ISCG 7420
Web Application Development
S1 2020

Week 1

Admin, Preliminaries

Admin - About Me

- Kris Pritchard
- Software Consultant / Engineer
- kpritchard@unitec.ac.nz
- Room 183-1001 (Email first to check if/when I’m there)

Admin - Course Details

- Course Length: 150 hours
  - Contact / Lecture: 39 hours (3hrs/week for 13 weeks)
  - Unsupervised / Student-Managed: 111 hours (9hrs/week for 13 weeks)
- You're expected to spend at least **9hrs/week** on student-managed study and learning material for this course. **If you spend less time than this you will likely struggle**. Please make sure you're meeting these minimum requirements, and ask me for help if you get stuck on anything.
- 2 Assignments:
    - 1 worth 60%, 1 worth 40%

Admin - Group Chat

- Manas has created a class group chat, please join it here:
    - https://discord.gg/kwQjXhN
- I've also created a Github repository which we can use for course material and projects:
    - https://github.com/kris-classes/7420/
- Email me your Github username to be added.

Preliminaries

- We'll be using Python for this course, because it allows you to develop a working complex web application in a very short amount of time. The knowledge you get from this course can be applied to web applications built using any of the other major languages (C#, Ruby, JavaScript/ NodeJS, PHP, Java, and others)

Preliminaries

- We'll also be using a web framework known as Django, which provides a large amount of functionality to start from.
- Comparable web frameworks:
- C# / ASP.NET
- Ruby / Ruby on Rails
- PHP / Laravel
- Java / Spring
- See https://en.wikipedia.org/wiki/Comparison_of_web_frameworks for a complete list.

Preliminaries

- Repeat and practice everything here until you know it off by heart.
- All of these basics will be required not only for assignments, but also in your career.

Preliminaries

- Data Types (`int`, `bool`, `float`, `None`, `string` types & `string` formatting)
- Abstract Data Types (`list`, `tuple`, `range`, `set`, `dict`)
- Control Flow (`if/else/elif/for/while`)
- Functions & Classes
- Modules, Files, Exceptions

Preliminaries
(Repeat everything here until you know it off by heart)

- Use the built-in `help()` function from the Python Shell

```
>>> help(int)
Help on class int in module builtins:

class int(object)
 |  int([x]) -> integer
 |  int(x, base=10) -> integer
 |
 |  Convert a number or string to an integer, or
```

- `help(int)`, `help(str)`, `help(bool)`, `help(None)`, `help(float)`
- `help(list)`, `help(dict)`, `help(tuple)`, etc.
- Functions with two underscores (`__gte__` etc) are called 'dunder'/'double-underscore' functions or *magic methods*, and are *sort of* equivalent to private methods in other languages. We'll come back to them later when covering __init__, __repr__, etc.

Preliminaries - Data Types

- You can use `type()` to check the data type of something
- `type(124)` # Is of type <class 'int'>
- `type(342.12)` # <class 'float'>
- What about `True/False`? What about strings?
- If you're using the Python Shell IDLE you might need to do `print(type(124))`
- You can also use `isinstance(my_list, list)` to check an objects type.

Preliminaries - Data Types

- Beginner's Python Cheat Sheet (See Moodle) or https://ehmatthes.github.io/pcc_2e/cheat_sheets/cheat_sheets/ # You will need to thoroughly revise & know everything from Overview, Lists, Dictionaries, If Statements and While Loops, Functions, Classes, Files and Exceptions, and Testing Your Code for this course. These are just the basics (taught in ISCG5420 and 5421) so please practice them if you don't already know them off by heart.
- Primitive Data Types: Integer, Float, Boolean, Strings
- Some types of Literals:
    - 4, 123, 125333932 # Are all `int` literals
    - 1.243, 3.14 # Are `float` literals.
    - 'hello' # Is a `string` literal

Preliminaries - Data Types

- Type conversion/casting. You can convert one data type to another.
- `int('234')` # Converts the string '234' into an integer.
- `str(12.0)` # Converts the float 12.0 into a string.
- `bool(0)` # Converts the integer 0 into a bool.
- `bool(42)` # Converts the integer 42 into a bool.
- `int('hello')` # See what happens when you try this. Try other examples.
- What happens if you try to convert a list to a float?

