



香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen

# Introduction to Computer Science: Programming Methodology

## Lecture 2 Python Basics

Tongxin Li

School of Data Science

# **Last week**

Computer Systems

Data Representation and Conversion

**This week:** Python Basics

# Question

Is Von Neumann architecture the only one possible?

# Example



## Two paradigms for intelligence

### The logic-inspired approach

The essence of intelligence is reasoning.

This is done by using symbolic rules to manipulate symbolic expressions.

- Learning can wait. Understanding how knowledge is represented must come first.

### The biologically-inspired approach

The essence of intelligence is learning the strengths of the connections in a neural network.

- Reasoning can wait. Understanding how learning works must come first.

**Hinton's  
Slides**

# Example

## A new type of computer

- Computers were designed to faithfully implement instructions because it was assumed that the only way to get a general purpose computer to solve a specific task was to tell it exactly what to do.
- This is no longer true, but the research community has been slow to realize the implications of deep learning for how computers are built.
  - We will see a completely new type of computer.
  - It will not replace digital computers.

# Example

## Mortal computation

- If we abandon immortality and accept that the knowledge is inextricable from the precise physical details of a specific piece of hardware, we get two big benefits:
  - We can use very low power analog computation.
  - We can grow hardware whose precise connectivity and analog behavior are unknown.
- The learning procedure running in a particular piece of hardware must learn to make use of the specific properties of that particular piece of hardware, without knowing what all those properties are.
  - FF is a promising candidate for the learning procedure.



# Session Assistant

Jielong Xujun

[222043010@link.cuhk.edu.cn](mailto:222043010@link.cuhk.edu.cn)

[Schedule an appointment](#)



**Parseltongue** is the language of serpents and those who can converse with them. An individual who can speak Parseltongue is known as a **Parselmouth**. It is very uncommon skill, and may be hereditary. Nearly all known Parselmouths are descended from Salazar Slytherin.

[Http://harrypotter.wikia.com/wiki/Parseltongue](http://harrypotter.wikia.com/wiki/Parseltongue)





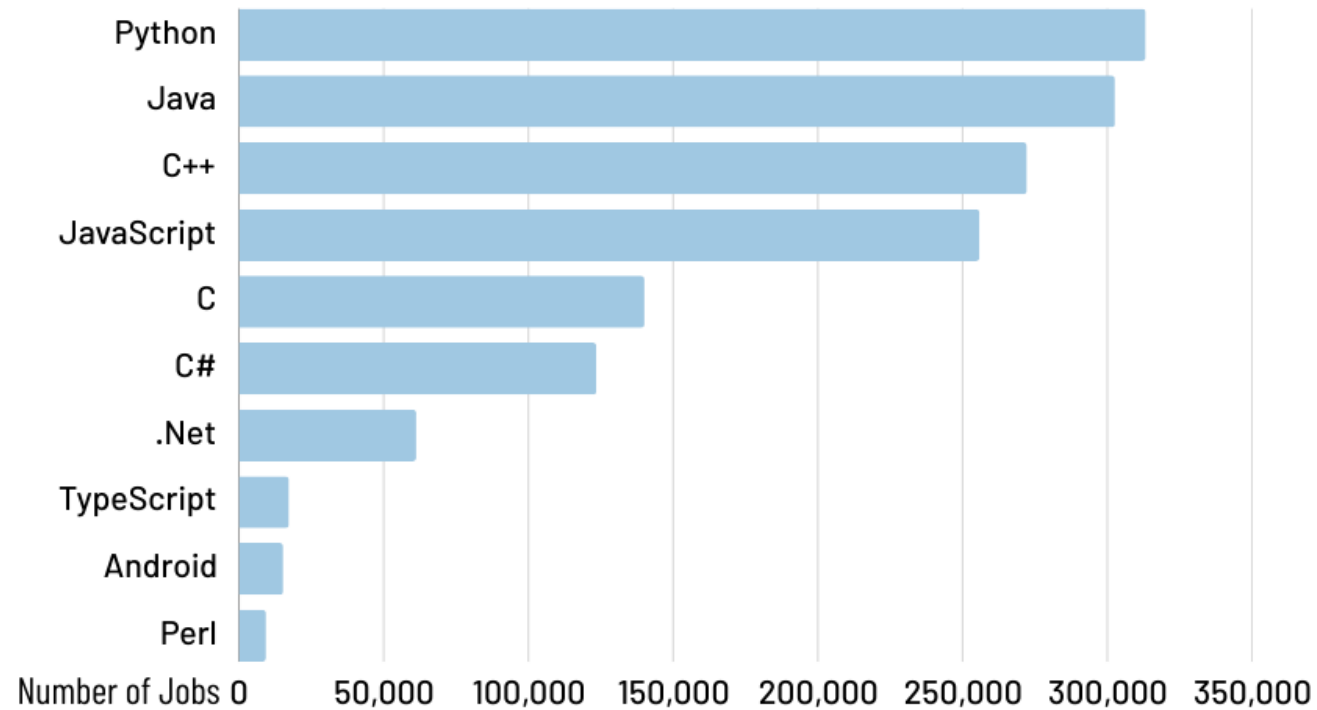
**Python** is the language of Python interpreter and those who can converse with them. An individual who can speak Python is known as a **Pythonista**. It is very uncommon skill, and may be hereditary. Nearly all known Pythonistas use software initially developed by Guido van Rossum



