Python Lesson 4

Data Structures

We've already seen several different types of value in Python:

- Strings (str), like "Hello World" or "Sorry Dave, I can't do that".
- Integers (int) which are whole numbers like 0, 10001 or -79.
- Floating Point (float) which are numbers with decimal points like 0.0, 3.141592653589793 or -99.999.
- Boolean (bool) values True and False.

Python has a number of other data types which can behave as collections or containers of other values: **lists**, **sets** and **dictionaries**. You can put other values into these collections, examine items in the collection (get and iterate), and remove items.

Lists

Lists in Python are ordered collections of elements. Each element in the list has a position or **index**, starting from zero.

Here is an example of creating an empty list and add three string values to it. Note that lists can contain any type of value, not just strings:

```
>>> my_list=[]
>>> my_list.append("sausages")
>>> my_list.append("eggs")
>>> my_list.append("beans")
>>> my_list
['sausages', 'eggs', 'beans']
```

Instead of adding elements one at a time, we could have done this:

```
>>> my_list=["sausages", "eggs", "beans"]
```

Find the size of a list:

```
>>> len(my_list)
3
```

Read items from a list by position. Note that the start of a list is position zero!

```
>>> print(my_list[1])
'eggs'
```

Replace an item in a list:

```
>>> my_list[0]="bacon"
>>> my_list
['bacon', 'eggs', 'beans']
```

```
>>> my_list.reverse()
>>> print(my_list)
```

```
['beans', 'eggs', 'bacon']
```

```
>>> my_list.sort()
>>> my_list
['bacon', 'beans', 'eggs']
```

Sets

Sets in Python contain values, as lists do, but the values aren't stored in any particular order.

You can think of them a bit like a bag containing values. The items in the bag may be jumbled up while you're not looking, so it doesn't make any sense to refer to the items by index.

Unlike lists, you can't store two different copies of the same value in a set. If you add the same value to the set twice, you'll only see it once.

Sets are really most useful when you want to store a collection of items and check *very quickly* whether the item is present in the set. Searching a set of a million items can be much faster than searching a list of a million items. Removing items from the a set can also be faster than removing items from the start of a list – the items in the list would all need to be shuffled up to fill the gap!

Sets use { } brackets instead of [ ].

```
>>> my_set={"sausages", "eggs", "beans"}
>>> my_set
{'eggs', 'sausages', 'beans'}
>>> "eggs" in my_set
True
>>> "tomato" in my_set
False
>>> my_set.add('bacon')
>>> my_set
{'bacon', 'eggs', 'sausages', 'beans'}
```

Adding bacon to a set where it already exists does nothing!

```
>>> my_set.add('bacon')
>>> my_set
{'bacon', 'eggs', 'sausages', 'beans'}
```

```
>>> my_set.discard("bacon")
>>> my_set
{'eggs', 'sausages', 'beans'}
```

While you're getting started with Python, if you're not sure whether to use a list or a set, I would suggest using a list. If your program ends up being too slow, you can consider switching to a set.

Super Speedy Maths: Calculating Prime Numbers

Prime numbers are numbers which only appear in the 1 times table and their own times table. For example, 11 is a prime number as it is only in the 1 times table and 11 times table; 9 is not a prime because it's also in the 3 times table.

We'll write a short program that uses lists and sets to calculate *all* the prime numbers up to 1000.[1] This is how it works:

1. Put all the numbers between 2 and 1000 into a set.
2. Count through all the times tables from 2 to 1000. For each times table, calculate all the values in that times table up to 1000, skipping the 1x table. Remove these from the set.
3. The numbers left in the set at the end will be just the prime numbers!
4. Finally the program copies the results into a list so that we can sort them before printing. Remember that sets are unordered bags of values, so if we didn't do this the values could come out in the wrong order.

Here is a small example to calculate the prime numbers less than 10.

- start with 2, 3, 4, 5, 6, 7, 8, 9 and 10 in a set
- discard 4, 6 and 8 because they are in the 2 times table
- discard 6 and 9 because they are in the 3 times table
- discard 8 (again!) because it is in the 4 times table – this doesn't actually do anything because the item was already removed!
- we are left with 2, 3, 5 and 7 in the set at the end of the program. These are the prime numbers less than 10.

