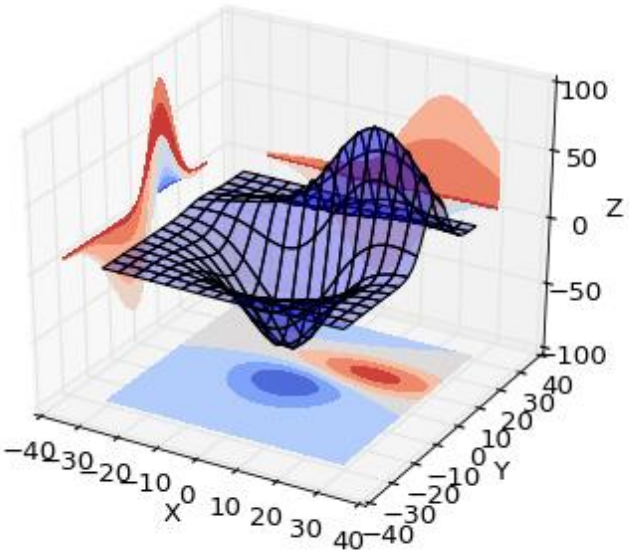
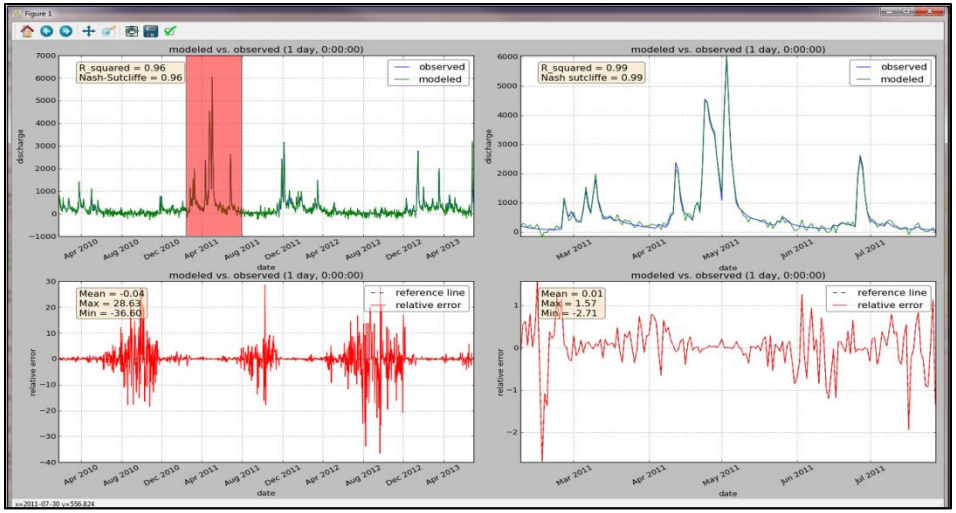
Introduction to Scientific Computing
Meeting 18
Programming with Python



python™

```
# Write 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
```



Jeremiah Lant, Hydrologist
USGS Kentucky Water Science Center
jlant@usgs.gov

Last Meeting

- Learned about built-in container called a **dictionary**

Last Meeting - dictionary

- **Dictionary** is a **container** or a **collection of items**.
  - Unordered collection of key/value pairs
  - Mapping/association between keys and values

dictionary = {*"key"*: *value, "key"*: *value***}**

```
>>> info = {"name": "Bob", "age": 30, "height": 6}
>>> info["name"]
Bob
>>> info["age"]
30
```

Last Meeting - dictionary

- Unlike a **list** where you must **use an integer index** to **access items**, in a **dictionary**, you **use a key** to **access items**.

```
>>> info_list = [“Bob”, 30,  6,]
>>> info_list[0]
Bob
>>> info_list[1]
30

>>> info_dict = {“name”: “Bob”, “age”: 30, “height”:  6}
>>> info_dict[“name”]
Bob
>>> info_dict[“age”]
30
```

Last Meeting - dictionary

- **Adding items** to a dictionary after it is created

```
>>> info_dict[“weight”] = 160
>>> info_dict[“hobbies”] = [“golf”, “tennis”]
```

- Test whether key is present using **in**

```
>>> “weight” in info_dict
True
>>> “birthday” in info_dict
False
```

Last Meeting - dictionary

- Dictionaries have **methods**
  - https://docs.python.org/2/library/stdtypes.html

```
>>> temperature = {“measurements”: [50, 60, 70],
                   “units”: “Fahrenheit”}
>>> temperature.keys()
[“units”, “measurements”]
>>> temperature.values()
[“Fahrenheit”}, [50, 60, 70]]
>>> temperature.get(“measurements”)
[50, 60, 70]
>>> temperature.items()
[(“units”, “Fahrenheit”), (“measurements”, [50, 60, 70])]
```

Last Meeting - dictionary

- **Using a for loop to print all keys and key/value pairs**

```
>>> temperature = {“measurements”: [30, 40, 50], “units”: “Fahrenheit”}

>>> for key, value in info_dict.items():
. . .     print(“Key {} maps to values {}”.format(key, value))
Key measurements maps to values [30, 40, 50]
Key units maps to values Fahrenheit
```

Review – page 1

- Explain how lists and dictionaries are the same

- Explain how lists and dictionaries are different

- When to use a list and when to use a dictionary?