# Most popular programming languages

## Most in-demand programming languages of 2024

Based on LinkedIn job postings in the US

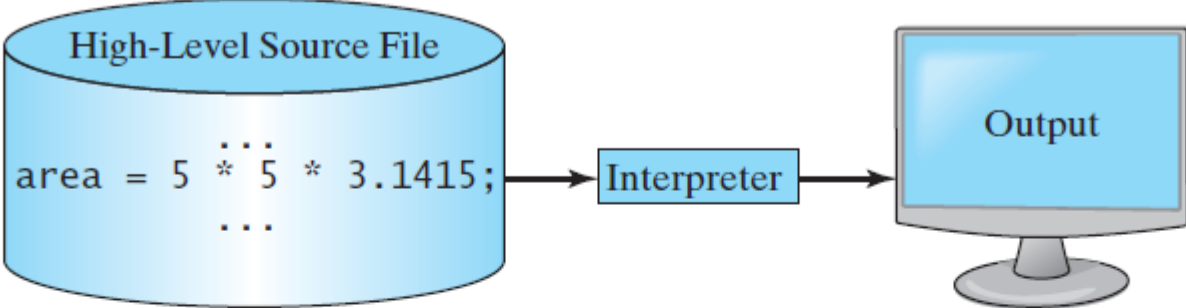


By: CodingNomads

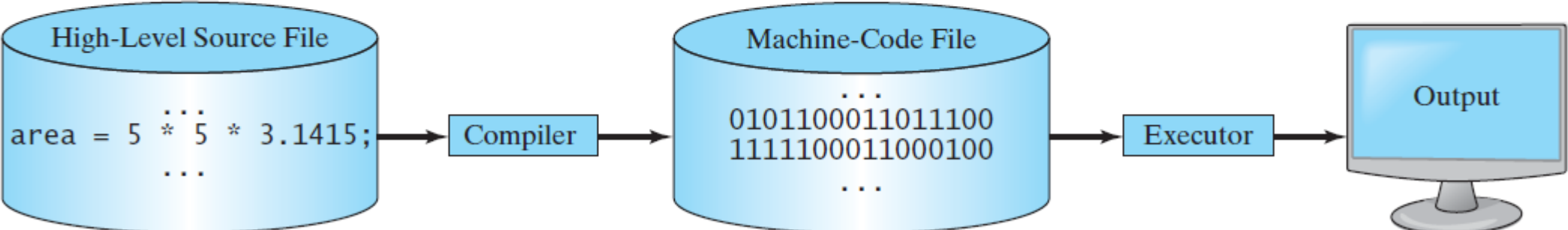
# Interpreter v.s. compiler

- **Interpreter (解释器)** is a computer program that **directly** executes, i.e. performs, instructions written in a programming or scripting language, **without previously compiling** them into a machine language program
- **A compiler (编译器)** is a computer program (or a set of programs) that transforms source code written in a programming language (the **source language**) into another computer language (the **target language**), with the latter often having a binary form known as **object code**

# Interpreter v.s. compiler



(a)

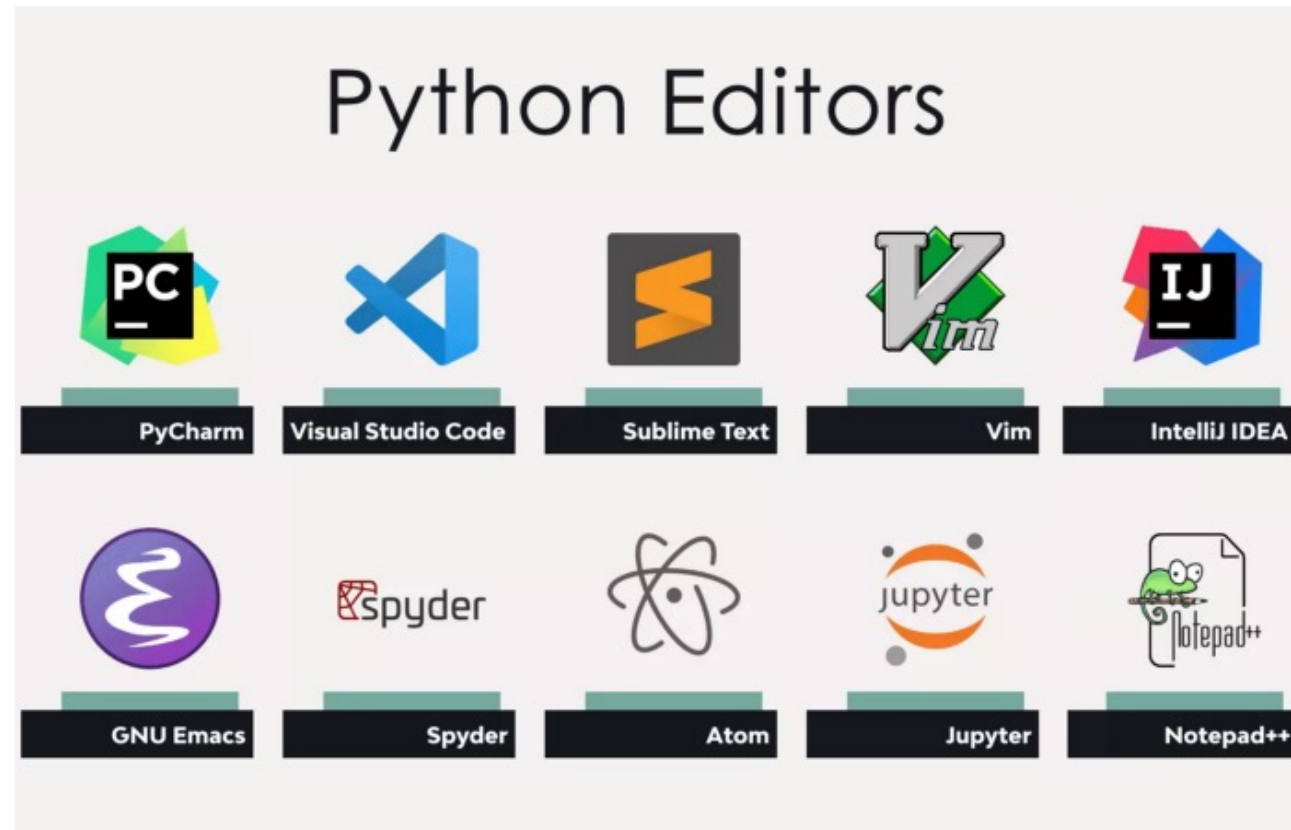


(b)

# Early learner: syntax error

- We need to learn the Python language so we can communicate our instructions to Python. In the beginning we will make lots of mistakes and speak gibberish like small children
- When you make a mistake, the computer does not think you are “cute”. It says “**syntax error**” – given that it “knows” the language and you are just learning it. It seems like Python is cruel and unfeeling
- You must remember that **you are intelligent and can learn**, while the computer is simple and very fast – **but cannot learn**
- It is **easier** for you to learn Python than for the computer to learn human language

# Integrated development environment



# Python Versions

Feel free to use any python version in your assignments/exams as long as it compiles

- [Python 3.13.0rc2](#), documentation released on 6 September 2024. [**pre-release candidate**]
- [Python 3.12.6](#), documentation released on 6 September 2024.
- [Python 3.12.5](#), documentation released on 6 August 2024.
- [Python 3.12.4](#), documentation released on 6 June 2024.
- [Python 3.12.3](#), documentation released on 9 April 2024.
- [Python 3.12.2](#), documentation released on 6 February 2024.
- [Python 3.12.1](#), documentation released on 8 December 2023.
- [Python 3.12.0](#), documentation released on 2 October 2023.
- [Python 3.11.10](#), documentation released on 6 September 2024.
- [Python 3.11.9](#), documentation released on 2 April 2024.
- [Python 3.11.8](#), documentation released on 6 February 2024.
- [Python 3.11.7](#), documentation released on 4 December 2023.
- [Python 3.11.6](#), documentation released on 2 October 2023.



