CUCaTS Python Workshop

Lent 2015

Literacy is a bridge from misery to hope. It is a tool for daily life in modern society. It is a bulwark against poverty, and a building block of development, an essential complement to investments in roads, dams, clinics and factories. Literacy is a platform for democratization, and a vehicle for the promotion of cultural and national identity. Especially for girls and women, it is an agent of family health and nutrition. For everyone, everywhere, literacy is, along with education in general, a basic human right. . . Literacy is, finally, the road to human progress and the means through which every man, woman and child can realize his or her full potential.

– Kofi Annan

Python

- The command prompt allows you to navigate through folders and run programs. Open it and start `python`.

```
Python 3.4.2 (default, Dec 27 2014, 13:16:08)
[GCC 4.9.2] on linux
Type "help", "copyright", "credits" or "licen
>>>
```

- This is the interpreter.
- Type lines of Python code into it to run them.
- Changes here aren't permanent. Can close and start again.

Values

- A program processes data.
- Data such as numbers, text, images, dates/time, sound, files and so on.
- Even complex and abstract things like a road network or the structure of a protein.
- A value is a piece of data stored in a computer's memory.

Datatypes

All values have a datatype. Some examples:

- Numbers (integers, floats)
  e.g. `5` or `9.6`

Datatypes

All values have a datatype. Some examples:

- Numbers (integers, floats)
  e.g. `5` or `9.6`
- Strings
  `"Hello world!"`

Datatypes

All values have a datatype. Some examples:

- Numbers (integers, floats)
  e.g. `5` or `9.6`
- Strings
  `"Hello world!"`
- Booleans
  `True` and `False`

Datatypes

All values have a datatype. Some examples:

- Numbers (integers, floats)
  e.g. `5` or `9.6`
- Strings
  `"Hello world!"`
- Booleans
  `True` and `False`
- Lists
  `[4, 8, 15, 16, 23, 42]`
  or `["first", "second", "fourth"]`
  or even `[[], "some text", 99]`

Operations

Arithmetic:

```
>>> 5 + 9
14
```

Operations

Arithmetic:

```
>>> 5 + 9
14
>>> 7.5 * 2
15.0
```

Operations

Arithmetic:

```
>>> 5 + 9
14
>>> 7.5 * 2
15.0
```

Comparisons:

```
>>> 5 > 6
False
>>> 9 == 18 / 2
True
```

Operations

Arithmetic:

```
>>> 5 + 9
14
>>> 7.5 * 2
15.0
```

Comparisons:

```
>>> 5 > 6
False
>>> 9 == 18 / 2
True
```

String concatenation (joining):

```
>>> "beans" + "talk"
"beanstalk"
```

Try it out

```
>>> "a" + "b"
>>> "test" + "ing" == "testing"
>>> 3 == "3"
>>> "a" < "b"
>>> "Na"*8 + " batman"
```

Variables

- Variables are labels put onto values. They allow us to refer to those values by a name.

```
>>> x = 5
>>> y = x+6
>>> y
11
```

Variables

- Variables are labels put onto values. They allow us to refer to those values by a name.

```
>>> x = 5
>>> y = x+6
>>> y
11
```

- They can be reassigned:

```
>>> x = 5
>>> x = 7
>>> x
7
```

Accessing elements in lists

```
>>> days = ["Mon", "Tue", "Wed", "Thu",
...         "Fri", "Sat", "Sun"]
>>> days[1]
```

Accessing elements in lists

```
>>> days = ["Mon", "Tue", "Wed", "Thu",
...         "Fri", "Sat", "Sun"]
>>> days[1]
```

Let's make a list. To get items from the list, we use square brackets. What happens if we ask for the first element?

Accessing elements in lists

```
>>> days = ["Mon", "Tue", "Wed", "Thu",
...         "Fri", "Sat", "Sun"]
>>> days[1]
'Tue'
```

Accessing elements in lists

```
>>> days = ["Mon", "Tue", "Wed", "Thu",
...         "Fri", "Sat", "Sun"]
>>> days[1]
'Tue'
```

Tuesday? Why not Monday?

Accessing elements in lists

```
>>> days = ["Mon", "Tue", "Wed", "Thu",
...          "Fri", "Sat", "Sun"]
>>> days[1]
'Tue'
```

- First element is index `0`.
- Think of the index as "how many items into the list to move".

```
>>> days[0] == "Mon"
True
```

Accessing elements in lists

```
>>> days = ["Mon", "Tue", "Wed", "Thu",
...         "Fri", "Sat", "Sun"]
>>> days[1]
'Tue'
```

- First element is index 0.
- Think of the index as "how many items into the list to move".

```
>>> days[0] == "Mon"
True
```

It's because indices start with zero in Python.

