California State University Fullerton

CPSC-235P
Python Programming

Stephen T. May

Python Tutorial
Section 1: Whetting Your Appetite
Section 2: Using the Python Interpreter
Section 3: An Informal Introduction to Python

https://docs.python.org/release/3.10.2/tutorial/index.html

Slide Notes

- Command typed at the Linux command prompt ($)

```
$ python3.9
```

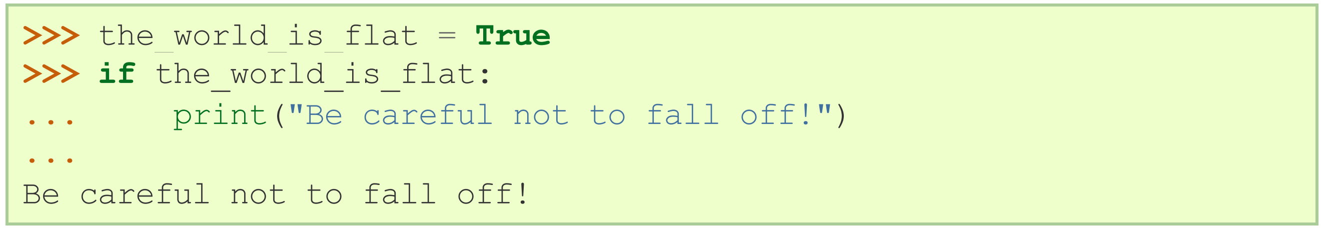
- Command typed at the Python interpreter command prompt (>>>)

```
>>> Ctrl-D
```

- Python source code

```
print(“Hello world!”)
```

- Mixed example



```
>>> the_world_is_flat = True
>>> if the_world_is_flat:
...     print("Be careful not to fall off!")
...
Be careful not to fall off!
```

1 Whetting Your Appetite

- Python is a very-high-level language
- It has high-level data types built in
  - flexible arrays
  - dictionaries
  - lists
- Allows the program to be split into modules
  - Large collection of standard modules
  - Programmer defined modules can be reused in other Python programs
- Python is an interpreted language
  - No compilation or linking required
  - Less development time
  - Execution slower than compiled binary
  - Execution faster than Unix script
  - Interactive interpreter for quick code
- Much shorter code than C++
  - High-level data types allow complex operations in a single statement
  - Indentation used for blocks of code rather than beginning and ending brackets
  - No variable declaration needed
  - No semicolons needed
- Python is extensible
  - Open source – C code is available and can be extended and recompiled
  - Add additional build-in functions or modules
- Python is named after the BBC show "Monty Python's Flying Circus"
- Dynamically Typed – variable type changes based on the data type held

1 Whetting Your Appetite (cont.)

- Importing Modules

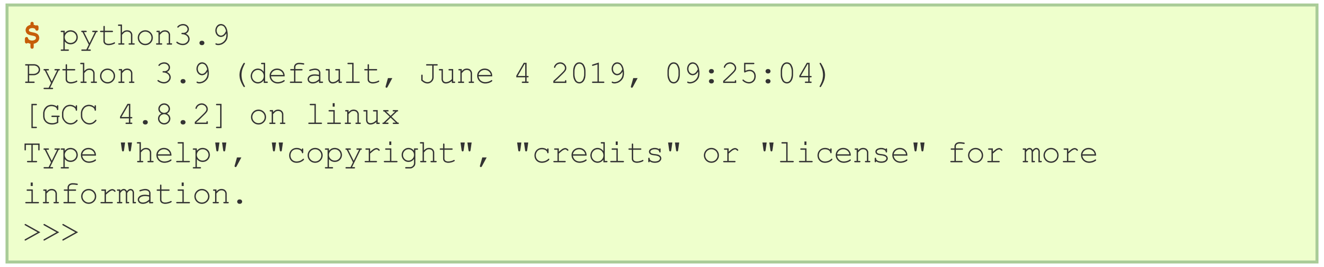
```
import math
print(math.factorial(5))

import math as M
print(M.factorial(5))

from math import factorial
print(factorial(5))

from math import *
print(factorial(5))

print(dir(math))
```