# Installing Python

The image shows a screenshot of the Python.org website. At the top, there is a navigation bar with links for Python, PSF, Docs, PyPI, Jobs, and Community. Below this is the Python logo and a search bar with a 'GO' button. To the right of the search bar are links for 'Socialize' and 'Sign In'. A secondary navigation bar contains links for 'About', 'Downloads', 'Documentation', 'Community', 'Success Stories', 'News', and 'Events'. The main content area features a code snippet on the left and a text block on the right. The code snippet is a Python 3 script that calculates the Fibonacci series up to n. The text block is titled 'Functions Defined' and explains that the core of extensible programming is defining functions, mentioning mandatory and optional arguments, keyword arguments, and arbitrary argument lists. It includes a link to 'More about defining functions in Python 3'. At the bottom of the text block are five numbered buttons (1-5). Below the code and text is a footer that says 'Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)'.

```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

**Functions Defined**

The core of extensible programming is defining functions. Python allows mandatory and optional arguments, keyword arguments, and even arbitrary argument lists. [More about defining functions in Python 3](#)

1 2 3 4 5

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)

<https://www.python.org>

# Installing Python



The screenshot shows the Python.org website with a dark blue header. The navigation menu includes links for Python, PSF, Docs, PyPI, Jobs, and Community. The main content area features the Python logo, a search bar, and a secondary navigation bar with links for About, Downloads, Documentation, Community, Success Stories, News, and Events. The primary focus is on the 'Downloads' section, which offers links to download Python 3.5.1 and Python 2.7.11 for Windows. Below these links, there are informational links regarding version differences, OS support, and pre-releases. An illustration of two parachutes carrying crates is positioned to the right of the download links.

Python PSF Docs PyPI Jobs Community

python™

Search GO Socialize Sign In

About Downloads Documentation Community Success Stories News Events

## Download the latest version for Windows

[Download Python 3.5.1](#) [Download Python 2.7.11](#)

Wondering which version to use? [Here's more about the difference between Python 2 and 3.](#)

Looking for Python with a different OS? Python for [Windows](#), [Linux/UNIX](#), [Mac OS X](#), [Other](#)

Want to help test development versions of Python? [Pre-releases](#)

Looking for a specific release?

Python releases by version number:

Python 3 v.s. Python 2 ?

## PYTHON 2

## PYTHON 3

← **Legacy**

It is still entrenched in the software at certain companies

 **Library**

Many older libraries built for Python 2 are not forwards-compatible

0100  
0001 **ASCII**

Strings are stored as ASCII by default

 **5/2=2**

It rounds your calculation down to the nearest whole number

**print "hello"**

Python 2 print statement

>

**Future** →

It will take over Python 2 by 2020

**Library** 

Many of today's developers are creating libraries strictly for use with Python 3

**Unicode** 

Text strings are Unicode by default

+

**5/2=2.5** 

The expression 5 / 2 will return the expected result

≠

**print ("hello")**

The print statement has been replaced with a print () function

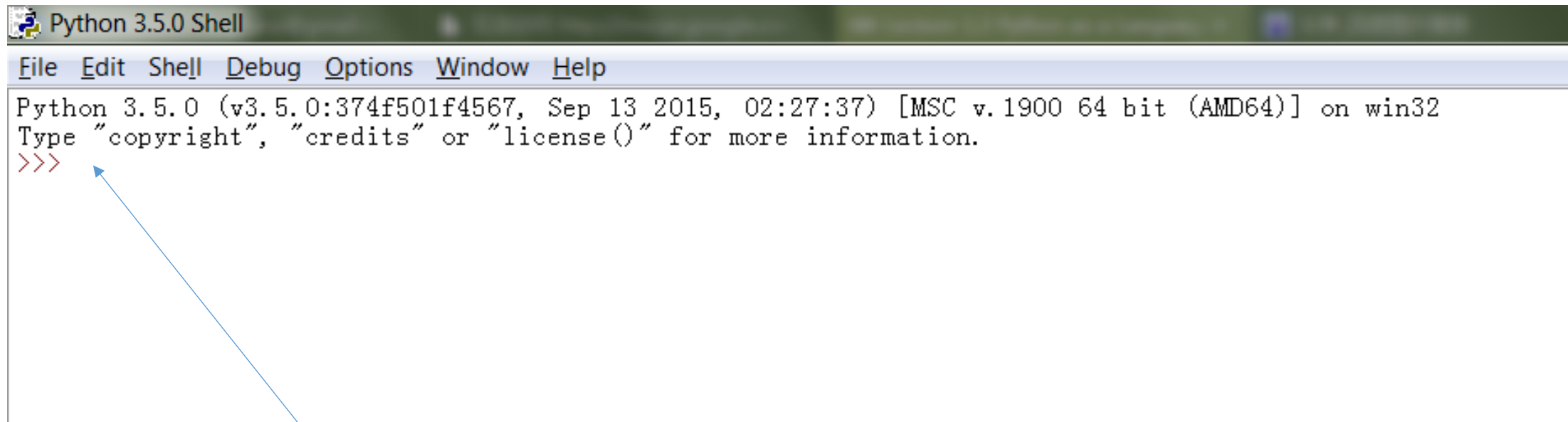
**Range function**

**Unicode support**

**Integer division**

**The new print() function**

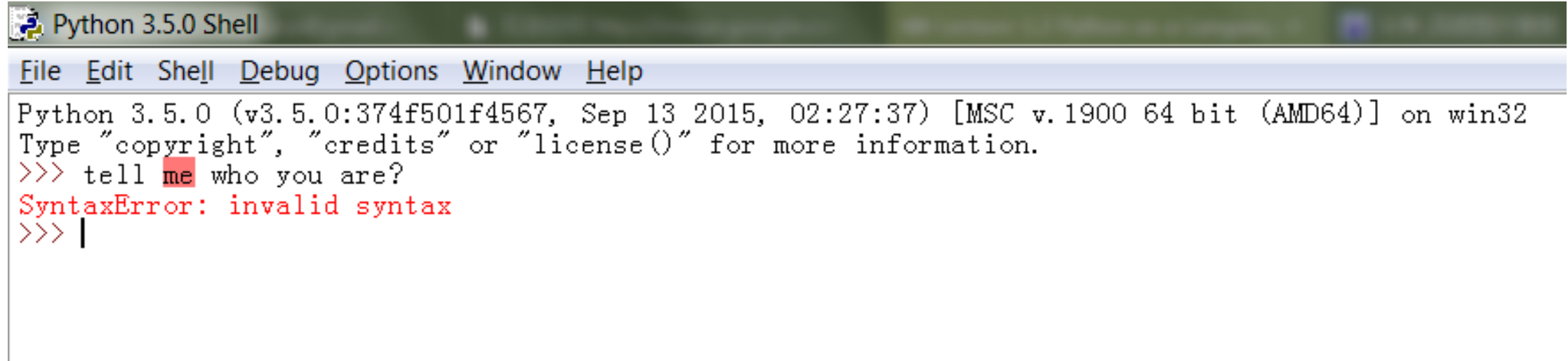
# Python Shell



