Introducing Python

Computer & Information Sciences

W. H. Bell

Overview

- Python: motivation, features, support, versions.
- Basic types.
- Conditions and loops.
- Functions.

Python motivation

- Strong developer community.
- Wide range of supporting libraries.
- Simple syntax.
- Interfaces well with other languages, e.g. C.
- Flexibility - supports many programming paradigms.

Python features

- Interpreted language.
- Not compiled.
- Libraries may use compiled C code.
- Interactive or batch use.
- Many development tools available.
- Integrated development environments (IDEs).
- Linter – to check for syntax errors and suggest functions.
- Debugger.

Python support

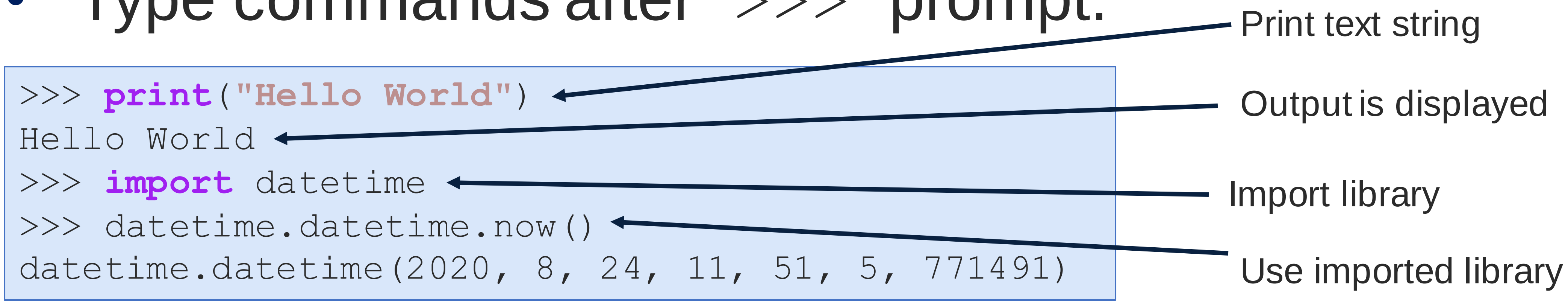
- Supported on several operating systems:
- Linux – distributed libraries, PyPI libraries, Anaconda.
- Mac – development tools, PyPI libraries, Anaconda.
- Windows – Visual Studio, PyPI libraries, Anaconda.
- Recommend that Anaconda is used on Windows.
- Anaconda includes a full suite of packages.
- Examples provided using VS Code IDE.

Python versions

- Python 2.7.x continues to be used on many systems.
- May find 2.7.x is installed on your PC.
- Python 3.x is being actively developed.
- This course assume Python 3.6.x or greater.
- Python 3 syntax is slightly different – print statements, string functions.

Interactive shell

- Type commands after ">>>" prompt.



```
>>> print("Hello World")
Hello World
>>> import datetime
>>> datetime.datetime.now()
datetime.datetime(2020, 8, 24, 11, 51, 5, 771491)
```

- Other interactive Python shells exist for data science applications.

Batch processing

- On Linux and Mac:

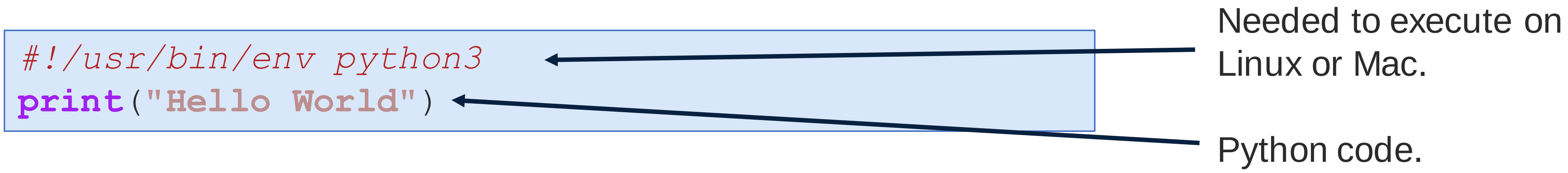
```
./myScript.py
```

- File must be executable and include interpreter reference in first line.
- On Linux, Mac and Windows:

```
python3 myScript.py
```

- "`python`" normally implies Python 2.x.