Accessing elements in lists

```
>>> days = ["Mon", "Tue", "Wed", "Thu",
...          "Fri", "Sat", "Sun"]
>>> days[1]
'Tue'
```

- First element is index `0`.
- Think of the index as "how many items into the list to move".

```
>>> days[0] == "Mon"
True
```

- Can also access in reverse.

```
>>> days[-1]
'Sun'
```

Accessing elements in lists

```
>>> days = ["Mon", "Tue", "Wed", "Thu",
...         "Fri", "Sat", "Sun"]
>>> days[1]
'Tue'
```

- First element is index 0.
- Think of the index as "how many items into the list to move".

```
>>> days[0] == "Mon"
True
```

- Can also access in reverse.

```
>>> days[-1]
'Sun'
```

We can also index into lists with negative numbers, to work backwards from the end of the list.

Accessing elements in lists

```
>>> days = ["Mon", "Tue", "Wed", "Thu",
...          "Fri", "Sat", "Sun"]
>>> days[1]
'Tue'
```

- First element is index `0`.
- Think of the index as "how many items into the list to move".

```
>>> days[0] == "Mon"
True
```

- Can also access in reverse.

```
>>> days[-1]
'Sun'
```

"Mon"	"Tue"	"Wed"	"Thu"	"Fri"	"Sat"	"Sun"
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Errors

- There are rules governing what you can't do with certain types and values.

Errors

- There are rules governing what you can't do with certain types and values. For example:
  - Adding an integer to a string.

```
>>> 16 + "hello"
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Errors

- There are rules governing what you can't do with certain types and values. For example:
  - Adding an integer to a string.

```
>>> 16 + "hello"
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

  - Indexing into something that is not a sequence.

```
>>> 9.0[1]
TypeError: 'float' object has no attribute '__getitem__'
```

Errors

- There are rules governing what you can't do with certain types and values. For example:
  - Adding an integer to a string.

```
>>> 16 + "hello"
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

  - Indexing into something that is not a sequence.

```
>>> 9.0[1]
TypeError: 'float' object has no attribute '__getitem__'
```

  - Asking for the sixth element of a list that only has four.

```
>>> [4, 5, 6, 7][5]
IndexError: list index out of range
```

Errors

- There are rules governing what you can't do with certain types and values. For example:
  - Adding an integer to a string.

```
>>> 16 + "hello"
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

  - Indexing into something that is not a sequence.

```
>>> 9.0[1]
TypeError: 'float' object has no attribute '__getitem__'
```

  - Asking for the sixth element of a list that only has four.

```
>>> [4, 5, 6, 7][5]
IndexError: list index out of range
```

- Designed to help you spots problems with your code.

Try it out

```
>>> "this is a string" + 2
>>> "this is a string" - "string"
>>> 5/0

>>> list = [1, 2, 3]
>>> list[1.0]
```

The print function

```
print(value)
```

- `print` writes things onto the screen.

The print function

```
print(value)
```

- `print` writes things onto the screen.
- `print` can print all kinds of values.

The print function

```
print(value)
```

- `print` writes things onto the screen.
- `print` can print all kinds of values.
- A useful function to see "what's going on" in your program.

Making a program: writing Python code in a file

We have been using the interpreter. There is another way to use Python.

- We can save some Python commands in a file and then invoke the interpreter on the file.

Making a program: writing Python code in a file

We have been using the interpreter. There is another way to use Python.

- We can save some Python commands in a file and then invoke the interpreter on the file.
- Write `print("Hello!")` in a file using your text editor and save it somewhere under the name `my_python_script.py`.
- Note the `.py` extension!

Making a program: writing Python code in a file

We have been using the interpreter. There is another way to use Python.

- We can save some Python commands in a file and then invoke the interpreter on the file.
- Write `print("Hello!")` in a file using your text editor and save it somewhere under the name `my_python_script.py`.
- Note the `.py` extension!
- Open a command prompt at the folder where you saved the script and type in the name of your script (`my_python_script.py`).
- The computer should respond: `Hello!`

Control flow

The order that things happen in.