```
Python 3.5.0 Shell
File Edit Shell Debug Options Window Help
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

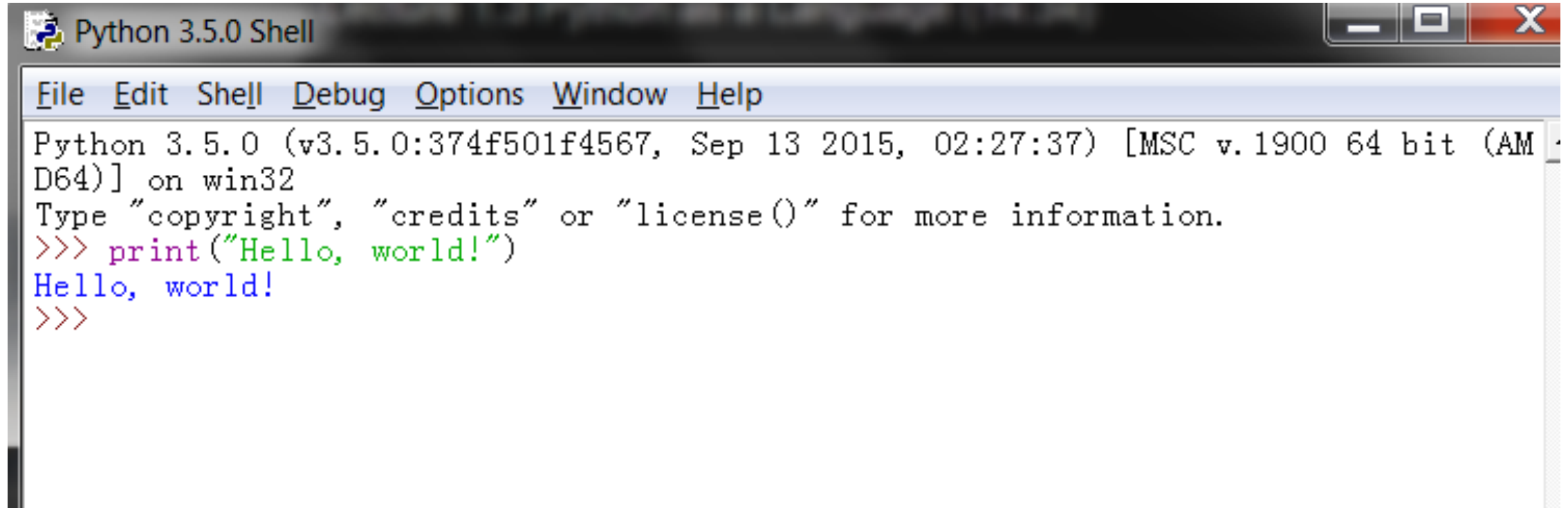
What is next?

# Syntax Error



```
Python 3.5.0 Shell
File Edit Shell Debug Options Window Help
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> tell me who you are?
SyntaxError: invalid syntax
>>> |
```

# Hello, world!

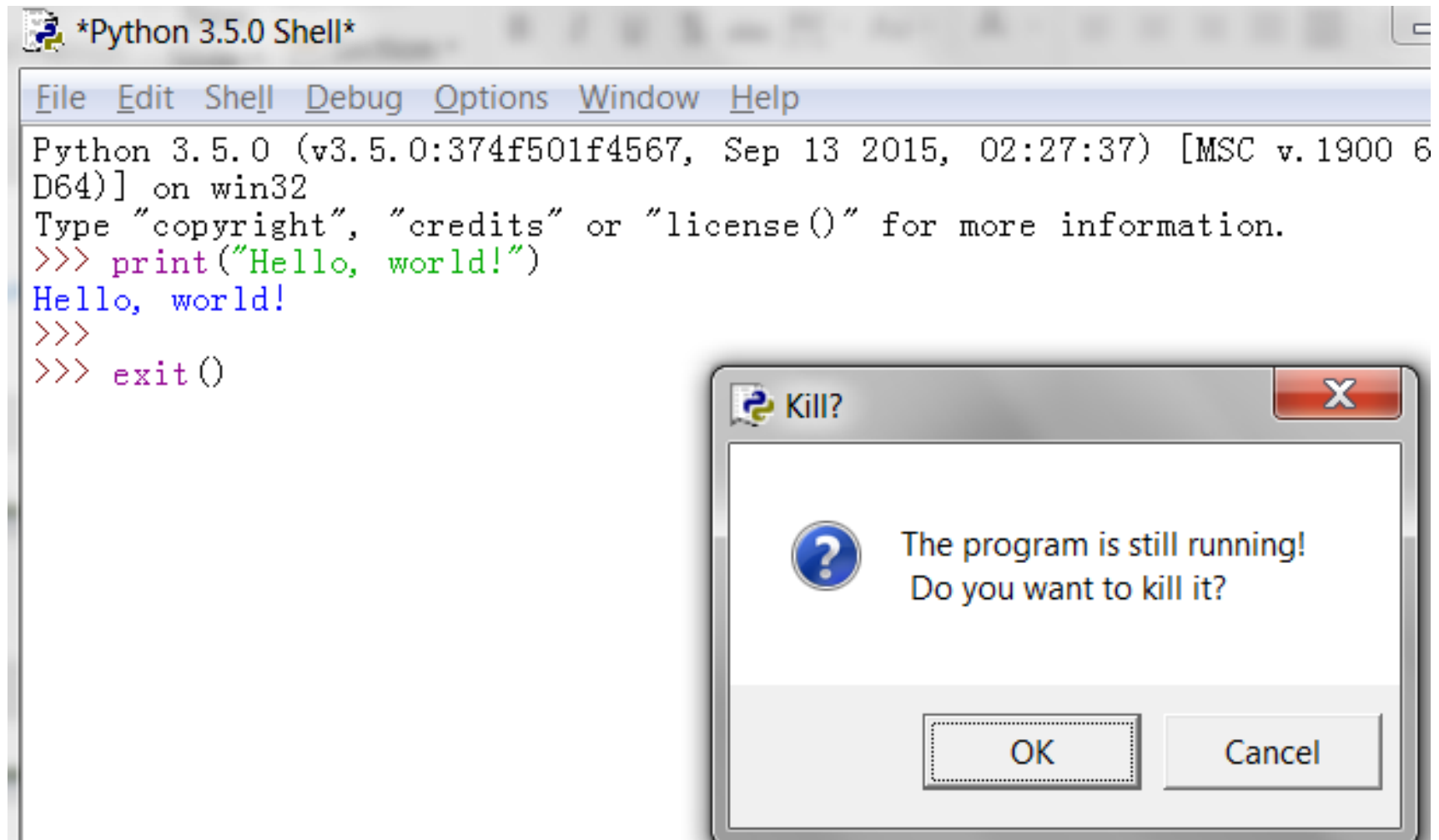


```
Python 3.5.0 Shell
File Edit Shell Debug Options Window Help
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello, world!")
Hello, world!
>>>
```

