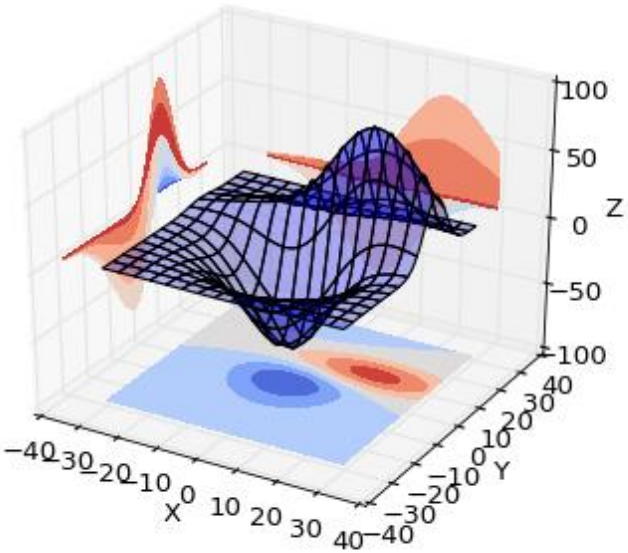
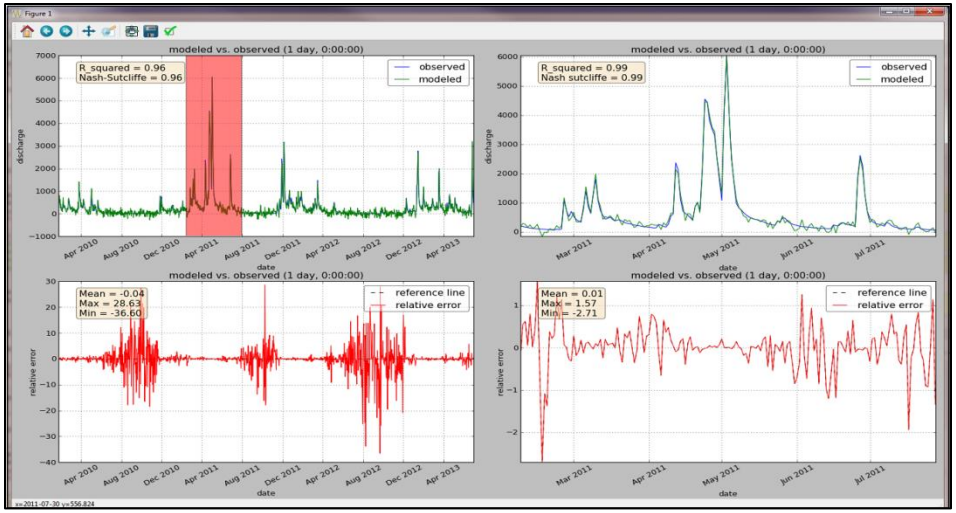
Introduction to Scientific Computing
Meeting 20
Programming with Python



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 **strings**

Demo- strings

- **Strings** are sequences of characters

string = “hello world”

string = ‘hello world’

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

Demo- strings

- **Strings** can be **indexed** just like **lists**
- **For loops with strings**
- Can test for **string equality**
- **Strings** are **immutable** (cannot be changed in place)
- **Concatenate strings** using **+**
- **Duplicate strings** using *****
- Can test for membership with **in**
- **Strings** have **methods; .split(), .strip(), .join(), and many more**

Demo- strings

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

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

- **tab \t**

```
>>> name = “\tAlbert\tEinstein”
>>> print(name)
Albert    Einstein
>>> name.strip(“\t”)
```

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
```

- **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
```

Demo- strings

- **Formatting strings**
  - **{<field name>: <format specification> }**
  - **{x:5.2f}**

ENTHOUGHT

Format Specification

```
>>> 'price: ${0:=-7.2f}'.format(3.4)
'price: $   3.40'
```

The format spec is a sequence of characters including:

- the alignment option,
- the sign option,
- the width (and .precision) option
- the type code.

ALIGNMENT OPTION

Char	Meaning
<	Left aligned.
>	Right aligned.
=	(For numeric types only.) Pad after the sign but before the digits (e.g. +000000120).
^	Center within the available space.

If an alignment character is given, it may be preceded by a fill character.

SIGN OPTION

For numbers only.

