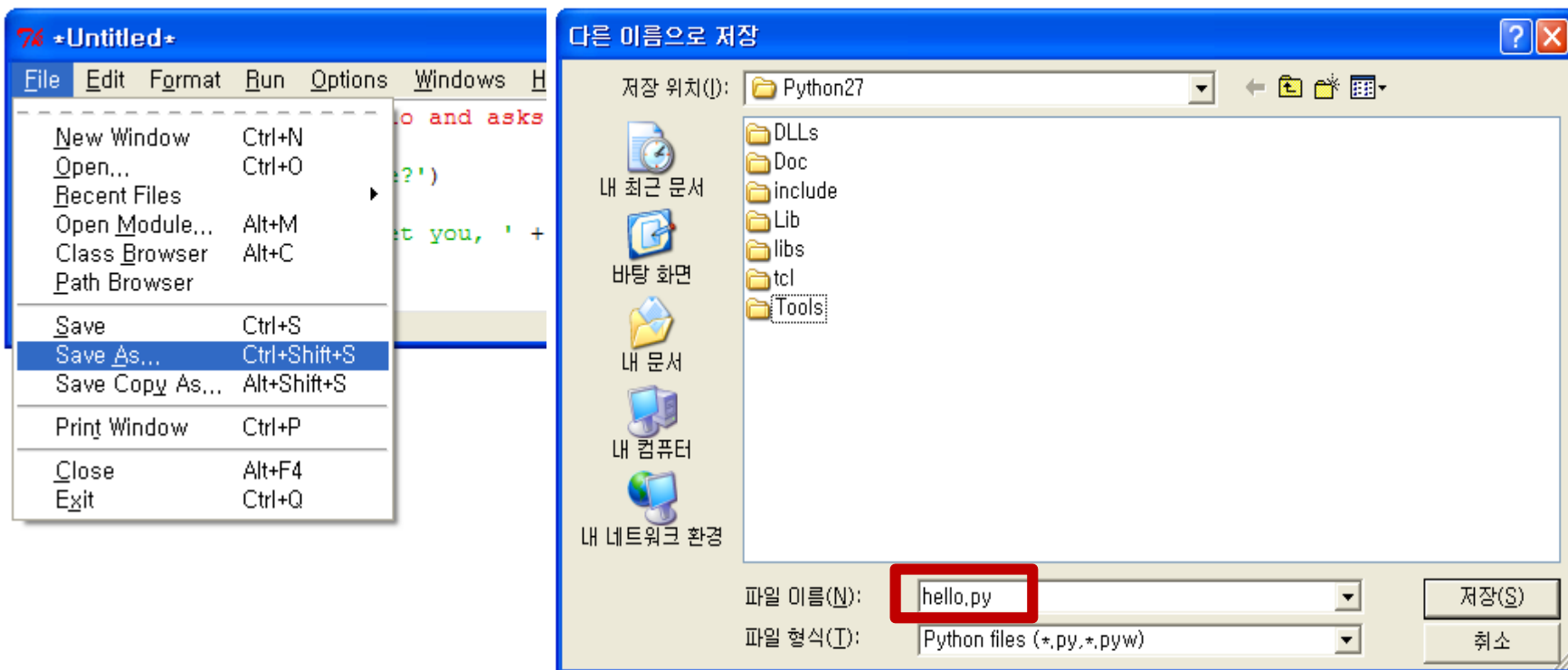
Write the First Program “Hello World!”

- Program “Hello World!”
 - Type the following text into this new window.

```
# This program says hello and asks for my name.  
print('Hello world!')  
print('What is your name?')  
myName = input()  
print('It is good to meet you, ' + myName)
```