- You must say something that Python interpreter can understand!!
- `Print()` is a **function** in Python



# Exit()



The image shows a Python 3.5.0 Shell window with a menu bar (File, Edit, Shell, Debug, Options, Window, Help) and a command prompt. The prompt displays the Python version and build information, followed by instructions to use 'copyright', 'credits', or 'license()' for more information. The user has entered the command `print("Hello, world!")`, which has been executed, resulting in the output `Hello, world!`. The user has then entered `exit()`. A 'Kill?' dialog box is overlaid on the shell window, containing a question mark icon and the text 'The program is still running! Do you want to kill it?'. The dialog box has 'OK' and 'Cancel' buttons at the bottom.

```
*Python 3.5.0 Shell*
File Edit Shell Debug Options Window Help
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 6
D64] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello, world!")
Hello, world!
>>>
>>> exit()
```

Kill?

The program is still running!  
Do you want to kill it?

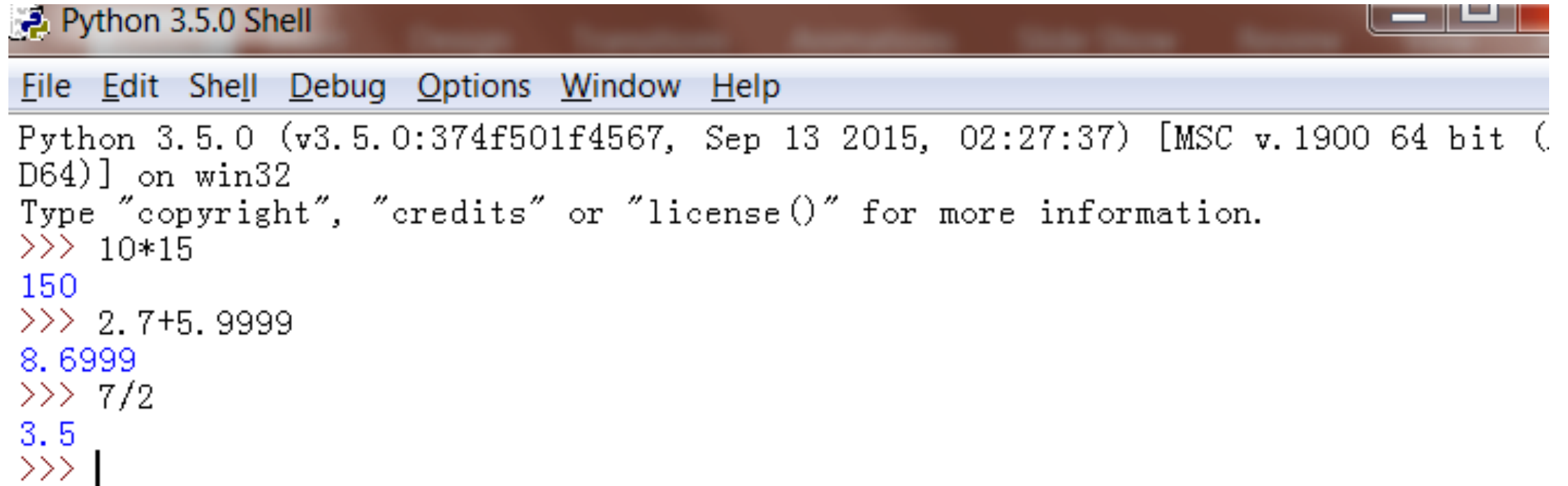
OK Cancel

What should we say to Python ?