Using this approach (an "**algorithm**", or set of instructions) Python can calculate a set of prime numbers much faster than even the speediest speedy maths expert.

File prime_numbers.py:

```
maximum=1000

# Create an empty set and put every number into it
prime_numbers=set([])
for number in range(2, maximum):
    prime_numbers.add(number)

# Count through the times tables between 2 and 1000 and remove all the answers
for times_table in range(2, maximum):
    for value in range(2*times_table, maximum, times_table):
        prime_numbers.discard(value)

# Copy the prime numbers into a list, sort and print
sorted_primes=list(prime_numbers)
sorted_primes.sort()
print("There are", len(sorted_primes), "prime numbers less than", maximum)
print(sorted_primes)
```

1You can pick a higher maximum value if you want, but after about 1 million or 10 million the program might get too slow, or even eventually run out of memory to store all the numbers in a set!

Dictionaries

Dictionaries store pairs of items: a key, and a value. You could use a Python dictionary data structure to create an English dictionary, where each key is a word and the value is a definition of that word. Dictionaries use { } brackets like sets, but each entry in the dictionary uses a colon to separate the key and value, like *key*: *value*. For example:

```
>>> my_dictionary={
"aardvark": "A nocturnal burrowing mammal from Africa.",
"python": "A large non-venomous snake.",
"zebra": "An African wild horse in striped pyjamas."}
>>> my_dictionary["python"]
'A large non-venomous snake.'
```

You can add items to a dictionary:

```
>>> my_dictionary["internet"]="A global computer network."
```

You can look up items in a dictionary:

```
>>> my_dictionary["internet"]
```

or

```
>>> my_dictionary.get("internet")
```

The two different ways of looking something up from a dictionary do the same thing *unless the item isn't in the dictionary*. The `dict.get(item)` function will return the special value **`None`** if the item isn't found, but `dict[item]` will fail with an error (it will "throw an exception"). To see the difference try **`my_dictionary["spoon"]`** and **`my_dictionary.get("spoon")`**.

You can count the number of entries in a dictionary:

```
>>> len(my_dictionary)
```

You can replace items in a dictionary:

```
>>> my_dictionary["internet"]="A global computer network for sharing cat
videos."
>>> my_dictionary["internet"]
```

You can remove items from a dictionary:

```
>>> del my_dictionary["internet"]
```

You can check if a dictionary contains a key:

```
>>> "aardvark" in my_dictionary
True
>>> "spoon" in my_dictionary
False
```

Classic Text Adventure: Escape from Stamford Green!

For this project we will be using a **dictionary** data structure to create a fictional map of Stamford Green School.

We will create a simple game that allows you to walk around the school map using commands "north", "south", "east" and "west". You need to explore and find the exit!

Each entry in the dictionary will have a key that names a room or location in the school (e.g. "the_office") and a value that describes the properties of that room. The properties of each room will be stored in another dictionary – this means that the map of the school is a dictionary containing more dictionaries!

- The "description" property of each room will be the text that is displayed to the player when they enter the room.
- The "north", "south", "east" and "west" properties of the room will contain identifiers of the rooms where the player will end up if they walk in that direction.

Before we get to the real game, here's a short example which creates a school map containing only a single location called "the_office". It has a description, and it has other properties which contain the names of different locations that you'll end up in if you walk in any direction from the office.

```
school_map={
    "the_office": {
        "description": "You are standing outside the school office. You are surrounded by
gleaming sofas made of solid gold. Paths lead in all directions.",
        "north": "east_gate",
        "east": "staff_room",
        "south": "hall"
    }
}

location="the_office"
school_map[location]["description"] # gets the_office room description
school_map[location].get("east")    # gets the name of the room to the east, 'staff_room'
school_map[location].get("west")    # gets the name of the room to the west, None
```

The only other new thing in this Python program that we haven't seen in any other lesson is the `textwrap` module, which is imported at the start. This provides a function called `textwrap.fill()` which formats text by inserting newline characters so that it prints nicely on 80 column displays. It makes the room descriptions nicer to read, without words being cut off at the end of a line.