- Upper vs. Lowercase
  - Class names should be uppercase
  - All other names should be lowercase
- Leading underscores
  - Single underscore before a name acts as a week form of privatization
  - Function names with single underscore will not be imported
  - Double underscore before a name in a class prevents subclasses from having conflicts
- Trailing underscores
  - Single underscore after a name is used to avoid naming conflicts such as keywords
  - Double underscore after a name and before a name to indicate a special function i.e. __init__
- Keywords
  - Python Language Reference §2.3.1
- Comments
  - Use the # sign

2.1 Invoking the Interpreter

- Start the Interpreter

  ```
  $ python3.9
  ```



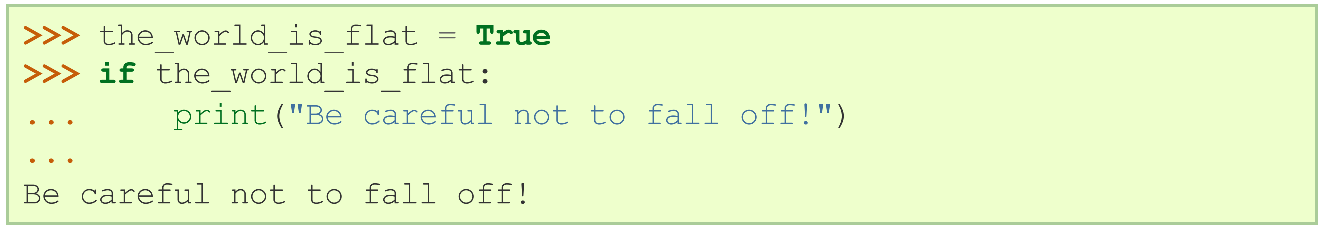
```
$ python3.9
Python 3.9 (default, June 4 2019, 09:25:04)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

- Exit the Interpreter

  ```
  >>> Ctrl-D
  >>> quit()
  ```

- Interactive Editing

  ```
  >>> ↑
  >>> ↓
  >>> ←
  ```



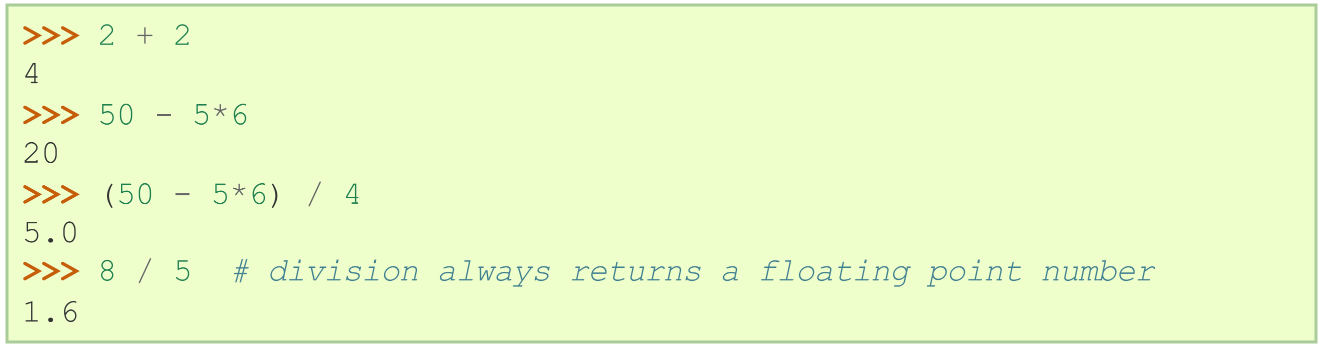
```
>>> the_world_is_flat = True
>>> if the_world_is_flat:
...     print("Be careful not to fall off!")
...
Be careful not to fall off!
```

- Continuation Lines

