COMP10001 Foundations of Computing
Characters, Strings and Lists

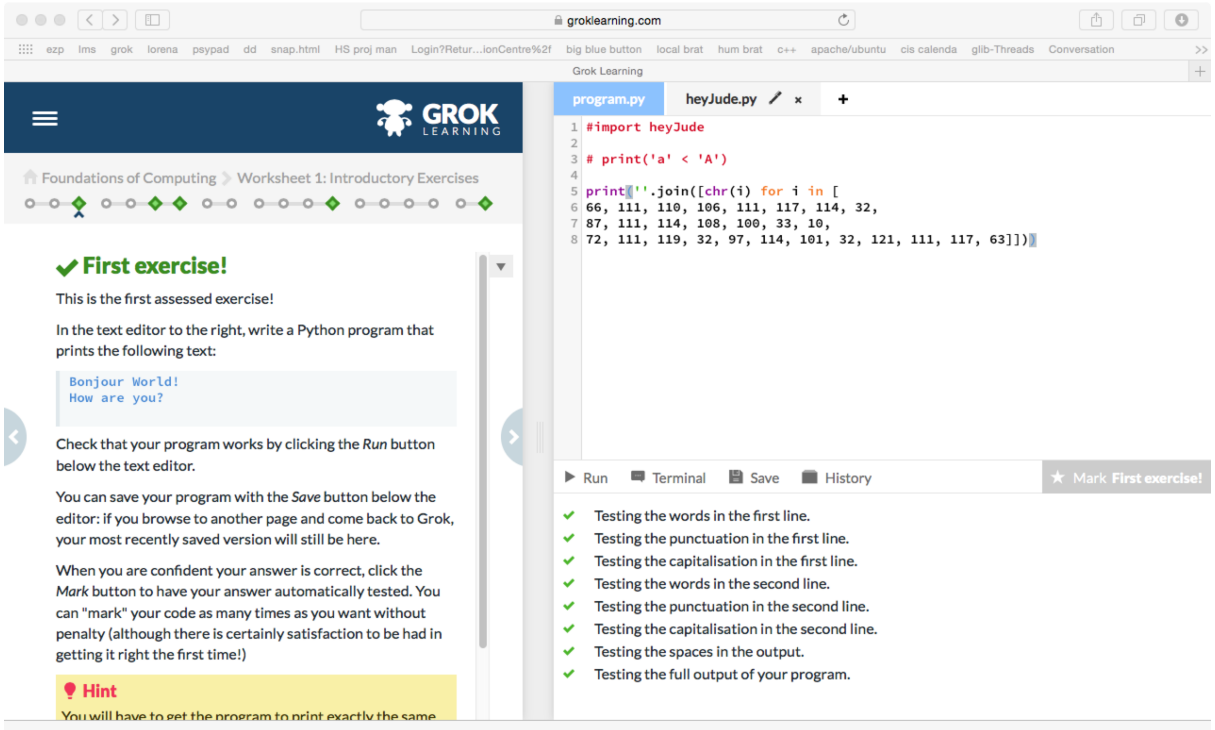
Semester 2, 2016

Chris Leckie

July 31, 2016

Reminders

- Complete Worksheets 1 and 2 by 23:59 Monday night 8/8.
- Online Help Tue-Thursday evenings via LMS "Online Help" chat box.



Lecture Agenda

- Last lecture:
  - Types
  - Strings
  - Literals, variables and assignment
- This lecture:
  - Type conversion
  - Printing
  - Comments
  - Character representation
  - Strings

Type Conversion

- Python implicitly determines the type of each literal and variable, based on its syntax (literals) or the type of the assigned value (variables)
- To "cast" a literal/variable to a different type, we use functions of the same name as the type: `int()`, `float()`, `str()`, `complex()`

```
>>> float(1)
1.0
>>> int(1.0)
1
>>> int(1.5)
1
>>> int('a')
Traceback (most recent call last):
  File "<web session>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'a'
```

