Python refresher

Python refresher

In Weeks 4 to 6 you will be writing Python programs to work with big data.

We assume that you already know Python, either from doing the course ZZEN9021 Principles of Programming or something equivalent, so we won't be teaching it to you. But here is a refresher on the aspects of Python that are particularly important for this course.

Printing

You can print using the `print()` function. If you supply the function with multiple arguments it prints them all, separated by spaces:

```
print('Hello')
print('Hello' + ' ' + 'world')
print('Hello', 'world')
```

To print a tab character you can use `\t`, and for a newline character you can use `\n`:

```
print('Hello' + ' ' + 'world')
print('Hello' + '\t' + 'world')
print('Hello' + '\n' + 'world')
```

Working with text

You can find the length of a piece of text using the `len()` function:

```
print(len('Hello'))
```

You can strip whitespace from the ends of text using the `strip()` method:

```
print(len('  Hello '))
print(len('  Hello '.strip()))
```

You can strip other characters by passing them as an argument string:

```
print('Hello.'.strip())
print('Hello.'.strip('.'))
```

You can split a piece of text into a list using a certain character is the splitting point, using the `split()` method:

```
passage = "Hello. My name is Ben. I am 185cm tall."
words = passage.split(' ')
sentences = passage.split('.')
print(words)
print(sentences)
```

Working with lists

You can create a list using a list literal:

```
numbers = [1, 2, 3, 4, 5]
print(numbers)
```

You can get the number of items in a list using the `len()` function:

```
numbers = [1, 2, 3, 4, 5]
print(len(numbers))
```

You can access items in a list using the `[ ]` operator:

```
numbers = [1, 2, 3, 4, 5]
# Print the first item:
print(numbers[0])
# Print the last item:
print(numbers[-1])
```

You can loop through a list using a `for ... in ...` statement:

```
numbers = [1, 2, 3, 4, 5]
for n in numbers:
    print(2 * n)
```

You can sort a list using the `sort()` method. Note that `sort()` sorts the list in-place, and does not return a value:

```
words = ['dog', 'cat', 'mouse', 'goat', 'python']
words.sort()
print(words)
```

You can join the items in a list into a string using the join() method. Note that this is a method of the joining text, not of the list:

```
letters = ['h', 'e', 'l', 'l', 'o']
```

```
print(''.join(letters))
print('-'.join(letters))
print('[ ]'.join(letters))
```

Working with dictionaries

You can set and get values in a dictionary using the `[ ]` operator:

```
pet_frequencies = {'dog': 17, 'cat': 12}
print(pet_frequencies['dog'])
pet_frequencies['mouse'] = 3
print(pet_frequencies['mouse'])
```

If you try to get the value of a key that doesn't exist you will get an error. To avoid this you can use the `get()` method, which return `None` if the key does not exist. You can optionally specify an alternative value to return if the key does not exist:

```
pet_frequencies = {'dog': 17, 'cat': 12, 'mouse': 3}
print(pet_frequencies.get('hare'))
print(pet_frequencies.get('hare', 0))
print(pet_frequencies.get('hare', 'No such pet'))
```

You can get the keys of a dictionary as a list using the `keys()` method:

```
pet_frequencies = {'dog': 17, 'cat': 12, 'mouse': 3}
print(pet_frequencies.keys())
```

You can iterate through the keys of a dictionary using a `for ... in ...` statement:

```
pet_frequencies = {'dog': 17, 'cat': 12, 'mouse': 3}
for key in pet_frequencies:
    print(key, pet_frequencies[key])
```

You can iterate through the key-value pairs of a dictionary by using the items() method:

```
pet_frequencies = {'dog': 17, 'cat': 12, 'mouse': 3}
for key, value in pet_frequencies.items():
    print(key, value)
```

Type conversions

You can get errors if you use a value of type `int` when Python is expecting a value of type `str`, and vice-versa.

```
word = 'Hello'
```

```
number = 2
print(word + number)
```

```
word = 'Hello'
number = 2
print(word/number)
```

To avoid these errors you can force a value to be an `int` using the `int()` function, or a `str` using the `str()` function:

```
word = 'Hello'
number = 2
print(word + str(number))
```

Built-in functions

Python has many inbuilt functions. Here are some we will be using:

```
# max() takes two or more arguments and returns the maximum value
print(max(3, 5, 0, 4))

# max() can also take a list as argument
print(max([3, 5, 0, 4]))

# sum() takes a list as argument and returns the sum of its elements
print(sum([3, 5, 0, 4]))
```

Lambda functions

By using the `lambda` keyword you can have function literals (also called anonymous functions):

```
add = lambda x, y: x + y
print(add(3, 4))
```

These are especially useful when using functions that take a function as argument, such as `map()`:

```
words = ['dog', 'cat', 'mouse', 'goat', 'python']
first_letters = map(lambda word: word[0], words)
for letter in first_letters:
    print(letter)
```

Yielding values

Suppose you want a function to return a list of values, say the first five integers (whole numbers). Here's one way to do it:

```
def numbers():
    return [1,2,3,4,5]
for x in numbers():
    print(x)
```

When `numbers` is called in line 3 it returns the list of numbers and the list gets put into memory. That's okay if the list is small, but if the list is large then it can put a strain on memory. And if all you are doing is looping through the list one-by-one then you don't need the whole list in memory anyway - you just need one item at a time. An alternative is to use the `yield` keyword:

```
def numbers():
    for x in [1,2,3,4,5]:
        yield x
for x in numbers():
    print(x)
```

In this case `numbers` does not return a list - it returns a **generator**. When you iterate through this generator in line 4 it generates the numbers one-by-one as they are needed.