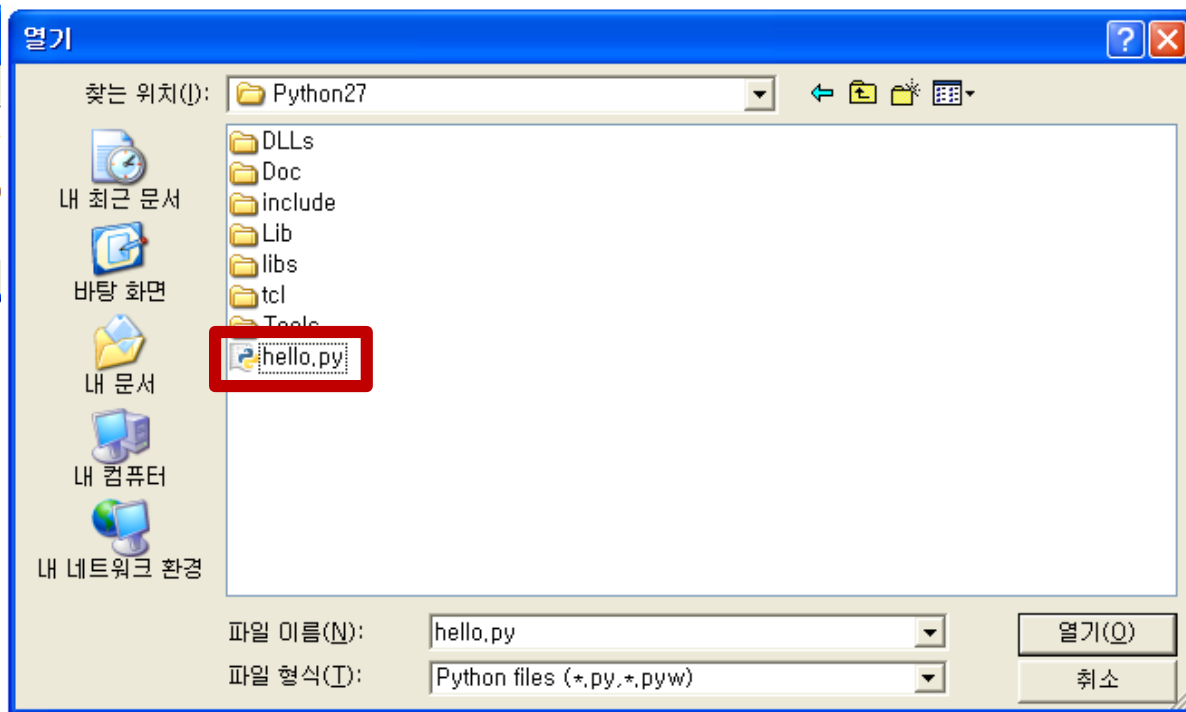
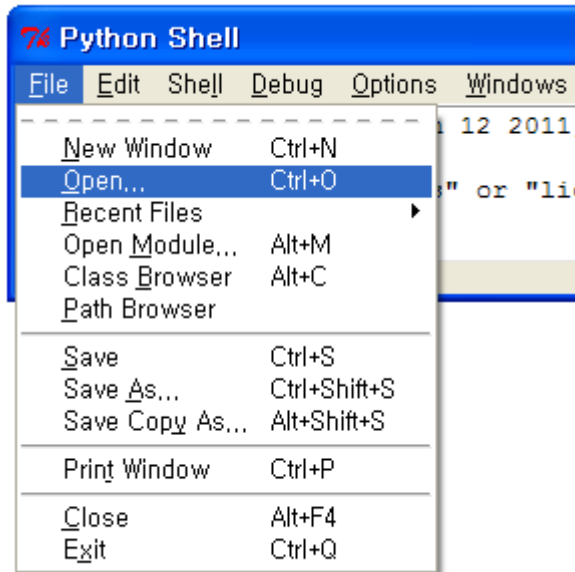
Write the First Program “Hello World!”

- Program “Hello World!”
 - Save the program.



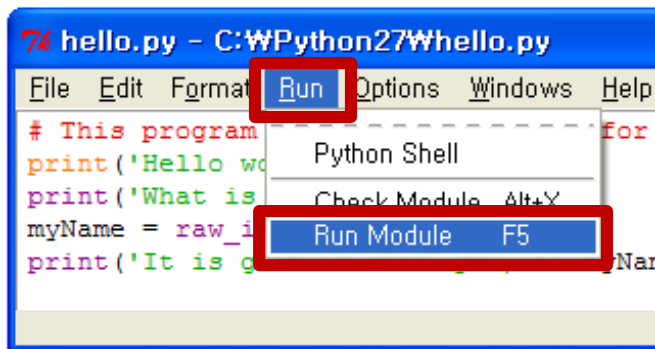
Write the First Program “Hello World!”