A Couple of Other Useful Functions

- `abs()`: return the absolute value of the operand
- `len()`: return the length of the iterable operand (i.e. a `str` for now)

```
>>> len('apple')
5
>>> len(1)
Traceback (most recent call last):
  File "<web session>", line 1, in <module>
TypeError: object of type 'int' has no len()
```

Class Exercise (1)

- Given `num` containing an `int`, calculate the number of digits in it

The print() Function

- The `print()` function can be used to print the value of the operand (of any type)

```
>>> a = 1
>>> print(a)
1
```

- In the console, there is no noticeable difference between printing and executing a variable:

```
>>> a = 1
>>> print(a)
1
>>> a
1
```

but when you "run" code from a file, you will only see the output of `print()` functions

The print Statement

- In Python 2, you can use either the `print` statement (`print ...`) or the `print` function `print(...)`, but Python 3 only allows the `print` function

```
>>> a = 1
>>> print(a)
1
>>> print a    # Python 2
1
```

so if you use Python 2 code from the www, remember to convert print statements to print functions.

Comments

- Comments are notes of explanation that document lines or sections of a program, which follow a # (hash) character
- Python ignores anything following a # on a single line (multi-line commenting possible with """):

```
# OK, here goes
"""Three blind mice,
Three blind mice,
..."""
print("Hello world")
```

Commenting Expectations

- For this subject we require:
  - A set of comments at the beginning of every python program:

```
# What does this program do
# Author(s): Who wrote me
# Date created
# Date modified and reason
```

  - All key variables should have comments about what they are used for (as should user-defined functions)
  - Commenting can also be used to stop lines of code from being executed. This is called "commenting out" code.

More on String Manipulation

- As well as "assembling" strings via + and $*$, we are able to pull strings apart in the following ways:
  - "indexing" — return the single character at a particular location
  - "slicing" — extract a substring of arbitrary length
  - "splitting" — break up a string into components based on particular substrings

String Manipulation: Indexing

- Each character in a string can be accessed via "indexing" relative to its position from the left of the string (zero-offset) or the right of the string ([minus] one-offset):

I	t		w	a	s		a		d	a	r	k
0	1	2	3	4	5	6	7	8	9	10	11	12
−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1

```
>>> story[-8]
's'
>>> story[5]
's'
```

String Manipulation: Slicing

- It is possible to "slice" a string by specifying a left (`L`) and (non-inclusive) right (`R`) `int` value:

```
>>> story[1:11]
't was a da'
```

N.B. the sliced substring length $=$ `R` $-$ `L`

- By default, `L=0` and `R` is the length of the string:

```
>>> story[:-7]
'It was'
```

- It is also possible to specify slice "direction":

```
>>> story[:-7:-1]
'krad a'
```

Class Exercise (2)

- Generate the "middle half" of a given string

Strings and Formatting

Often we want our output to be pretty.
Use the `format()` method of a string:

```
>>> "{0} and {1}".format(1,1.0)
'1 and 1.0'
>>> "{0:.2f} and {0}".format(1,1.0)
'1.00 and 1'
>>> "{0:d} {0:x} {0:o} {0:b}".format(42)
'42 2a 52 101010'
>>> "{0[0]} {0[1]}".format('abcdef')
'a b'
```

Method: a function that is a member of an object.
Object: a collection of data and functions. eg `str`

More later in the course - don't panic

Character Representation

- Computers like bits, and so represent characters as (positive) integer codes
- Python3 defaults to UTF-8 encoding: Unicode, with 8 bits for ASCII, where the character `'A'` has a numerical value of 65, `'B'` is 66, ...
- Code↔character conversion:
  - `ord()`: convert an ASCII character into its code
  - `chr()`: convert an `int` code (0–255) into its corresponding ASCII character
- This is important when we sort strings/check for string "precedence"

Lecture Summary

- Type conversion: what and how?
- Comments: what and how?
- Character representation: how are characters represented, and how can they be converted from/to their internal representation?
- Strings: what are indexing, slicing and splitting? how do we format strings?