Preliminaries - Operators and Operands

- Common Operators: + (add), - (subtract), * (multiply), / (divide)
  - // (floor division), % (modulus/remainder), ** (exponent/power)
  - < (less than), <= (less than or equal), == (is equal), > (greater than), >= (greater than or equal)
  - e.g. `2 + 3  # In this expression 2 and 3 are operands , and + is the operator`
- Table of built-in operators: https://docs.python.org/3/library/operator.html#mapping-operators-to-functions # Revise the table here and make sure you're familiar with everything. Ask me for help if there's anything you don't know or understand. Ignore Matrix Multiplication (the @ operator).
- Later on in the course we'll cover an object-oriented concept known as *operator overloading*, which allows you to override what an operator such as + does.
- An example is that `2 + 3` adds two ints, but 'hello' + 'world' concatenates two strings together.

Preliminaries - Operators and Operands

- Logic Operators:
    - `and` (Logical AND, sometimes `&&` in other programming languages)
    - `or` (Logical OR, `||` in other languages)
    - `not` (Logical NOT, `!` in other languages)
    - e.g. `if username is not 'Guido': # Do something if a username is not Guido`
- Bitwise Operators: x `<<` y (left-shift), x `>>` y (right-shift), x `&` y (bitwise-AND), x `|` y (bitwise OR), `~`x (bitwise complement), x `^` y (bitwise XOR)
- `a, b = b, a` # Swap two variables, This is called tuple-unpacking.

Preliminaries - Strings

- Official tutorial: https://docs.python.org/3/tutorial/introduction.html#strings
- Official docs: https://docs.python.org/3/library/stdtypes.html#string-methods
- Familiarize yourself with how `capitalize()`, `casefold()`, `format()`, `lower()`, `join()`, `split()`, `upper()`, and others from above work.
    - e.g. `s = 'OPTIMUS priME'`
    - `s.title()`

Preliminaries - Strings

- String Formatting (and f-strings in Python 3.6 and above)
- Good reference: https://pyformat.info/ # Read and try out all of these. You don't need to memorize them, but you need to be aware of them.
- `name = 'Bob'`
  - Ancient: `print("hello %s" % name)`
  - Oldish: `print('hello {}'.format(name))`
  - New `print(f'hello {name}')`

Preliminaries - Strings

- `my_string = 'some,collection,of,comma,separated,values,123,4.56,True'`
- `my_string.split(',')` # Split my_string into a list, using the comma as a separator.
- Note: Each individual item in the list is still a string. Other types are not implicitly converted.
- Play around with the other string methods from the previous slide and those from the official documentation.

Preliminaries - Functions

- Official tutorial: https://docs.python.org/3/tutorial/controlflow.html#defining-functions # Read 4.7 - 4.7.2.

- ```
  def say_hello(name):
      print('You entered ' + name) # format() or f-strings for Python3.6+
      greeting = 'hello ' + name
      return greeting
  ```

- ```
  say_hello('Ada')
  ```

- `say_hello` is the *function name*, `name` is a *parameter*, `greeting` is the *return-value*, and `'Ada'` is the *argument* we *pass* to the function when *calling* the function.

Preliminaries - Flow Control

- `if`, `elif`, `else`
- `for`, `while`, `break`, `continue`
-
```
for i in range(1, 100, 5):
    if i % 2 == 0:
        if i == 42:
            print('What is 6 x 9?')  # Hint: hhgttg
            continue
        print('{} is even'.format(i))
    elif i == 69:
        print('Stopping at 69.')
        break
    else:
        print('{} is odd'.format(i))
```

Exercise - Problem

- Write a program loops from 1 to 100 (inclusive).
- If the number is a multiple of 3, print ‘Fizz’
- If the number is a multiple of 5, print ‘Buzz’
- If the number is a multiple of BOTH 3 and 5, print ‘FizzBuzz’
- Otherwise just print the number.
- Try to do it without using the hint. Hint is on the next slide.