The full program that you can type in is on the next page, but since it's so long we'll try to provide a saved copy that you can open and read through yourself, so you don't need to get held back by your typing speed! Try looking in "Lesson 4" at https://github.com/chrisglencross/python-lessons.

Once you have the program working, feel free to make changes of your own. Add your own rooms, change room descriptions, make objects or treasure that you can pick up (find a key to open the west gate?), add monsters or teachers, and do whatever you want. But most of all, be creative and have fun!

File escape_from_stamford_green.py:

```
import textwrap

school_map={
    "the_office": {
        "description": "You are standing outside the school office. You are surrounded by
gleaming sofas made of solid gold. Paths lead in all directions.",
        "north": "east_gate",
        "east": "staff_room",
        "west": "corridor",
        "south": "hall"
    },
    "east_gate": {
        "description": "You are standing outside the main door to the school. Your escape
to the road is blocked by a deep muddy trench from The Great War. Beyond the trench you
can see barbed wire across no-man's land. The door to the school is south.",
        "south": "the_office"
    },
    "staff_room": {
        "description": "You are in the staff room. There is a strong smell of garlic,
socks and chocolate digestives. The only door leads west.",
        "west": "the_office"
    },
    "hall": {
        "description": "You are in the Great Viking Hall of Stamford Green. Long oak
tables with silver goblets are lie across the room ready for a banquet, and Valkyries
soar overhead. A sign on the wall says that the value of the month is 'Pillaging'. The
office is north.",
        "north": "the_office"
    },
    "corridor": {
        "description": "You are in a corridor leading towards the West Wing. Cones,
sirens, flashing lights and a 'DO NOT ENTER' sign suggest that construction is not quite
finished. To the west, where the building should be, there is a currently a deep hole in
the ground. You cannot see the bottom of the pit. A path east leads back to the office.",
        "east": "the_office",
        "west": "tunnel"
    },
    "tunnel": {
        "description": "You are in a tunnel at the bottom of a pit, with a dark passage
to the north, and a light to the east. Scratched on the wall are the cryptic letters
'AshLEy wiL nvR WIN'.",
        "east": "corridor",
        "north": "maze1"
    },
    "maze1": {
        "description": "You are in an underground maze of twisting passages, all alike.",
        "east": "tunnel",
        "west": "maze2",
        "north": "maze1",
        "south": "maze2"
    },
    "maze2": {
        "description": "You are in an underground maze of twisting passages, all alike.
You can feel a warm gentle breeze.",
        "east": "maze1",
        "west": "maze2",
        "north": "maze1",
        "south": "escape"
    },
    "escape": {
        "description": "You emerge into daylight at the top field beside a running track.
The West Gate is open."
    }
}

def look():
```

```
    formatted_description=textwrap.fill(school_map[location]["description"])
    print(formatted_description)

def go(direction):
    next_location=school_map[location].get(direction)
    if next_location == None:
        print("You can't go that way.")
    return next_location

def help():
    print("Escape From Stamford Green!")
    print("--------------------------")
    print("Instructions:")
    print("1. Use 'north', 'east', 'south' or 'west' (or 'n', 'e', 's' or 'w') to move.")
    print("2. Type 'look' to see what you can see.")
    print("3. Display this message again by typing 'help'.")

# The main part of the program starts here
help()
print()

location="the_office" # Global variable containing the player's current location
look()

while location != "escape":

    print()
    command=input("> ").lower()

    move_to_location=None
    if command=="north" or command=='n':
        move_to_location=go("north")
    elif command=="south" or command=='s':
        move_to_location=go("south")
    elif command=="east" or command=='e':
        move_to_location=go("east")
    elif command=="west" or command=='w':
        move_to_location=go("west")
    elif command=="look":
        look()
    elif command=="help":
        help()
    else:
        print("I don't understand that! Try 'north', 'south', 'east' or 'west', or
'help'.")

    if move_to_location != None:
        location=move_to_location
        look()

print()
print("Congratulations, you have escaped from Stamford Green!")
```