COMP10001 Foundations of Computing
Python Basics

Semester 1, 2019
Tim Baldwin, Nic Geard, Farah Khan, and Marion Zalk

THE UNIVERSITY OF MELBOURNE

— version: 1472, date: March 12, 2019 —

Reminders

- All workshops (W01 and W02) are now in full swing
- Complete Worksheets 0, 1 and 2 by the end of Monday of next week
- Email the lecturers urgently if you are having issues accessing the Grok materials, or the forums on Grok:

  `comp10001-lecturers@lists.unimelb.edu.au`

Lecture Agenda

- Last lecture — Grok Worksheet 0
  - Programming basics with Blockly
- This lecture — Grok Worksheet 1
  - Literals
  - Types

Lecture Outline

Python Basics

To start out, let's use Python as a glorified calculator:

- basic arithmetic: `+` (addition) `-` (subtraction) `/` (division) `*` (multiplication)
- also: `**` (exponent), `%` (modulo), `//` (integer/"floor" division)
- 

Beware, in Python2, $10/3 = 3$

Python Basics

- Python uses "BODMAS" to associate "operands" (the targets of operations) by default, which you can override with "parentheses":
  - `1 + 2 * 3` vs. `(1 + 2) * 3`
- special character `_` stores the value of the last calculation

Class Exercise

- Armed just with these operators, let's explore the limitations of what we can do; is it possible to:
  - calculate $n!$ ($= n \times (n-1) \times ...2 \times 1$) for an arbitrary n?
  - calculate the ith Fibonacci number?
  - numerically "break" Python?

Turtle Programming Redux

- Let's return to Turtle graphics briefly to introduce Python "commands":
  - `forward(N)`: advance forward `N` units
  - `left(N)`: turn left `N` degrees
  - `circle(X)`: draw a circle of radius `X`
- Need to have the following line of code before issuing any of these commands:

```
from turtle import *
```


Class Exercise

- What does the following code do:

```
from turtle import *

forward(20)
left(90)
circle(20)
right(90)
forward(60)
left(90)
circle(80)
left(90)
forward(160)
```


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

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

so if you use Python2 code from the web, remember to convert `print` statements to `print` functions

The print Function

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

```
>>> print(1)
1
```

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

```
>>> print(1)
1
>>> 1
1
```

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

Jargon Alert

- Syntax: "the arrangement of words and phrases to create well-formed sentences in a language"

`print hello''()`	incorrect syntax
`print('hello')`	correct syntax

- Semantics: "the meaning of a word, phrase, or text"

`print(1 + 2)`	"+" = add two numbers

Lecture Outline

Types (1)

- In Python, every object has a "type", which defines: (a) what operators, "functions", and "methods" can be applied to it, and (b) its semantics
- The two number types we will see most of are:
  - `int` (integer)
  - `float` (real number)

  also `complex` for complex numbers
- So how does Python work out the type for a given (real) number? If it contains a decimal place (`.`), it's a `float`, otherwise it's an `int`

Types (2)

- Use the function `type` to determine the type of an object:

```
>>> print(type(1))
<type 'int'>
>>> print(type(1.0))
<type 'float'>
```

- The semantics of operators and functions is determined by the types of the operands:

```
>>> print(type(1 + 2))
<type 'int'>
>>> print(type(1.0 + 2))
<type 'float'>
```


Lecture Summary

- Types: what are they, what basic types have we learned, and how do you determine the type of an object?
- What are the basic numeric types, what operators are associated with them, and what interactions are there between the types and operators?
- Type conversion: how do we test the type of an object, and how do you convert the type of an object?

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 "convert" a literal/variable to a different type, we use functions of the same name as the type: `int()`, `float()`, `str()`, `complex()`

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