# Elements of Python Language

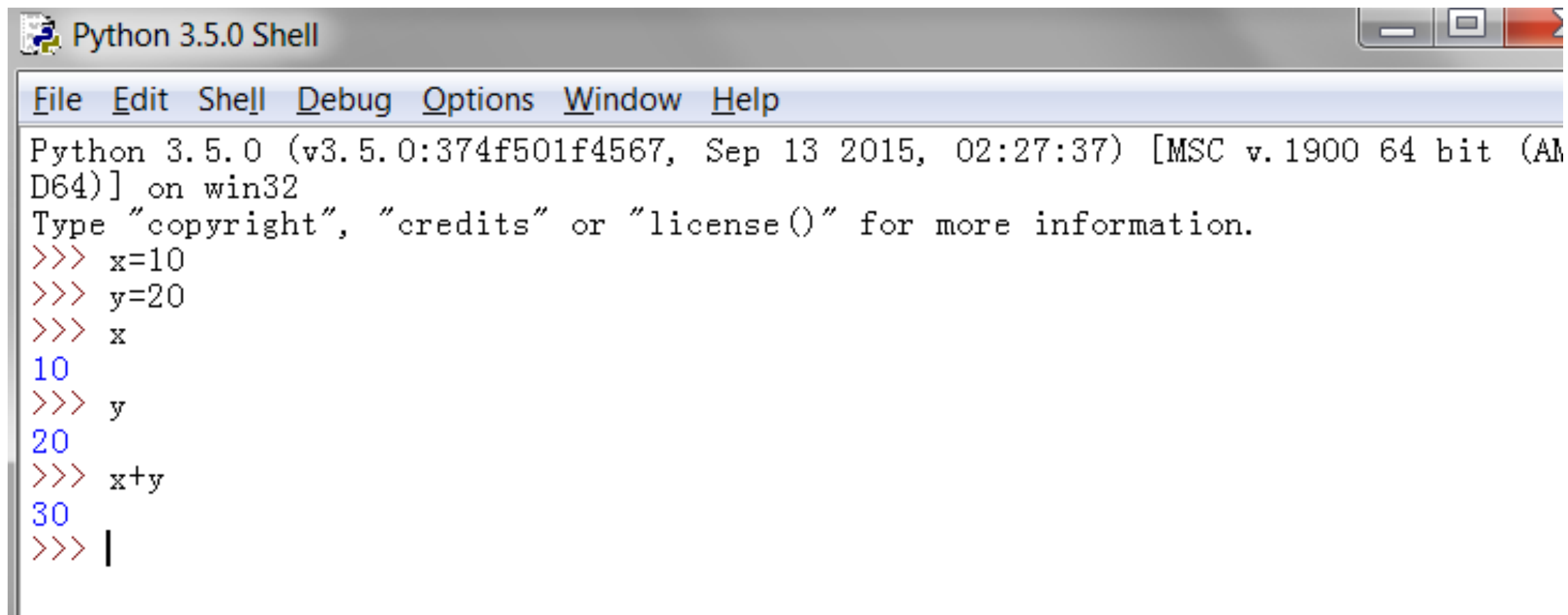
- Vocabulary/words – **Variables** and **Reserved words**
- Sentence structure – valid **syntax patterns**
- Story structure – constructing a **meaningful program** for some **purposes**

# Use Python as a calculator



```
Python 3.5.0 Shell
File Edit Shell Debug Options Window Help
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 10*15
150
>>> 2.7+5.9999
8.6999
>>> 7/2
3.5
>>> |
```

# Variables



```
Python 3.5.0 Shell
File Edit Shell Debug Options Window Help
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x=10
>>> y=20
>>> x
10
>>> y
20
>>> x+y
30
>>> |
```

# Reserved words

- You **cannot** use the following words as **variables**

False	None	True	and	as	assert	break
class	continue	def	del	elif	else	except
finally	for	from	global	if	import	in
is	lambda	nonlocal	not	or	pass	raise
return	try	while	with	yield		

# Sentences or lines

```
>>> x=2  
>>> x=x+2  
>>> print(x)  
4  
>>>
```

← Assignment statement

← Assignment with expressions

← Print statement (output statement)



# Programming scripts

- **Interactive Python** is good for experiments and programs of 3-4 lines long
- Most programs are **much longer**, so we have to type them **in a file** and **execute them all together**
- In this sense, we are giving Python a **script**
- As convention, **“.py”** is added as the **suffix** on the end of these files

# Interactive v.s. script

- Interactive

- ✓ You type directly to Python one line at a time and it responds

- Script

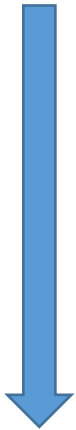
- ✓ You enter a sequence of statements (lines) into a file using a text editor and tell Python to execute the file

# Program steps or program flow

- Like a recipe, a program is a **sequence of steps** to be done in **pre-determined order**
- Some steps are **conditional**, i.e. they may be skipped
- Sometimes, we will **repeat** some steps
- Sometimes, we **store** a set of steps to be used over and over again in future as needed

# Sequential flow

Execute sequentially



```
>>> x=2
>>> print(x)
2
>>> x=x*10
>>> print(x)
20
>>> |
```

Outputs

- When a program is running, it flows from one step to the next
- We as programmers, set up “**paths**” for the program to follow

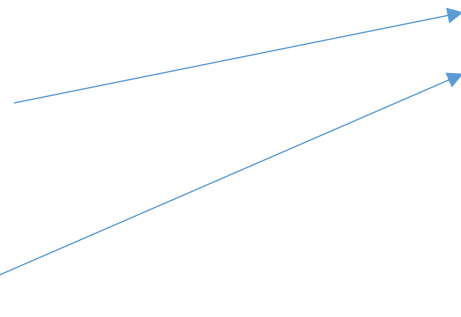
# Conditional flow

## Program

```
x=5
if x<10:
    print("smaller")
if x>20:
    print("bigger")
print("finished")
```

## Outputs

```
smaller
finished
>>> |
```



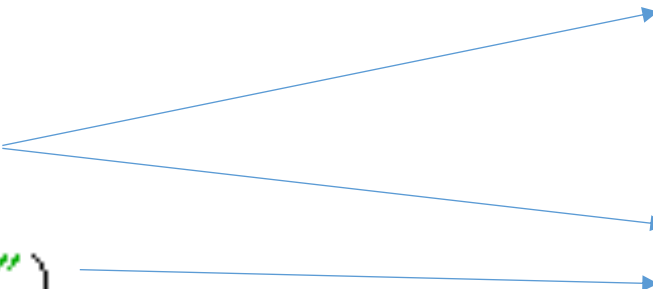
# Repeated flow

## Program

```
n=5
while n>0:
    print(n)
    n = n - 1
print("Finish")
```

## Outputs

```
5
4
3
2
1
Finish
>>>
```



- **Loops (repeated steps)** have **iterative variables** that change each time through a loop
- Often these iterative variables go through **a sequence of numbers**

# What the largest number is?

```
25  1  114 117 150 152 120  46  19 126
191 121 104 116 160 105  89 125  40  14
 31 139 113  94  97 193 154 140 195 122
112 163 177  48  78 101 130  83  35 197
 44  54 106 143  59  38  3  41  93  81
 20 164  4  11 131  0 107  71 159  69
181 178 173 148  62 142 170  72  37 145
 60 187 198  99  15  82  26  8 192  17
129  73  45  9  24 188  42 151  51 183
179  79  50  76  34  33 185 102 193 184
```

# What the largest number is?

25	1	114	117	150	152	120	46	19	126
191	121	104	116	160	105	89	125	40	14
31	139	113	94	97	193	154	140	195	122
112	163	177	48	78	101	130	83	35	197
44	54	106	143	59	38	3	41	93	81
20	164	4	11	131	0	107	71	159	69
181	178	173	148	62	142	170	72	37	145
60	187	198	99	15	82	26	8	192	17
129	73	45	9	24	188	42	151	51	183
179	79	50	76	34	33	185	102	193	184



# Constants

- Fixed values such as numbers and letters are called **constants**, since their values won't change
- **String** constants use single-quotes (') or double-quotes (")

# Variable

- A variable is a **named space** in the **memory** where a programmer can store **data** and later retrieve the data using the **variable name**
- Variable names are determined by programmers
- The **value** of a variable can be **changed later** in a program

# Rules for defining variables in Python

- Must start with a letter or underscore \_
- Can **only** contain letters, numbers and underscore
- Case **sensitive**
- **Good**: apple, car, myNumber123, \_light
- **Bad**: 456aaa, #ab, var.12
- **Different**: apple, Apple, APPLE

# Personal tips

- Use **meaningful words** as variable names
- Start with a **lower letter**
- Capitalize the **first letter** of each word
- **Example**: myBankAccountID, numOfCards, salaryAtYear1995...