  ```
  ...     print(“Be careful!”)
  ```

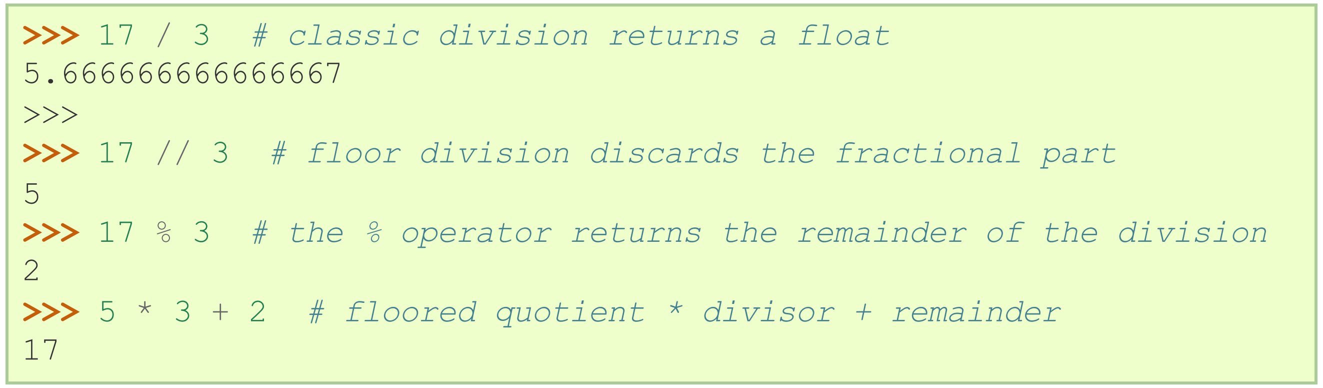
3.1.1 Numbers

- Interpreter acts as a simple calculator
- Usual operators and parentheses
- Whole number are integers
- Decimal numbers are floats



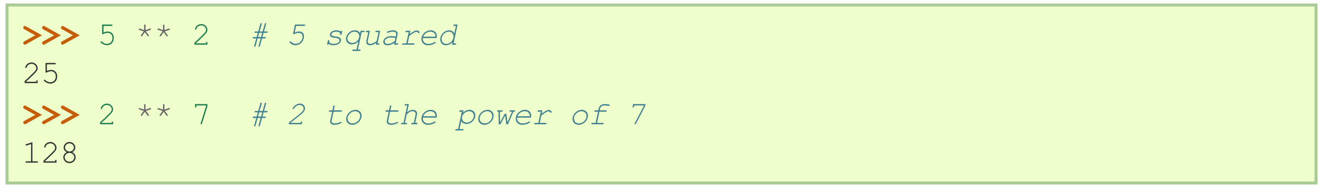
```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # division always returns a floating point number
1.6
```

- / returns float division
- // returns floor (integer) division
- % modulus calculates the remainder



```
>>> 17 / 3  # classic division returns a float
5.666666666666667
>>>
>>> 17 // 3  # floor division discards the fractional part
5
>>> 17 % 3  # the % operator returns the remainder of the division
2
>>> 5 * 3 + 2  # floored quotient * divisor + remainder
17
```

- ** calculates powers



```
>>> 5 ** 2  # 5 squared
25
>>> 2 ** 7  # 2 to the power of 7
128
```

3.1.2 Strings

- Strings can be enclosed in
  - Single quotes ('…')
  - Double quotes ("…")
- \ used to escape quotes

```
>>> 'spam eggs'  # single quotes
'spam eggs'
>>> 'doesn\'t'  # use \' to escape the single quote...
"doesn't"
>>> "doesn't"  # ...or use double quotes instead
"doesn't"
>>> '"Yes," they said.'
'"Yes," they said.'
>>> "\"Yes,\" they said."
'"Yes," they said.'
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
```

- Raw string uses an r before quote

```
>>> print('C:\some\name')  # here \n means newline!
C:\some
ame
>>> print(r'C:\some\name')  # note the r before the quote
C:\some\name
```

- Strings can be concatenated (glued together) with the + operator, and repeated with *

```
>>> # 3 times 'un', followed by 'ium'
>>> 3 * 'un' + 'ium'
'unununium'
```

```
>>> prefix = 'Py'
>>> prefix + 'thon'
'Python'
```

3.1.2 Strings (cont).

- Strings can be indexed (subscripted)
  - First character has index 0

```
>>> word = 'Python'
>>> word[0]  # character in position 0
'P'
>>> word[5]  # character in position 5
'n'
```

- Indices can be negative numbers
  - Start counting from the right

```
>>> word[-1]  # last character
'n'
>>> word[-2]  # second-last character
'o'
>>> word[-6]
'P'
```

- Strings can be sliced
  - Obtains a substring

```
>>> word[:2] + word[2:]
'Python'
>>> word[:4] + word[4:]
'Python'
```

- Built-in function len()
  - Returns the length of a string