- Program “Hello World!”
 - Open the program you've saved

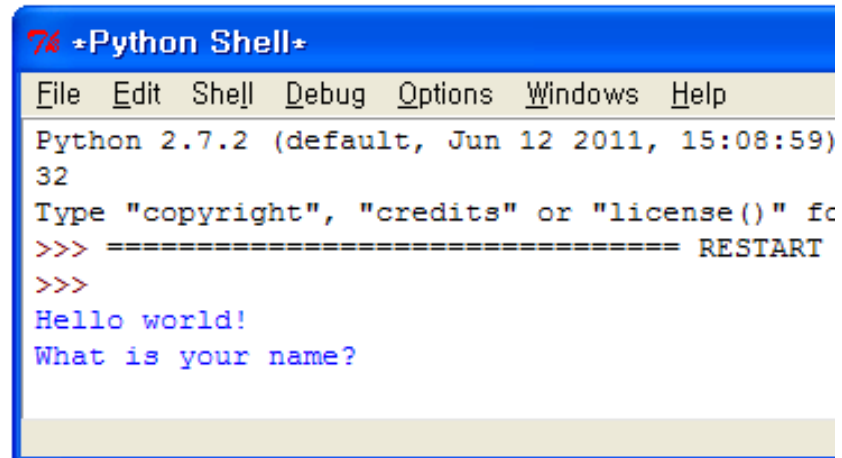


Run the First Program “Hello World!”

- Programs “Hello World!”
 - Run “Hello World!” program.
 - choose **Run > Run Module** or just press the **F5** key.



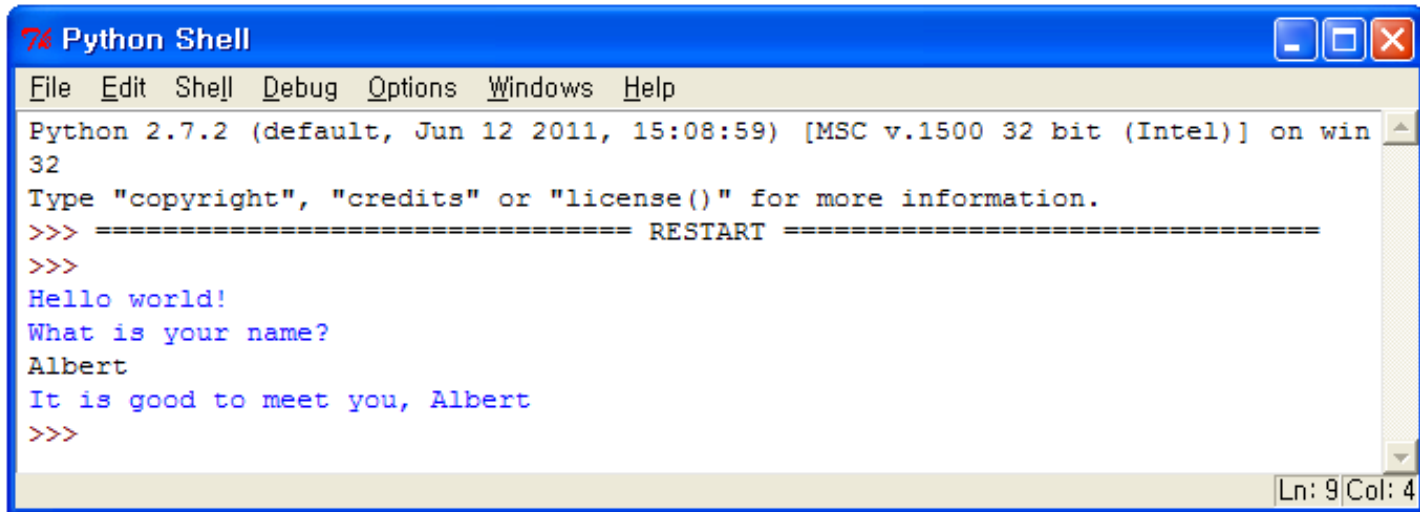
```
7% hello.py - C:\WPYthon27\hello.py
File Edit Format Run Options Windows Help
# This program
print('Hello world!')
print('What is your name?')
myName = raw_input()
print('It is good to meet you, ' + myName + '!')
```



```
7% +Python Shell+
File Edit Shell Debug Options Windows Help
Python 2.7.2 (default, Jun 12 2011, 15:08:59)
32
Type "copyright", "credits" or "license()" for more
>>> ===== RESTART
>>>
Hello world!
What is your name?
```


The First Program “Hello World!”

- Program “Hello World!”