Today’s Objectives

- Learn about **strings**

Demo- strings

- **Strings** are sequences of characters

string = “hello world”

string = ‘hello world’

```
>>> name = “Albert Einstein”
>>> type(name)
<type ‘str’>
```

Demo- strings

- **Strings** are sequences of characters

string = “hello world”

string = ‘hello world’

```
>>> name = “Albert Einstein”
>>> type(name)
<type ‘str’>
```

Video – Python Basics

software carpentry

- Software Carpentry, Greg Wilson
  - Python: Strings

  http://software-carpentry.org/v4/python/strings.html

Demo- strings

- **Strings** can be **indexed** just like **lists**

```
>>> name = “Albert Einstein”
>>> name[0]
‘A’
>>> name[-1]
‘n’
>>> name[:6]
‘Albert’
>>> name[7:]
‘Einstein’
```

Demo- strings

- **For loops with strings**

```
>>> name = “Albert Einstein”
>>> for char in name:
. . .     print(char)
```

- Can test for **string equality**

```
>>> name[:6] == “Albert”
True
>>> name[:6] == “albert”
False
```

Demo- strings

- **Strings** are **immutable** just like **lists** (cannot be changed in place)

```
>>> name = “Albert Einstein”
>>> name[0] = “B”
TypeError: ‘str’ object does not support item assignment
```

- **Concatenate strings** using **+**

```
>>> adjective = “great”
>>> noun = “scientist”
>>> sentence = name + “ “ + “is a” + “ ” + adjective + “ “ + noun
```

- **Duplicate strings** using ***

```
>>> “hello” * 3
```

- Can test for membership with **in**

```
>>> “E” in name
True
```

Demo- strings

- **Strings** have **methods**

https://docs.python.org/2/library/stdtypes.html#sequence-types-str-unicode-list-tuple-bytearray-buffer-xrange

```
>>> first_name = “albert”
>>> first_name.capitalize()

>>> name = “Albert Einstein”
>>> name.find(“E”)
>>> name.isdigit()
>>> name.upper()
>>> name.lower()
```

Demo- strings

- **Strings** have **methods**

```
>>> name = “     Albert          Einstein     ”
>>> name = name.strip()
‘Albert Einstein’
>>> name.strip(“Albert”)
‘ Einstein’

>>> name.split()
[‘Albert’, ‘Einstein’]
>>> full_file_name = “test-file.txt”
>>> full_file_name.split(“.”)
>>> file_name = full_file_name.split(“.”)[0]
>>> extension = full_file_name.split(“.”)[-1]
```

Demo- strings

- **Strings** have **methods**
- Example: better way to do the following using a string method called **join**

```
>>> adjective = “great”
>>> noun = “scientist”
>>> sentence = name + “ “ + “is a” + “ ” + adjective + “ “ + noun
>>> print(sentence)
Albert Einstein is a great scientist

>>>  sentence = “ “.join([name, “is a”, adjective, noun])
>>> print(sentence)
Albert Einstein is a great scientist

>>>  sentence = “-“.join([name, “is a”, adjective, noun])
```

Demo- strings

- **Strings** can have **escape characters:**
- **newline \n**

```
>>> name = “Albert\nEinstein”
>>> print(name)
Albert
Einstein
```

- **tab \t**

```
>>> name = “Albert\tEinstein”
>>> print(name)
Albert    Einstein
```

Demo- strings

- **Strings** can have **escape characters:**
- **single quote \’**

>>> print(“Don\’t you think that is a good idea.”)

Don’t you think that is a good idea

- **Double quote \”**

>>> print(“Do not say \”I can not do it\”, because you can.”

Do not say “I can not do it”, because you can.

Demo- strings

- **Formatting strings**
- **Basic formatting**

```
>>> print("{} {} {}".format("a", "b", "c")
a b c
```

- **Numbered fields refer to position of arguments**

```
>>> print("{2} {1} {0}".format("a", "b", "c")
c b a
```

- **Named fields formatting refer to keyword arguments**

```
>>> print("{x} {y} {name}".format(y = 5.5, x = 2, name = "Bob")
2 5.5 Bob
```

Demo- strings

- **Formatting strings**
- **Positional and keyword arguments combined**

```
>>> print("{x} {0} {y} {1} {name}".format(9, "hello", y = 5.5, x = 2, name = "Bob")
2 9 5.5 hello Bob
```

- **Precision and padding**

```
>>> print({0:10} {1:10} {name:20} {x}).format("hello", "world", name = "Bob", x = 8)
```

```
>>> print("{0:10.2f} times {1:d} is {result:10.2f}".format(2.5, 3, result = 2.5*3)
```

Next meeting

- Python – Learn about
  - **Little more on strings**
  - **Input and output**