```
>>> s = 'supercalifragilisticexpialidocious'
>>> len(s)
34
```

3.1.3 Lists

- Lists are compound data types
  - Comma-separated values between square brackets

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
```

- Lists can be indexed and sliced
  - Slicing returns a new list

```
>>> squares[0]  # indexing returns the item
1
>>> squares[-1]
25
>>> squares[-3:]  # slicing returns a new list
[9, 16, 25]
```

- Lists support concatenation operations

```
>>> squares + [36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

- Lists are mutable

```
>>> cubes = [1, 8, 27, 65, 125]  # something's wrong here
>>> 4 ** 3  # the cube of 4 is 64, not 65!
64
>>> cubes[3] = 64  # replace the wrong value
>>> cubes
[1, 8, 27, 64, 125]
```

3.1.3 Lists (cont.)

- Assignment to slices is possible
  - Can change the size of the list

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> # replace some values
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> # now remove them
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']
>>> # clear the list by replacing all the elements with an empty
list
>>> letters[:] = []
>>> letters
[]
```

- List can be nested

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
```

3.2 First Steps Towards Programming

- Multiple assignment
  - Simultaneously get new values

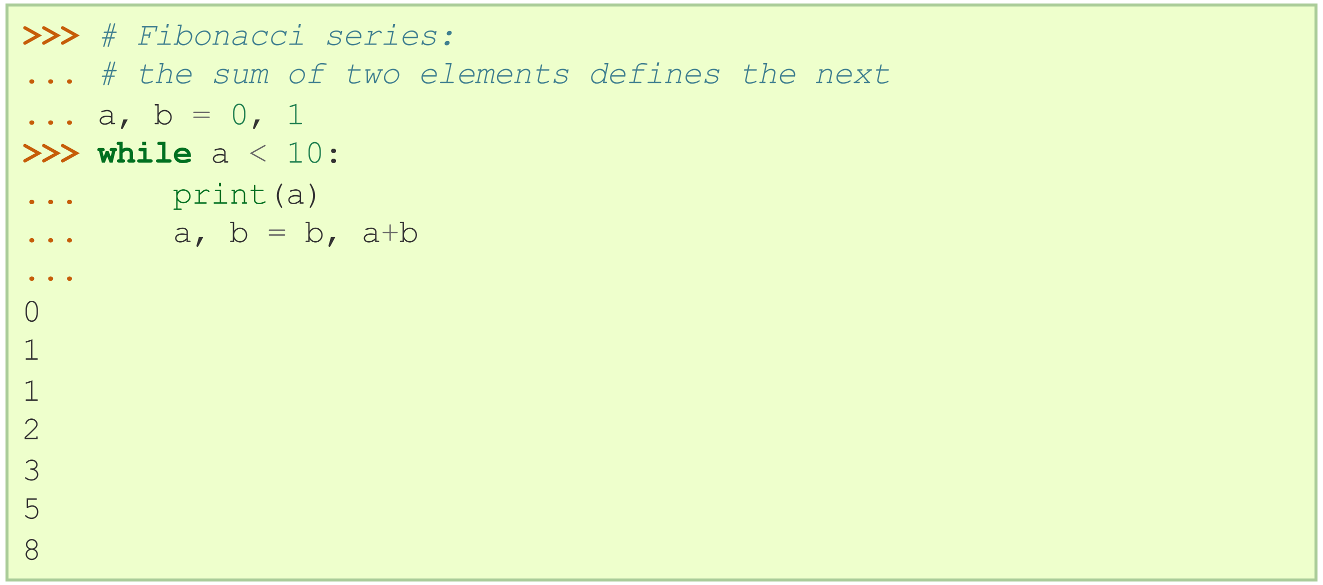
```
a, b = 0, 1
a, b = b, a+b
```

- While loop
  - Executes as long as the condition is true
  - Any non-zero integer is true, zero is false
  - Any non-zero length string is true
  - Operators: < <= > >= == !=
  - Body of the loop is indented

```
while a < 10:
```

- Print() function writes the value

```
print(a)
```



```
>>> # Fibonacci series:
... # the sum of two elements defines the next
... a, b = 0, 1
>>> while a < 10:
...     print(a)
...     a, b = b, a+b
...
0
1
1
2
3
5
8
```