The screenshot shows a Python Shell window with a blue title bar and standard Windows window controls. The menu bar includes File, Edit, Shell, Debug, Options, Windows, and Help. The main text area displays the following content: Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win32, followed by a prompt to type 'copyright', 'credits', or 'license()'. After a restart, the shell shows 'Hello world!', a prompt 'What is your name?', the input 'Albert', and the output 'It is good to meet you, Albert'. The status bar at the bottom right indicates 'Ln: 9 Col: 4'.

```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Hello world!
What is your name?
Albert
It is good to meet you, Albert
>>>
Ln: 9 Col: 4
```

Users	run and use the program
Programmers	write the program
Flow of execution or execution	program starts at the very top and then executes each line
	Executes each statement step-by-step

The First Program “Hello World!”

- Code Explanation

- **Comment**

- Any text following a **# sign** (called the **pound sign**) is a comment.
 - Not for the computer, but for the programmer.

```
1. # This program says hello and asks for my name.
```

- **Print function**

- The **print** keyword followed by an expression enclosed in parentheses.
 - Will **display** the text on the screen.

```
2. print('Hello world! `')  
3. print('What is your name? `')
```

The First Program “Hello World!”

- Code Explanation

- **Function**

- a bit of code that does a particular action.
 - a function can take any expression as an argument.

- **Function call**

- a piece of code that tells our program to run the code inside a function.

- **Return value**

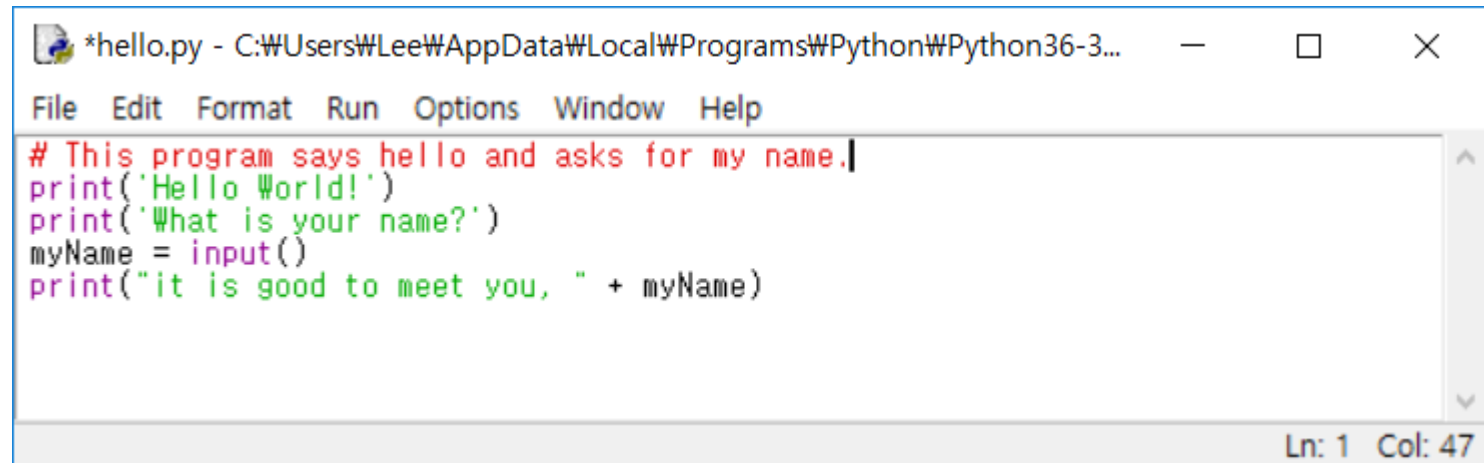
- The **value that the function call** evaluates to is called the return value.
 - Function output!

- **Ending the Program**

- Once the program executes the **last line, it stops.**
At this point, your monitor displays **terminated** or **exited**.

The First Program “Hello World!”

- Code Explanation

A screenshot of a Python IDE window titled '*hello.py - C:\Users\WLee\AppData\Local\Programs\Python\Python36-3...'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The code editor contains the following Python code:

```
# This program says hello and asks for my name.  
print('Hello World!')  
print('What is your name?')  
myName = input()  
print("it is good to meet you, " + myName)
```

The status bar at the bottom right shows 'Ln: 1 Col: 47'.