What is this code doing?

```
x1q3z9ocd = 35.0  
x1q3z9afd = 12.50  
x1q3p9afd = x1q3z9ocd * x1q3z9afd  
print x1q3p9afd
```

# Reserved words

- You **cannot** use the following words as **variables**

False	None	True	and	as	assert	break
class	continue	def	del	elif	else	except
finally	for	from	global	if	import	in
is	lambda	nonlocal	not	or	pass	raise
return	try	while	with	yield		

# Sentences or lines

```
>>> x=2          ← Assignment statement
>>> x=x+2        ← Assignment with expressions
>>> print(x)     ← Print statement (output statement)
4
>>>
```

Variable    Operator    Constant    Reserved words

# Assignment statement

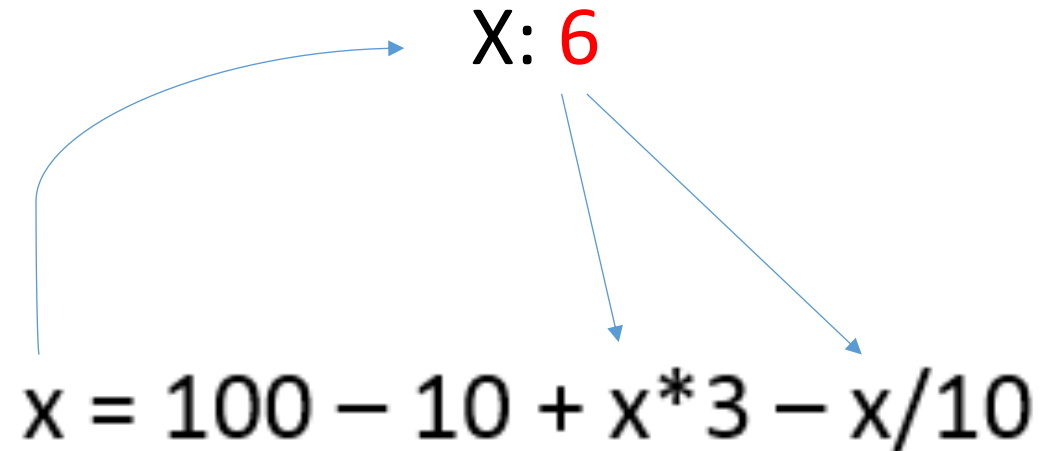
- We assign a value to a variable using the **assignment operator (=)**
- An assignment statement consists of an **expression on the right hand side**, and a **variable** to store the result

**Example:**  $x = 100 - 10 + x * 3 - x / 10$



# Assignment statement

- There is a location in the memory for  $x$
- Whenever the value of  $x$  is needed, it can be retrieved from the memory
- After the expression is evaluated, the result will be put back into  $x$



# Cascaded assignment

- We can set multiple variables into the same value using a single assignment statement

## Example

```
>>> z = y = x = 2 + 7 + 2
>>> x, y, z
(11, 11, 11)
```

# Simultaneous assignment

- The values of two variables can be exchanged using simultaneous assignment

## Example

```
>>> c = "deepSecret"           # Set current password.
>>> o = "you'll never guess"   # Set old password.
>>> c, o                        # See what passwords are.
('deepSecret', 'you'll never guess')
>>> c, o = o, c                # Exchange the passwords.
```

# Practice

- Write a program to exchange the values of two variables **without** using simultaneous assignment

# Bad use of simultaneous assignment

```
>>> # A bad use of simultaneous assignment.
>>> x, y = (45 + 34) / (21 - 4), 56 * 57 * 58 * 59
>>> x, y
(4.647058823529412, 10923024)
>>> # A better way to set the values of x and y.
>>> x = (45 + 34) / (21 - 4)
>>> y = 56 * 57 * 58 * 59
>>> x, y
(4.647058823529412, 10923024)
```

# Order evaluation

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

**Example:**  $X = 1 + 2 * 3 - 4 / 5 ** 6$

# Numeric expression and operators

- We use some keys we have on the keyboard to denote the classic math operators
- **Asterisk** (\*) is the multiplication operator
- **Double asterisk** (\*\*) is used to denote Exponentiation (raise to a power)

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

# Operator precedence rules

- **Highest to lowest precedence rule**

- ✓ Parenthesis are always with highest priority
- ✓ Power
- ✓ Multiplication, division and remainder
- ✓ Addition and subtraction
- ✓ Left to right





# Operator precedence

Example:  $x = 1 + 2 * 3 / 4 * 5$

# Floor division

```
>>> time = 257          # Time in seconds.
>>> minutes = time // 60 # Number of complete minutes in time.
>>> print("There are", minutes, "complete minutes in", time, "seconds.")
There are 4 complete minutes in 257 seconds.
>>> 143 // 25
5
>>> 143.4 // 25
5.0
>>> 9 // 2.5
3.0
```

# divmod()

```
>>> time = 257          # Initialize time.
>>> SEC_PER_MIN = 60   # Use a "named constant" for 60.
>>> divmod(time, SEC_PER_MIN) # See what divmod() returns.
(4, 17)
>>> # Use simultaneous assignment to obtain minutes and seconds.
>>> minutes, seconds = divmod(time, SEC_PER_MIN)
>>> # Attempt to display the minutes and seconds in "standard" form.
>>> print(minutes, ":", seconds)
4 : 17
>>> # Successful attempt to display time "standard" form.
>>> print(minutes, ":", seconds, sep="")
4:17
>>> # Obtain number of quarters and leftover change in 143 pennies.
>>> quarters, cents = divmod(143, 25)
>>> quarters, cents
(5, 18)
```

# Augmented assignment

- The general form of augmented assignment looks like

`<lvalue> <op>= <expression>`

## Example

```
>>> x = 22          # Initialize x to 22.
>>> x += 7          # Equivalent to: x = x + 7
>>> x
29
>>> x -= 2 * 7      # Equivalent to: x = x - (2 * 7)
>>> x
15
```