Char	Meaning
+	Include a sign for positive and negative number.
-	Indicate sign for negative numbers only (default)
space	Include a leading space for positive numbers.

STRING TYPE CODES

Type	Meaning
s	String. This is the default, and may be omitted.

INTEGER TYPE CODES

Type	Meaning
b	Binary format.
c	Character; converts int to unicode char.
d	Decimal integer (base 10).
o	Octal (base 8).
x	Hex (base 16), lower case.
X	Hex (base 16), upper case.
n	Number; same as 'd', but uses current locale.
None	Same as 'd'.

FLOATING POINT TYPE CODES

Type	Meaning
e	Scientific notation.
E	Scientific notation, with upper case 'E'.
f	Fixed point.
F	Fixed point; same as 'f'.
g	General format.
G	General format; same as 'g', with upper case 'E' when necessary.
n	Number; same as 'g', but uses current locale.
%	Percentage. Multiplies by 100 and displays with 'f', followed by a percent sign.
None	Same as 'g'.

Demo- strings

optional

- **Formatting strings**
  - **{<field name>: <format specification> }**
  - **{ x : 5.2f }**

Format Specification

ENTHOUGHT

```
>>> 'price: ${0:=-7.2f}'.format(3.4)
'price: $   3.40'
```

The format spec is a sequence of characters including:

- the alignment option,
- the sign option,
- the width (and .precision) option
- the type code.

ALIGNMENT OPTION

Char	Meaning
<	Left aligned.
>	Right aligned.
=	(For numeric types only.) Pad after the sign but before the digits (e.g. +000000120).
^	Center within the available space.

If an alignment character is given, it may be preceded by a fill character.

SIGN OPTION

For numbers only.

Char	Meaning
+	Include a sign for positive and negative number.
-	Indicate sign for negative numbers only (default)
space	Include a leading space for positive numbers.

STRING TYPE CODES

Type	Meaning
s	String. This is the default, and may be omitted.

INTEGER TYPE CODES

Type	Meaning
b	Binary format.
c	Character; converts int to unicode char.
d	Decimal integer (base 10).
o	Octal (base 8).
x	Hex (base 16), lower case.
X	Hex (base 16), upper case.
n	Number; same as 'd', but uses current locale.
None	Same as 'd'.

FLOATING POINT TYPE CODES

Type	Meaning
e	Scientific notation.
E	Scientific notation, with upper case 'E'.
f	Fixed point.
F	Fixed point; same as 'f'.
g	General format.
G	General format; same as 'g', with upper case 'E' when necessary.
n	Number; same as 'g', but uses current locale.
%	Percentage. Multiplies by 100 and displays with 'f', followed by a percent sign.
None	Same as 'g'.

Demo- strings

- **Formatting strings**
  - **{<field name>: <format specification> }**
  - **{x:5.2f}**

- **Precision and padding**

```
>>> print("[{x:5.0f}]   [("[{x:5.1f}] [("[{x:5.2f}]".format(x = 3.14)
[    3]   [    3.1]   [    3.14]

>>> 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)
```

Demo- strings

- **Formatting strings**
  - **{<field name>: <format specification> }**

- **Alignment**

```
>>> print("[{0:<10s}]".format("Albert")
[Albert    ]
>>> print("[{0:>10s}]".format("Albert")
[    Albert]
>>> print("[{0:^10s}]".format("Albert")
[  Albert  ]
```

Demo- strings

- **Formatting strings**
  - **{<field name>: <format specification> }**

- **Alignment will fill character**

```
>>> print("[{0:10s}]".format("Albert")
[Albert    ]
>>> print("[{0:*<10s}]".format("Albert")
[Albert****]
>>> print("[{0:*>10s}]".format("Albert")
[****Albert]
>>> print("[{0:*^10s}]".format("Albert")
[**Albert**]
```

Practice Objectives - strings

- Fill in answers to einstein_quotes.py

Today’s Objectives

- Learn about **input from user**

Demo – input from user

- You can receive input from a user to your programs using the **raw_input** function

```
answer = raw_input("some message")
```

Demo – input from user

- You can receive input from a user to your programs using the **raw_input** function

```
answer = raw_input("some message")
```

raw_input returns input from user as a **string**

You provide some **string** message

Demo – input from user

- Let’s build a little program called input_statements.py that will request for user information, print the result, and organize all the information into a collection/container of your choice.

Next meeting

- Input from user via **sys** module
- **File Input and output**