- **Variable**

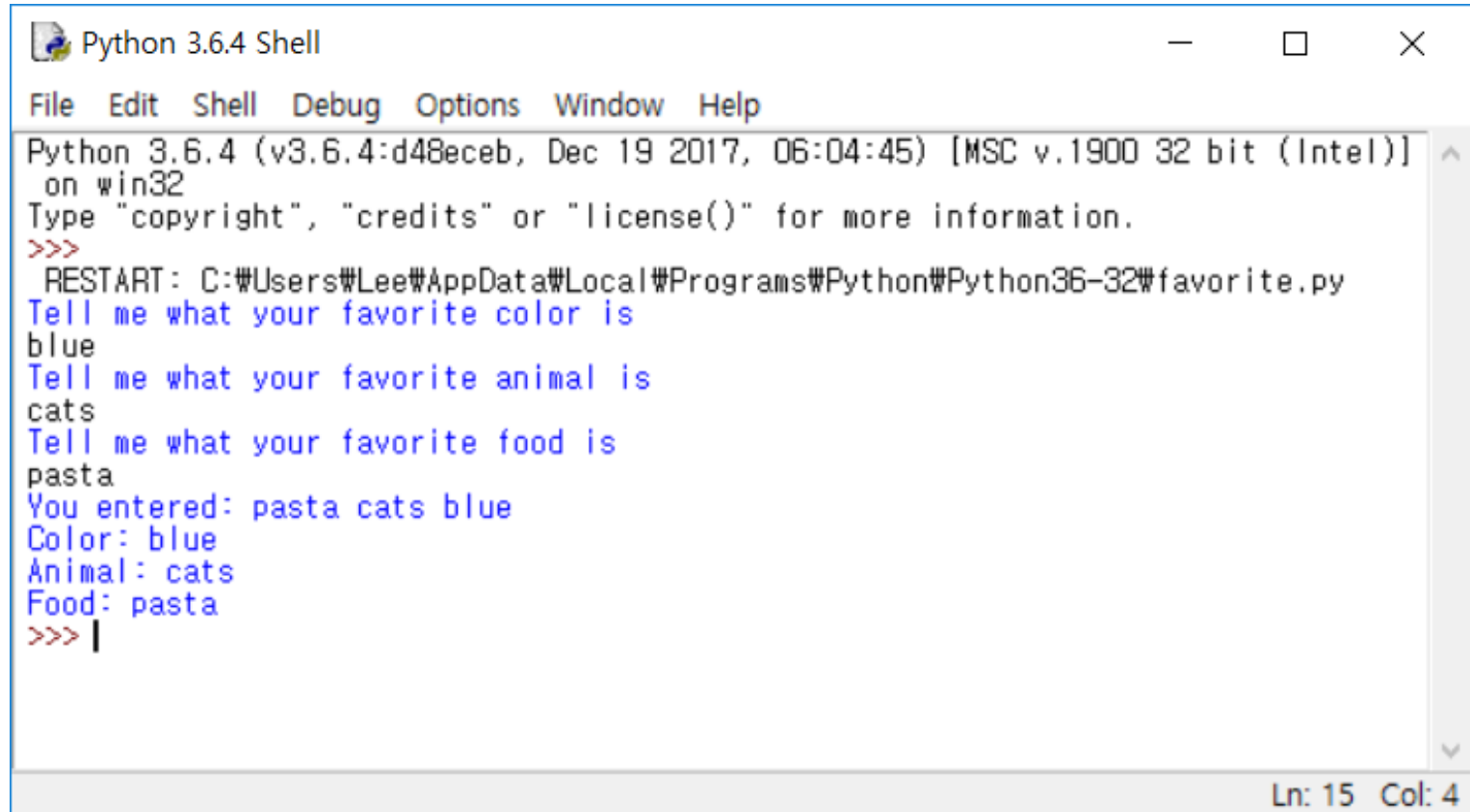
- myName

- **Function**

- print(), input()

“My Favorite Stuff”

- Program “My Favorite Stuff”



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:\Users\Lee\AppData\Local\Programs\Python\Python36-32\favorite.py
Tell me what your favorite color is
blue
Tell me what your favorite animal is
cats
Tell me what your favorite food is
pasta
You entered: pasta cats blue
Color: blue
Animal: cats
Food: pasta
>>> |
```

Ln: 15 Col: 4

“My Favorite Stuff”

- Program “My Favorite Stuff”

```
# Favorite stuff
print('Tell me what your favorite color is')
favoriteColor = input()

print('Tell me what your favorite animal is')
favoriteAnimal = input()

print('Tell me what your favorite food is')
favoriteFood = input()

# display our favorite stuff
print('You entered: ' + favoriteFood + ' ' + favoriteAnimal + ' ' + favoriteColor)

# print 'Here is a list of your favorite things.'
print('Color: ' + favoriteColor)
print('Animal: ' + favoriteAnimal)
print('Food: ' + favoriteFood)
```

“My Favorite Stuff”

- Code Explanation

- **Comment**

- The program **ignores it**.
 - All the text after the **pound sign(#)** will be ignored by the program.