Control flow

The order that things happen in.
(Very) broadly:

- Loops
- Branches
- Function calls

‘for’ loops

Run a block of code for every element in a list. Indentation specifies the block.

```
>>> numbers = [4, 8, 15, 16, 23, 42]
>>> total = 0
>>> for num in numbers:
...     total = total + num
```

‘for’ loops

Run a block of code for every element in a list. Indentation specifies the block.

```
>>> numbers = [4, 8, 15, 16, 23, 42]
>>> total = 0
>>> for num in numbers:
...     total = total + num

>>> print(total)
108
```

‘for’ loops

Run a block of code for every element in a list. Indentation specifies the block.

```
>>> numbers = [4, 8, 15, 16, 23, 42]
>>> total = 0
>>> for num in numbers:
...     total = total + num

>>> print(total)
108
```

num	total
	0

‘for’ loops

Run a block of code for every element in a list. Indentation specifies the block.

```
>>> numbers = [4, 8, 15, 16, 23, 42]
>>> total = 0
>>> for num in numbers:
...     total = total + num

>>> print(total)
108
```

num	total
	0
4	4

‘for’ loops

Run a block of code for every element in a list. Indentation specifies the block.

```
>>> numbers = [4, 8, 15, 16, 23, 42]
>>> total = 0
>>> for num in numbers:
...     total = total + num

>>> print(total)
108
```

num	total
	0
4	4
8	12

‘for’ loops

Run a block of code for every element in a list. Indentation specifies the block.

```
>>> numbers = [4, 8, 15, 16, 23, 42]
>>> total = 0
>>> for num in numbers:
...     total = total + num

>>> print(total)
108
```

num	total
	0
4	4
8	12
15	27
16	43
23	66
42	108

Processing lists

```
>>> nums = [3, 4, 5, 10, 15]
>>> squares = [9, 16, 25, 100, 225]
```

Processing lists

```
>>> nums = [3, 4, 5, 10, 15]
>>> squares = [x*x for x in nums]
```

Processing lists

```
>>> nums = [3, 4, 5, 10, 15]
>>> squares = [x*x for x in nums]
>>> print(squares)
[9, 16, 25, 100, 225]
```

Processing lists

```
>>> nums = [3, 4, 5, 10, 15]
>>> squares = [x*x for x in nums]
>>> print(squares)
[9, 16, 25, 100, 225]
```

- Creates a new list out of another by using each element to create a corresponding new value.
- 'for' and 'in' are keywords.
- This is called a list comprehension.

if-else

Provide two different pieces of code to run depending on some condition.

if-else

Provide two different pieces of code to run depending on some condition.
A condition is an expression that evaluates to a boolean, a truth value – either `True` or `False`.

if-else

Provide two different pieces of code to run depending on some condition.
A condition is an expression that evaluates to a boolean, a truth value – either `True` or `False`.

```
if ____:
    __ #Do something if the condition is true
else:
    __ #Something else if it's false
```

if-else examples

Let's write some code to go through a bunch of values in a list and output whether or not they are even or odd.

if-else examples

Let's write some code to go through a bunch of values in a list and output whether or not they are even or odd.

```
numbers = [6, 7, 8, 9, 10, 5, 4, 3, 2, 1]

for x in numbers:
    if (x % 2) == 0:
        print(x, "is even.")
    else:
        print(x, "is odd.")
```

if-else examples

Let's write some code to go through a bunch of values in a list and output whether or not they are even or odd.

```
numbers = [6, 7, 8, 9, 10, 5, 4, 3, 2, 1]

for x in numbers:
    if (x % 2) == 0:
        print(x, "is even.")
    else:
        print(x, "is odd.")
```

Try it out!

Functions

A function is a reusable piece of code. It takes an input value (parameter), processes it and gives back an output (return value).

Functions

A function is a reusable piece of code. It takes an input value (parameter), processes it and gives back an output (return value).

- Definition:

```
def sum(numbers):
    total = 0
    for num in numbers:
        total = total + num
    return total
```

Functions

A function is a reusable piece of code. It takes an input value (parameter), processes it and gives back an output (return value).

- Definition:

```
def sum(numbers):
    total = 0
    for num in numbers:
        total = total + num
    return total
```

- Use:

```
y = sum([5, 6, 10, 9])
#The value of y is now 30.
```

The value of the function call is specified by `return`, which ends the function.

Functions

Functions can also take multiple parameters:

```
def find(list, value):
    for item in list:
        if item == value:
            return True

    return False
```

```
print(find(["Bob", 16, 0.5, 4], 5))
#"False"
print(find(["Bob", 16, 0.5, 4], 0.5))
#"True"
```

Functions

Functions can also take multiple parameters:

```
def find(list, value):
    for item in list:
        if item == value:
            return True

    return False
```

```
print(find(["Bob", 16, 0.5, 4], 5))
#"False"
print(find(["Bob", 16, 0.5, 4], 0.5))
#"True"
```

Try changing it to return the index, if found.