# Personal tips

- Use **parentheses**
- Keep mathematical expressions **simple** so that they are easy to understand
- **Break up** long series of math expressions to make them easy to understand

# Integer division in Python

- In Python 2, Integer division **truncates**
- Integer division produces floating point numbers in Python 3
- The **conversion** between integer and floating point numbers is done **automatically** in Python 3
- Things change between Python 2 and Python 3

# Data Type

- In Python, variables and constants have an associated “**type**”
- Python **knows the difference** between a number and a string

- **Example:**

```
>>> a = 100 + 200
>>> print(a)

>>> b = "100" + "200"
>>> print(b)
```

# Type matters

- Python knows what type everything is
- Some operations are **prohibited** on certain types
- You cannot “**add 1**” to a string
- We can **check the type** of something using function `type()`



# Types of numbers

- Numbers in Python generally have **two types**:
  - ✓ Integers: 1, 2, 100, -20394209
  - ✓ Floating point numbers: 2.5, 3.7, 11.32309, -30.999
- There are other number types, which are variations on float and integer

# Type can change

- The type of a variable can be **dynamically changed**
- A variable's type is determined by the value that is **last assigned to the variable**

```
>>> x = 7 * 3 * 2
>>> y = "is the answer to the ultimate question of life"
>>> print(x, y)           # Check what x and y are.
42 is the answer to the ultimate question of life
>>> x, y                 # Quicker way to check x and y.
(42, 'is the answer to the ultimate question of life')
>>> type(x), type(y)    # Check types of x and y.
(<class 'int'>, <class 'str'>)
>>> # Set x and y to new values.
>>> x = x + 3.14159
>>> y = 1232121321312312312312 * 9873423789237438297
>>> print(x, y)         # Check what x and y are.
45.14159 12165255965071649871208683630735493412664
>>> type(x), type(y)   # Check types of x and y.
(<class 'float'>, <class 'int'>)
```

# Type conversion

- When an expression contains both integer and float, integers will be converted into float **implicitly**
- You can control this using functions `int()` and `float()`

- **Example:**

```
>>> print(float(99)/100)
```

```
>>> i=42
```

```
>>> type(i)
```

```
>>> f=float(i)
```

```
>>> print(f)
```

```
>>> type(f)
```

```
>>> print(1+2*float(3)/4-5)
```

# String conversions

- You can also use `int()` and `float()` to convert strings into numbers
- You will **get an error** if the string contains characters other than numbers

# Converting numbers into string

- We can convert numbers into string using function `str()`

```
>>> str(5) # Convert int to a string.
'5'
>>> str(1 + 10 + 100) # Convert int expression to a string.
'111'
>>> str(-12.34) # Convert float to a string.
'-12.34'
>>> str("Hello World!") # str() accepts string arguments.
'Hello World!'
>>> str(divmod(14, 9)) # Convert tuple to a string.
'(1, 5)'
>>> x = 42
>>> str(x) # Convert int variable to a string.
'42'
```

# User input

- We can instruct Python to **stop and take user inputs** using function `input()`
- The `input()` function returns a **string**

# Practice

- The BMI (body mass index) of a human can be calculated using the following equation:

$$\text{BMI} = \text{weight (kg)} \div \text{height}^2 \text{ (m)}$$

- Write a program to input a user's weight and height, and then output his BMI



# Converting user input

- If we want to read a number using `input()`, we must then convert the input into a number using `int()` or `float()`
- Later we will deal with bad input data



# Comments

- Anything after a “#” is ignored by Python
- Why comment?
  - ✓ Describe **what is going to happen** in a sequence of code
  - ✓ Document **who wrote the code** and other important information
  - ✓ **Turn off** a line of code – usually temporarily

# String operations

- Some operators **apply to strings**

- ✓ “+”: concatenation

- ✓ “\*”: multiple concatenation

- Python **knows** whether it is dealing with a number or a string

# Practice

- Write a program to instruct the user to input two of his friends' names, and then output a sentence "I am the friend of XX and XX."

# Output using Print()

```
>>> print(42, "42") # An int and a str that looks like an int.  
42 42  
>>> print('3.14') # A str that looks like a float.  
3.14  
>>> print(3.14) # A float.  
3.14
```

# More details on print()

```
print(...)  
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

Prints the values to a stream, or to `sys.stdout` by default.

Optional keyword arguments:

`file`: a file-like object (stream); defaults to the current `sys.stdout`.

`sep`: string inserted between values, default a space.

`end`: string appended after the last value, default a newline.

`flush`: whether to forcibly flush the stream.

# Examples

```
>>> print("I", "am", "Daniel")
```

```
>>> print("I", "am", "Daniel", sep="")
```

```
>>> print("I", "am", "Daniel", sep=",")
```

## Example

```
print ("Test line 1")  
print ("Test line 2")
```

```
print ("Test line 1", end = " ")  
print ("Test line 2")
```

```
print ("Test line 1", end = "----")  
print ("Test line 2")
```

# A powerful function - eval()

- The eval() function takes a string argument and evaluates that string **as a Python expression**, i.e., just as if the programmer had directly entered the expression as code
- The function returns **the result of that expression**
- Eval() gives the programmers the **flexibility** to determine what to execute **at run-time**
- One should be **cautious** about using it in situations where users could potentially cause problems with “inappropriate” inputs



# Example

```
>>> string = "5 + 12" # Create a string.
>>> print(string)      # Print the string.
5 + 12
>>> eval(string)       # Evaluate the string.
17
>>> print(string, "=", eval(string))
5 + 12 = 17
>>> eval("print('Hello World!')") # Can call functions from eval().
Hello World!
>>> # Using eval() we can accept all kinds of input...
>>> age = eval(input("Enter your age: "))
Enter your age: 57.5
>>> age
57.5
>>> age = eval(input("Enter your age: "))
Enter your age: 57
>>> age
57
>>> age = eval(input("Enter your age: "))
Enter your age: 40 + 17 + 0.5
>>> age
57.5
```

# Example

```
>>> eval("10, 32")           # String with comma-separated values.
(10, 32)
>>> x, y = eval("10, 20 + 12") # Use simultaneous assignment.
>>> x, y
(10, 32)
>>> # Prompt for multiple values. Must separate values with a comma.
>>> x, y = eval(input("Enter x and y: "))
Enter x and y: 5 * 2, 32
>>> x, y
(10, 32)
```