```
1. # Favorite stuff
```

- Display a bit of text asking the user to type in their **favorite color**.

```
2. print('Tell me what your favorite color is.')
```

“My Favorite Stuff”

- Code Explanation
 - **input ()** function
 - Let the user type in their favorite color.
 - string the user entered is stored in the variable **favoriteColor**.
 - reads any input as a **string**

```
3. favoriteColor = input()
```


“My Favorite Stuff”

- Code Explanation
 - `input()` function
 - It used to be `raw_input()` for Python 2.
 - There is a **blank line** before the print statement, which is ignored in Python.

```
5. print('Tell me what your favorite animal is.')
```

```
6. favoriteAnimal = input()
```

```
8. print('Tell me what your favorite food is.')
```

```
9. favoriteFood = input()
```

“My Favorite Stuff”

- Code Explanation

- **Another comment**

- It does not have to be at the top all the time (can be **anywhere**).

```
11. # display our favorite stuff
```

- **print() function**

- Show us the favorite food, animal, and color we entered.
 - The **plus sign** is used to combine the strings.

```
12. print('You entered: ' + favoriteFood + ' ' +  
        + favoriteAnimal + ' ' + favoriteColor)
```

“My Favorite Stuff”

- Code Explanation
 - Another comment line

```
13. # print the list of favorite things
```

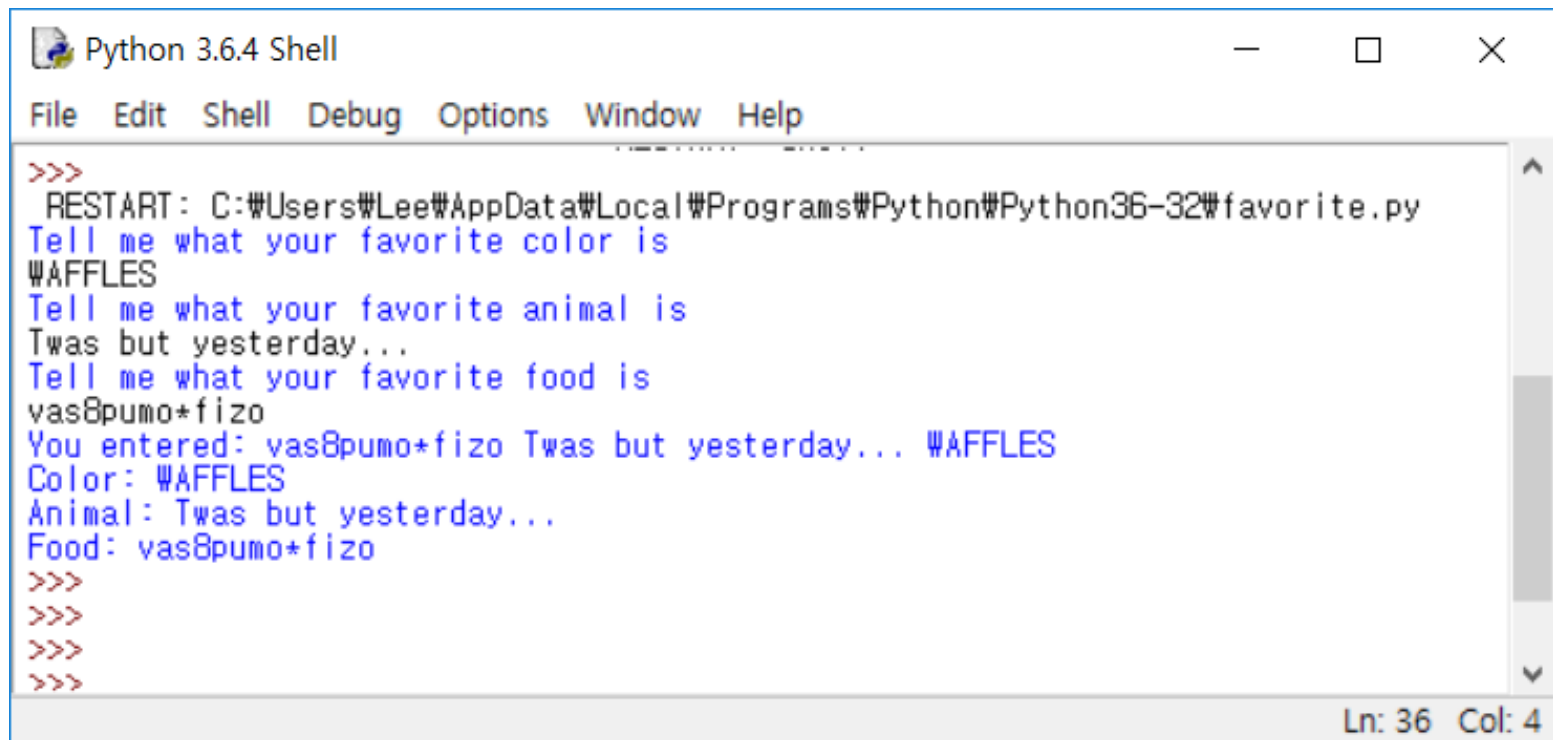
– **print** function

- These three lines say your favorite things

