CUCaTS Python Workshop

Lent 2015

Environment setup

Python

Python is a language: a set of rules for constructing programs. You can experiment with these rules using “IDLE”.

- When you open IDLE, you can type things in and press enter to have IDLE read what you wrote.
- When you type things in, you should write in Python. IDLE will complain if you don’t.
- Things you do in IDLE aren’t permanent. You can always close IDLE and open it afresh.
- As we introduce new ideas today, you can follow along by typing things into IDLE.

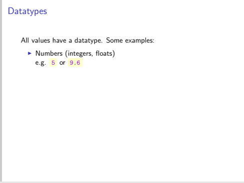
Values

- 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:

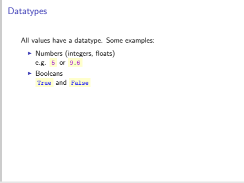
- Numbers (integers, floats)
  e.g. `5` or `9.6`

Integers can hold whole numbers only. "Floats", or floating-point numbers, are capable of fractional numbers as well.

Datatypes

All values have a datatype. Some examples:

- Numbers (integers, floats)
  e.g. `5` or `9.6`
- Booleans
  `True` and `False`



Datatypes

All values have a datatype. Some examples:

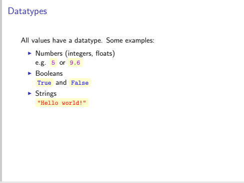
- Numbers (integers, floats)
  e.g. `5` or `9.6`
- Booleans
  `True` and `False`

True/false values are called “Booleans” (after George Boole). They are used for expression conditions.

Datatypes

All values have a datatype. Some examples:

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



Datatypes

All values have a datatype. Some examples:

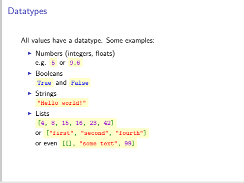
- Numbers (integers, floats)
  e.g. `5` or `9.6`
- Booleans
  `True` and `False`
- Strings
  `"Hello world!"`

Strings can use single quotes or double quotes, but we will stick to double quotes for consistency.

Datatypes

All values have a datatype. Some examples:

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



Datatypes

All values have a datatype. Some examples:

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

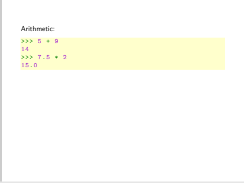
Lists hold a sequence of values. The third example is a list containing firstly an empty list, then a string and then an integer. As you can see, the elements of a list do not have to all be the same datatype. In Python, a list is simply a list, not a list *of* something.

Arithmetic:

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

Arithmetic:

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



Arithmetic:

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

Note that the multiplication results in `15.0`, a float, because `7.5` was a float.

Arithmetic:

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

Comparisons:

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

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.
  - 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.
  - Indexing into something that is not a sequence.
  - Asking for the sixth element of a list that only has four.

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

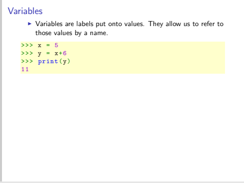
Errors

- There are rules governing what you can't do with certain types and values. For example:
  - Adding an integer to a string.
  - Indexing into something that is not a sequence.
  - Asking for the sixth element of a list that only has four.
- Designed to help you spots problems with your code.

Variables

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

```
>>> x = 5
>>> y = x+6
>>> print(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
>>> print(y)
11
```

Variable names must consist only of letters, digits and underscores. They can't start with a digit or have spaces. They are case sensitive,

Variables

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

```
>>> x = 5
>>> y = x+6
>>> print(y)
11
```

- They can be reassigned:

```
>>> x = 5
>>> x = 7
>>> print(x)
7
```

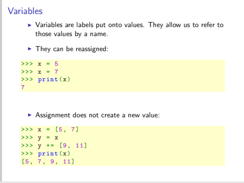
Variables

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

```
>>> x = 5
>>> x = 7
>>> print(x)
7
```

- Assignment does not create a new value:

```
>>> x = [5, 7]
>>> y = x
>>> y += [9, 11]
>>> print(x)
[5, 7, 9, 11]
```



Variables

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

```
>>> x = 5
>>> x = 7
>>> print(x)
7
```

- Assignment does not create a new value:

```
>>> x = [5, 7]
>>> y = x
>>> y += [9, 11]
>>> print(x)
[5, 7, 9, 11]
```

`x` and `y` now label the same value. When we add to `y`, it is the same list that gets modified.

Processing lists

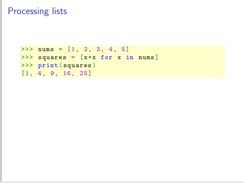
```
>>> nums = [1, 2, 3, 4, 5]
>>> squares = [1, 4, 9, 16, 25]
```

Processing lists

```
>>> nums = [1, 2, 3, 4, 5]
>>> squares = [x*x for x in nums]
```

Processing lists

```
>>> nums = [1, 2, 3, 4, 5]
>>> squares = [x*x for x in nums]
>>> print(squares)
[1, 4, 9, 16, 25]
```



Processing lists

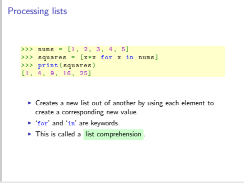
```
>>> nums = [1, 2, 3, 4, 5]
>>> squares = [x*x for x in nums]
>>> print(squares)
[1, 4, 9, 16, 25]
```

Keywords are ones that have a special meaning. You cannot use them as variable names. List comprehensions must use ‘`for`’ and ‘`in`’

Processing lists

```
>>> nums = [1, 2, 3, 4, 5]
>>> squares = [x*x for x in nums]
>>> print(squares)
[1, 4, 9, 16, 25]
```

- 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.



Processing lists

```
>>> nums = [1, 2, 3, 4, 5]
>>> squares = [x*x for x in nums]
>>> print(squares)
[1, 4, 9, 16, 25]
```

- 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.

Keywords are ones that have a special meaning. You cannot use them as variable names. List comprehensions must use '`for`' and '`in`'

for loops

```
>>> nums = [1, 2, 3, 4, 5]
>>> total = 0

>>> for x in nums:
        total = total + x
```

for loops

```
>>> nums = [1, 2, 3, 4, 5]
>>> total = 0

>>> for x in nums:
        total = total + x

>>> print(total)
15
```

if-else

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

```
if  :
```

Functions

Methods