Part 2.2:
Python: Loops, Lists, Files and Modules

Dana L Carper and Travis J Lawrence

Quantitative and Systems Biology

University of California, Merced

Python Development Environment: ATOM text editor

- Development environments provide tools to make writing code easier
  - Syntax highlighting
  - Automatic syntax enforcement
  - Intelligent autocomplete
  - Debugging
- ATOM text editor
  - Open source
  - Free
  - Easy to use and extend

Running Python Scripts

- Python is an interpreted language
  - Code is not compiled prior to executed
  - Code is read and executed by the Python interpreter
- Example:

```
python3 script_name.py argument1 argument2
```

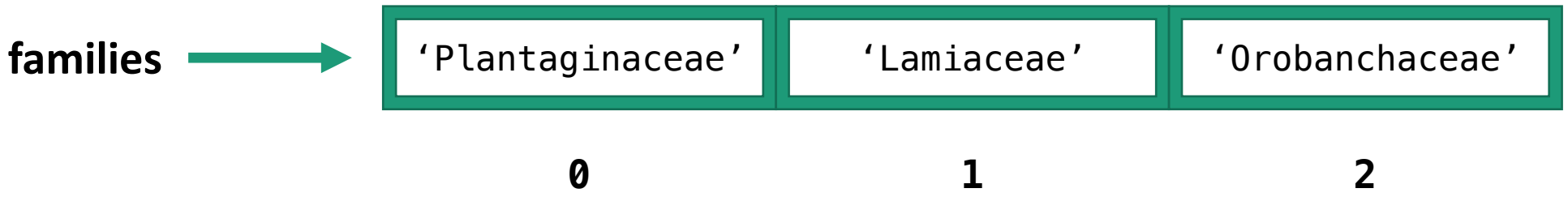
Running Python Scripts

- Python is an interpreted language
  - Code is not compiled prior to executed
  - Code is read and executed by the Python interpreter
- Example:

```
python3 script_name.py argument1 argument2
```

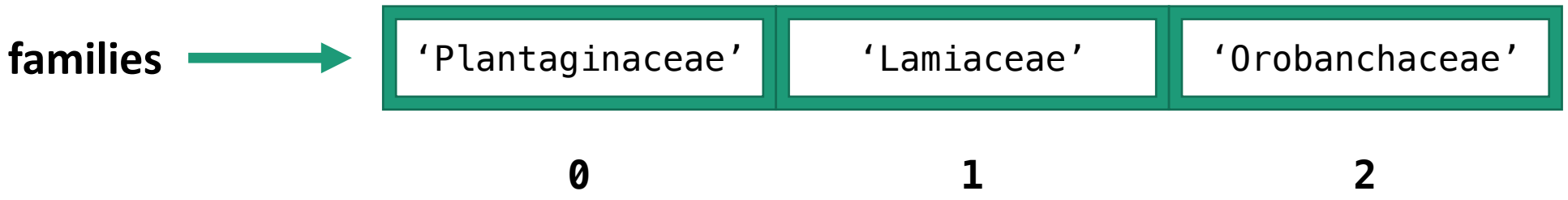
Data Structure: Lists

- Python lists are a sequence of Python datatypes (integers, floats, strings)



Data Structure: Lists

- Python lists are a sequence of Python datatypes (integers, floats, strings)



- List values are separated by commas and surrounded by square brackets
- Lists are zero indexed

```
families = ['Plantaginaceae', 'Lamiaceae', 'Orobanchaceae']
print(families[0])   # Plantaginaceae
print(families[2])   # Lamiaceae
print(len(families)) # 3
```

List Methods

- append
- extend
- index
- sort
- reverse
- remove
- pop

Files: Reading

- Python uses the open function to access files

```
open("filename.txt", "r") as file_handle
```

String indicating the file to open

- File mode
  - r – read
  - w – write

Variable containing the file object

- To read lines from the opened file use the `readline()` method on the file object

```
open("filename.txt", "r") as file_handle
line = file_handle.readline() #'First line\n'
line = file_handle.readline() #'Second line\n'
file_handle.close
```

Loops

- We often need to repeat a block of code several times
  - Reading lines from a file
  - Calculating values for each item of a list
  - Running simulations
- Programming languages have loop statements to introduce repetitions
- Python has two types of loops
  - for loops
  - while loops

Loops: for

- Python for loops are iterator based loops that step through sequences, such as, lists
- Syntax:

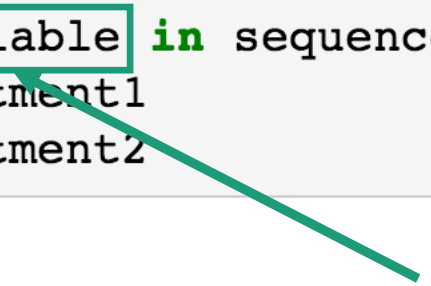
```
for variable in sequence:
    statment1
    statment2
```

for loop keyword

Loops: for

- Python for loops are iterator based loops that step through sequences, such as, lists
- Syntax:

```
for variable in sequence:
    statment1
    statment2
```



Variable that holds the current step of the sequence object

Loops: for

- Python for loops are iterator based loops that step through sequences, such as, lists
- Syntax:

```
for variable in sequence:
    statment1
    statment2
```

Loops: for

- Python for loops are iterator based loops that step through sequences, such as, lists
- Syntax:

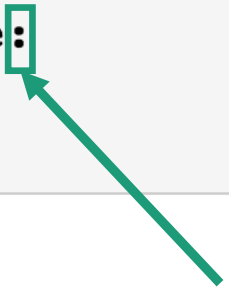
```
for variable in sequence:
    statment1
    statment2
```

Sequence object (e.g. list) to step through

Loops: for

- Python for loops are iterator based loops that step through sequences, such as, lists
- Syntax:

```
for variable in sequence:
    statment1
    statment2
```



The colon indicates the start of a indented code block

Loops: for

- Python for loops are iterator based loops that step through sequences, such as, lists
- Syntax:

```
for variable in sequence:
    statment1
    statment2
```

Indented code block that is executed each iteration of the loop

Loops: for

- Python for loops are iterator based loops that step through sequences, such as, lists
- Syntax:

```
for variable in sequence:
    statment1
    statment2
```

Indented code block that is executed each iteration of the loop

- Example: printing every line of a file:

```
open("filename.txt", "r") as file_handle
for line in file_handle:
    print(line)
```

Loops: While

- Syntax

```
while (condition is true):
    statement1
    statement2
```

- Commonly used for simulations

Modules

- Modules contain Python code that provides additional functionality
- Obtaining modules
  - Built-in
  - Installed from software repositories
- Using modules

```
import sys
```

Modules: sys

- Gives access to command-line arguments through sys.argv
  - sys.argv is a list of command-line arguments including the script name
- Syntax

```
python3 script_name.py argument1 argument2
```

```
import sys
print(sys.argv[0]) #script_name
print(sys.argv[1]) #argument1
print(sys.argv[2]) #argument2
```

- Arguments are commonly used to specify files