```
14. print('Color: ' + favoriteColor)
15. print('Animal: ' + favoriteAnimal)
16. print('Food: ' + favoriteFood)
```

“My Favorite Stuff”

- Crazy Answers and Crazy Names for our Favorite Stuff
 - The computer doesn’t really care what you type in.



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
>>>
RESTART: C:\Users\Lee\AppData\Local\Programs\Python\Python36-32\favorite.py
Tell me what your favorite color is
WAFFLES
Tell me what your favorite animal is
Twas but yesterday...
Tell me what your favorite food is
vas8pumo+fizo
You entered: vas8pumo+fizo Twas but yesterday... WAFFLES
Color: WAFFLES
Animal: Twas but yesterday...
Food: vas8pumo+fizo
>>>
>>>
>>>
>>>
Ln: 36 Col: 4
```

“My Favorite Stuff”

- Crazy Answers and Crazy Names for our Favorite Stuff
 - The program also **does not care** what name we give our variables.

```
1. # Favorite stuff 2
2. print('Tell me what your favorite color is.')
3. q = input()
4.
5. print('Tell me what your favorite animal is.')
6. fizzy = input()
7.
8. print('Tell me what your favorite food is.')
9. AbrahamLincoln = input()
10.
11. # display our favorite stuff
12. print('You entered: ' + q + ' ' + fizzy + ' ' + AbrahamLincoln)
13. # print 'Here is a list of your favorite things.'
14. print('Color: ' + q)
15. print('Animal: ' + fizzy)
16. print('Food: ' + AbrahamLincoln)
```

“My Favorite Stuff”

- Capitalizing our Variables
 - This is to make the variable names **easier to read**.
 - Because variable names can't have spaces within them.

```
thisnameiskindofhardtoread  
thisNameIsEasierToRead
```

- Leave the first word in **lowercase**.
 - But start all the following words with a **uppercase letter**.
 - This is what is called a **camelCase naming convention**.

“My Favorite Stuff”