Defining classes

You can define your own class of objects using a `class` statement:

```
class Person:
    def __init__(self, firstName, lastName):
        self.firstName = firstName
        self.lastName = lastName
    def fullName(self):
        return self.firstName + ' ' + self.lastName
    def reverseName(self, initialOnly = False):
        if initialOnly:
            return self.lastName + ', ' + self.firstName[0] + '.'
        else:
            return self.lastName + ', ' + self.firstName
person = Person('Jacinda', 'Ardern')
print(person.fullName())
print(person.reverseName())
print(person.reverseName(True))
```

Instance methods have `self` as their first argument, and then optionally have further arguments. In the example above, the method `reverseName()` has another argument.

When defining a class you can specify that it extends another class, which means that it inherits the properties and methods of that class. You can add new properties and methods, and override

inherited ones:

```
class Person:
    def __init__(self, firstName, lastName):
        self.firstName = firstName
        self.lastName = lastName
    def fullName(self):
        return self.firstName + ' ' + self.lastName
class Boy(Person):
    def fullName(self):
        return 'Master ' + self.firstName + ' ' + self.lastName
person = Person('Jacinda', 'Ardern')
print(person.fullName())
boy = Boy('Stuart', 'Simpson')
print(boy.fullName())
```

Importing code

You can import code from another program into your program:

```
import math
```

This makes all of the code in the program math.py available for use in your program. For example, in math.py there is a function `floor()` which rounds a number down to the nearest whole number. Having imported math.py you can use this function in your program:

```
import math
print(math.floor(2.76))
```

Note that you must attach a prefix to the function name when you use it: `math.floor()`. If you import the function directly using the `from` keyword, and then you don't need to use the prefix:

```
from math import floor
print(floor(2.76))
```

Being able to import a program into other programs means that there are two ways that it can run: as the main program, or as an imported program. Python has a special variable `__name__` that you can use in the program to check whether it's running as the main program or as an imported program. If it's running as the main program then the value of `__name__` will be "__main__", otherwise it will be the filename of the program. Suppose you have written the following program and called it "myFunctions.py":

```
def triple(x):
    return x * 3
if __name__ == '__main__':
    print(triple(6))
```

When you run the program as the main program the value of `__name__` is "__main__", so the last line is executed and 18 is printed. When you import the program into another program the value of `__name__` is "myFunctions", so the last line is not executed and 18 is not printed.

Using standard input

If you'd like your program to use standard input you can use `sys.stdin`. A newline character marks the end of the input and is included in the input, so you often need to use strip() to remove it:

```
import sys
for line in sys.stdin:
    line = line.strip()
    if line == 'exit':
        break
    else:
        print('You entered ' + line)
print('Done')
```

Check yourself

To check your Python skills, try writing the following two programs.

First program

In the panel on the right is a program called "words.py". It has a variable called "sentence" whose value is a sentence from E.B. White's story "Stuart Little". Your task is to finish writing this program so that it prints the average length of words in the sentence, to two decimal places (be aware not to count commas and full stops).

To run your program, enter the following command into the terminal:

```
$ python words.py
```

The correct answer is 4.49

Second program

Also in the panel on the right is a program called "sales.py". It has a variable called "sales" whose value is a fictitious set of sales data. Sale items are separated by semicolons (;). Within a sale item there are three pieces of data, separated by commas (,): product id, quantity sold, and sale price in dollars. Your task is to finish writing this program so that it prints the total sales amount in dollars.

To run your program, enter the following command into the terminal:

```
$ python sales.py
```

The correct answer is $898.64

Hint: You can split `sales` into its items using the `split()` method, and then loop through those items. And you can split each item into its three pieces of data also using the `split()` method.

Refresh some more

If you'd like to further refresh your Python skills there are two Python courses available to you as a UNSW student, at LinkedIn Learning.

Learning Python (2 hrs 11 mins)

This course provides an overview of the installation process, basic Python syntax, how to construct and run a simple Python program, how to work with dates and times, how to read and write files, and how to retrieve and parse HTML, JSON, and XML data from the web. Topics include:

- Installing Python
- Choosing an editor or IDE
- Working with variables and expressions
- Writing loops
- Using the date, time, and date time classes
- Reading and writing files
- Fetching internet data
- Parsing and processing HTML

Advanced Python (2 hrs 27 mins)

This course explains object-oriented programming with Python, how to log and track performance and user activity, how to port code from Python 2 to 3, and how to make your code more efficient and easier to read and maintain. Topics include:

- Truth value testing
- Template strings
- Iterators
- Transforms
- Advanced Python functions
- Advanced collections
- Advanced classes and objects
- Logging
- Python comprehensions: list, dictionary and set