Exercise - Hint

- Hint: Use a for loop, range(), % (modulo), and if/elif/else statements.
- Solution is on the next slide.

Exercise - Solution

- Congratulations! You just passed the FizzBuzz test.
- If you struggled with this, you really need to practice the fundamentals by next week's lecture.
- Your program should look something like the following.

```
for i in range(1, 101):
    if i % 3 == 0 and i % 5 == 0:
        print('FizzBuzz')
    elif i % 3 == 0:
        print('Fizz')
    elif i % 5 == 0:
        print('Buzz')
    else:
        print(i)
```

- FizzBuzz is a common job interview question used for filtering applicants. It tests your understanding of fundamentals and is a good indicator of how much practice you need.
- Some other solutions of it: https://en.wikipedia.org/wiki/Fizz_buzz#Programming and http://rosettacode.org/wiki/FizzBuzz#Python3:_Simple

Preliminaries - Abstract Data Types

- Lists
- Tuples (Immutable Lists)
- Dictionaries (Map, Hashmap, or Hashtable in other languages)
- Sets

Preliminaries - Lists

- Lists (*mutable* / able to be mutated, and *ordered*)
- Official tutorial: https://docs.python.org/3/tutorial/datastructures.html#more-on-lists # Read 5.0 - 5.2, but skip 5.1.3 - 5.1.4. We can cover List Comprehensions later in the course.
- Creating a list: `my_list = [1, 2, 3, 4, 5]`
- Indexing/Accessing List elements: `my_list[3]`
- List Slicing: `my_list[:]`, `my_list[2:5]`, `my_list[2:5:-1]`, `my_list[::-2]`

Preliminaries - Lists

- Checking existence: `if 2 in my_list: # Do something`
- Learn these from help(list): `append, clear, count, index, insert, pop, remove, reverse`, difference between `my_list.sort()` vs `sorted(my_list)` (in-place sort vs copied)
- e.g. `my_list.append(42), my_list.insert(420, 5), my_list.pop(), etc.`
- `nested_list = [ [1, 2, 3], ['hello', 'world'], [24.0, True] ]`
- `nested_list[2][0]  # is 24.0 from above.`

Preliminaries - Tuples

- Tuples are like *immutable* lists (unable to be mutated or changed after creation)
- Official tutorial: https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences
- Official docs: https://docs.python.org/3/library/stdtypes.html#tuple

```
>>> help(tuple)
Help on class tuple in module builtins:

class tuple(object)
 |  tuple(iterable=(), /)
 |
 |  Built-in immutable sequence.
 |
 |  If no argument is given, the constructor returns an empty tuple.
 |  If iterable is specified the tuple is initialized from iterable's
 |
 |  If the argument is a tuple, the return value is the same object.
 |
 |  Methods defined here:
```

Preliminaries - Tuples

- `my_tuple = (3840, 2160)  # Creates a tuple with two values.`
- Creating a tuple with only one element: `single_tuple = (1234, )`
- `type(single_tuple)  # Check what you get`
- `width, height = my_tuple  # Uses 'tuple-unpacking' to extract values.`
- We saw tuple-unpacking earlier with swapping two variable values. It works for tuples of any size.
- e.g. `a, b = b, a` instead of:

```
tmp = a  # Swapping variable values in other programming languages.
a = b
b = tmp
```

Preliminaries - Tuples

- You can access tuple items the exact same way as with lists.
- `my_tuple[2]` # Gets item from index 2 in my_tuple.
- When should you use tuples? See: https://stackoverflow.com/a/1708538/272666
- Summary: Tuples can’t be resized or changed. You can quickly check if elements exist in a tuple, so they’re good for speed when you need some set of values that don’t change.