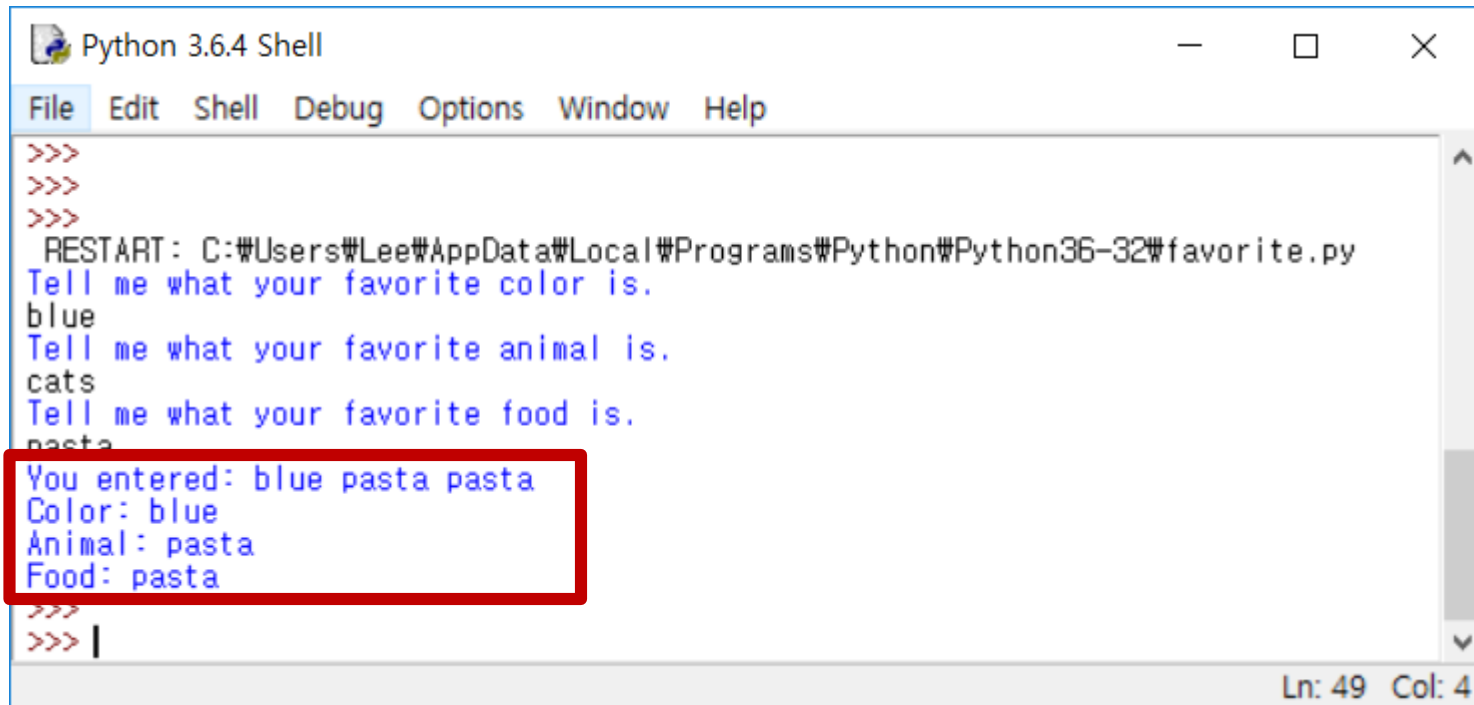


- What happened here?

```
1. # Favorite stuff 3
2. print('Tell me what your favorite color is.')
3. q = input()
4.
5. print('Tell me what your favorite animal is.')
6. AbrahamLincoln = input()
7.
8. print('Tell me what your favorite food is.')
9. AbrahamLincoln = input()
10.
11. # display our favorite stuff
12. print('You entered: ' + q + ' ' + AbrahamLincoln + ' ' + AbrahamLincoln)
13. # print 'Here is a list of your favorite things.'
14. print('Color: ' + q)
15. print('Animal: ' + AbrahamLincoln)
16. print('Food: ' + AbrahamLincoln)
```

“My Favorite Stuff”

- The favorite animal value was **overwritten**.
 - A variable can only store **one value at a time**.



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
>>>
>>>
>>>
RESTART: C:\Users\Lee\AppData\Local\Programs\Python\Python36-32\favorite.py
Tell me what your favorite color is.
blue
Tell me what your favorite animal is.
cats
Tell me what your favorite food is.
pasta
You entered: blue pasta pasta
Color: blue
Animal: pasta
Food: pasta
>>>
>>> |
```

Ln: 49 Col: 4

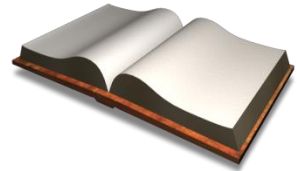
“My Favorite Stuff”

- Case-sensitivity
 - The computer considers these names to be **four different variables**.

```
fizzy
Fizzy
FIZZY
fIzZy
```

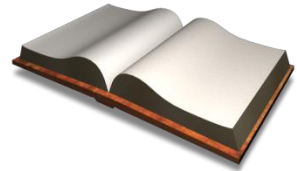
- The computer doesn't know of a function named **INPUT ()**.
- It only knows a function named **input ()**.

Things Covered In This Chapter(1/2)



- Introduction to Python
- Using IDLE's interactive shell to run instructions
- Flow of execution
- Expressions and their evaluations
- Statements
- Integer
- Operators (such as + - *)
- Variables
- Assignment statements
- Overwriting values in variables

Things Covered In This Chapter(2/2)



- Strings and string concatenation
- Data types (such as strings or integers)
- Type casting
- Using IDLE to write source code
- Saving and running programs in IDLE
- The `print()` function
- The `input()` function
- Comments
- Naming Convention
- Case-sensitivity