First script



```
#!/usr/bin/env python3
print("Hello World")
```

Needed to execute on Linux or Mac.

Python code.

Output

```
Hello World
```

Comment types

- Single line comments using #.
- Multiple line comments using """ and """.

```
"""
A script to demonstrate Python comments,
which might span several lines.
"""

# Another print statement
print("Comment examples") # This line prints a string
```

Variable assignment and typing

- Type is defined when variable is first assigned a value.

```
x = 10 # Integer
s = "A text string" # String
f = 3.14159 # Float
b = True # Boolean
l = [] # List
d = {} # Dictionary
```

- Can test the type using **`type`** or **`isinstance`** functions.
- Functionality is specific to type.
- Errors when the type is wrongly assumed.

Evaluation order

- Statements are not mathematical equations.
- Statements are evaluated in order.
- Results are then assigned.

```
import math
x = 10
x = x + 1
x += 1
f = 3.1415/2.0
f = math.cos(f)
```

Equivalent operations

Lists

- Sequential and dynamically allocated.

```
mass = []
mass = mass + [ 1.23 ]
mass += [ 2.34, 3.34 ]
print(mass)
```

Append an element.

Append two elements.

Output

```
[1.23, 2.34, 3.34]
```

- Access elements using index, e.g. `mass[2]`
- First element is the zeroth element.

Dictionaries

- Key and value, keys are stored in a hash, allocation is dynamic.

```
dataValues = {}
dataValues["Glasgow"] = 23.45
dataValues["Edinburgh"] = 13.23
print(dataValues)
print(dataValues["Glasgow"])
```

Adding key and value pair.

Adding key and value pair.

Output

```
{'Glasgow': 23.45, 'Edinburgh': 13.23}
23.45
```

Combining containers

- Can build up complex data structures.

```
dataTable = {}
dataTable["Element"] = [ "Al", "Fe" ]
dataTable["Mass"] = [ 2.3, 10.0 ]
print(dataTable)
```

Output

```
{'Element': ['Al', 'Fe'], 'Mass': [2.3, 10.0]}
```

Conditional statements

- Can combine logic requirements.

```
x = 9
if x < 10 and x > 0:
    print("0 < x < 10")
elif x == 15:
    print("x = 15")
else:
    print("(x <= 0 or x >= 10) and x != 15")
```

Output

```
0 < x < 10
```

Loops

- Iterate over values in range or list.

```
values = [ "A", "B", "C" ]
for value in values:
    print(value)

factorial = 1
for i in range(2,4):
    factorial *= i

print("3! = " + str(factorial))
```

Output

```
A
B
C
3! = 6
```

Functions

- Contain one or many instructions.
- Zero or many input arguments.
- Zero or one* return value.

```python
def myFunction(inputArgument):
    print(inputArgument)
    return True

print(myFunction(1.3))
```

Output

```
1.3
True
```

Main function

```
print("Hello") # Run on import and execution.

if __name__ == "__main__":
    print("Main") # Run on execution.
```

```
python3 main.py
```

Output

```
Hello
Main
```

```
>>> import main.py
```

Output

```
Hello
```

Recursive operations

- Recursion is useful when navigating graphs or hierarchies, as well as for some mathematical functions.

```
def factorial(x):
    if x == 0:
        return 1
    return x*factorial(x-1)

print(factorial(10))
```

Output

```
3628800
```

Mutability

- Mutable – assigned or passed by reference.
- Immutable – assigned or passed by value.

```
l = []
p = l
p += [ "A" ]
x = 10
y = x
y = 15
print(l)
print(x)
```

Output

```
['A']
10
```

Summary

- Introduced Python.
- Discussed basic types.
- Discussed conditions and loops.
- Discussed functions.
- More details are provided in the course notes.