Preliminaries - Dictionaries

- Dictionaries (keys must be unique. Try what happens when they're not)
- Official tutorial: https://docs.python.org/3/tutorial/datastructures.html#dictionaries # Read 5.5-5.8.
- API reference: https://docs.python.org/3/library/stdtypes.html#typesmapping # Read down to (but excluding Context Manager Types)
- `empty_dictionary = {} # Create an empty dictionary. Similar to empty_list = [].`
- `my_dictionary = { 'a_key': 'some_value' } # Create a dictionary`
- help(dict): `clear, copy, get, items, keys, pop, popitem, setdefault, update, values`

Preliminaries - Dictionaries

- `my_dictionary['newthing'] = 'something new'` # Add a new item to an existing dictionary.
- `my_dictionary['newthing'] = 234` # Overwrites 'something new' with 234
- `my_dictionary['newthing']` # Get the value stored by this key (234). Will raise a KeyError exception if key `'newthing'` does not exist.
- `my_dictionary.get('newthing')` # Will instead return None (equivalent to null in other languages) if the `'newthing'` key does not exist.
- `my_dictionary.get('newthing', 123)` # Checks for the 'newthing' key, and returns a default value (of 123) if the key does not exist.
- These are all extremely useful, so be sure to learn them thoroughly.

Preliminaries - Dictionaries

- Looping:
- `my_dictionary.keys()` # Returns all the keys.
- `my_dictionary.values()` # Returns all the values.
- `my_dictionary.items()` # Returns both the keys and items, as tuple pairs.
-
```
for items in my_dictionary.items():
    print(items) # Prints them as tuples
```

-
```
for k, v in my_dictionary.items():
    print('key is {} and value is {}'.format(k, v)) # Uses tuple-unpacking to print individually.
```

Preliminaries - Sets

- Sets (unordered and no duplicates)
- Official tutorial: https://docs.python.org/3/tutorial/datastructures.html#sets # Read 5.4
- Official docs: https://docs.python.org/3/library/stdtypes.html#set # Read the *set* section.
- Very useful for certain problems.

Preliminaries - Sequence Type Operators

- `list`, `tuple`, and `range` objects are the 3 basic sequence types - https://docs.python.org/3/library/stdtypes.html#typesseq

Operation	Result
`x in s`	`True` if an item of *s* is equal to *x*, else `False`
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`
`s + t`	the concatenation of *s* and *t*
`s * n` or `n * s`	equivalent to adding *s* to itself *n* times
`s[i]`	*i*th item of *s*, origin 0
`s[i:j]`	slice of *s* from *i* to *j*
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*
`len(s)`	length of *s*
`min(s)`	smallest item of *s*
`max(s)`	largest item of *s*
`s.index(x[, i[, j]])`	index of the first occurrence of *x* in *s* (at or after index *i* and before index *j*)
`s.count(x)`	total number of occurrences of *x* in *s*

Preliminaries - Classes

- ```
  class Clock:
      current_time = 6
      def tick(self):
          print('Ticking')
          self.current_time += 1
          return self.current_time
  ```

- ```
  my_clock = Clock()  # Create an instance of a Clock object
  ```

- ```
  my_clock.tick() # Tick once.
  ```

- ```
  my_clock.tick() # Tick again.
  ```

Preliminaries / Modules, Files, Exceptions

- ```
  import math # Importing. Type it into a Python shell.
  math.sqrt(9)
  math.log2(26)

  from math import ceil, floor  # Import single objects
  ceil(4.2)
  floor(4.7)
  ```

- ```
  with open('myfilename.txt') as myfile: # Opening a file.
    lines = myfile.readlines()
  print(lines)
  ```

- ```
  try:  # Exception handling
    x = int(input('Enter something: '))  # Try enter a string
  except ValueError:
    print('Must enter a number')
  else:
    print('Entered {}'.format(x)
  finally:
    print('Doing some final thing')
  ```
  ```

Next Week

- In your **9 hours** of weekly self-directed study I need you to revise & practice all of the fundamentals from these slides.
- If you need more information than these slides provide, please use the following cheat sheets. We'll be using material from *Overview*, *Lists*, *Dictionaries*, *If Statements and While Loops*, *Functions*, *Classes*, *Files and Exceptions*, and *Testing Your Code*.
    - Beginner's Python Cheat Sheet: https://ehmatthes.github.io/pcc_2e/cheat_sheets/cheat_sheets/
- In Week 2 we'll briefly review what you've learned in Week 1, then get started with installing packages and building a web application.
- Going forward I'll assume that everyone is familiar with all of the basics covered and referenced in these slides.