3_python

October 2, 2023

1 Plan

1.1 Recap

- Variables
- Comparison and control flow
- Lists
- Loops
- Methods

1.2 Today

- Understanding methods/functions and how they work.
- Using what was covered last week.
- Learning how to problem solve and read errors.

2 Functions

Functions are like tools designed to do a specific task. We've already used some built in functions like `print`, `len` and `type`. We also used functions when we used string methods like `.upper`., `.replace` and list methods like `.append`.

We can make custom functions to do specific jobs too. - Custom functions avoids having to repeat code. - Building them helps us better understand the how to use the methods others have built for us.

Functions are initialised using `def` (short for define).

[ ]:
```
def name_of_function(argument1, argument2, argument3): # You can have as many␣
↪arguments as you need.

    # Code here which takes the arguments and performs some activity

    return # The result of the function should be 'returned' to the code that␣
↪called it
```

[ ]:
```
def make_awesome(word):
    new_word = 'Awesome ' + word
    return new_word
```

[ ]:
```
make_awesome('pig')
```

[ ]:
```
'Awesome pig'
```

2.1 Multiple Arguments

Functions can take more than one argument. For example when we used the string method `.replace` it took two arguments, the text we wanted to match, and the value we wanted to replace it with.

[ ]:
```
'Hello my name is James'.replace('James', 'Fred')
```

[ ]:
```
'Hello my name is Fred'
```

[ ]:
```
def make_awesome(word, n_awesome):
    new_word = 'Awesome '*n_awesome + word
    return new_word
```

[ ]:
```
make_awesome('pig', 2)
```

[ ]:
```
'Awesome Awesome pig'
```

2.2 Optional Arguments

Some things we pass to functions or methods are necessary information for the function to do its job.

Other arguments are optional, and may change the behaviour of the function. Often the functions we will use will have one or two required arguments, and then a large list of optional arguments.

At the moment our function has two required arguments. `word` and `n_awesome`. Let's make `n_awesome` optional by giving it a default value that it uses if no value is given.

[ ]:
```
def make_awesome(word, n_awesome=1):
    new_word = 'Awesome '*n_awesome + word
    return new_word

make_awesome('cat')
```

[ ]:
```
'Awesome cat'
```

[ ]:
```
make_awesome('cat', n_awesome=3)
```

[ ]:
```
'Awesome Awesome Awesome cat'
```

We could add another optional argument that changes the behaviour of the function.

[ ]:
```
def make_awesome(word, n_awesome=1, make_upper=False):
    new_word = 'Awesome '*n_awesome + word
    if make_upper:
```

```
        new_word = new_word.upper()
    return new_word

make_awesome('cat', make_upper=True)
```

[ ]:
```
'AWESOME CAT'
```

3 Documentation

How do we know what arguments to pass, and what kind of options we have? Documentation should explain what a function/method does, and tell you what arguments it takes. We can access documentation multiple ways. - Documentation for built in Python and widely used libraries is usually just a Google search away. For example `"python replace documentation"`. - In Jupyter we can also use the `?` symbol to quickly access specific documentation.

[ ]:
```
print?
```

```
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

We'll be accessing documentation more when we move onto the Pandas library next week.

4 Exercises 1

Take a look at section 1 of the exercises sheet. Complete the tasks before moving on.

5

6 Solving Problems

Documentation is a key tool in helping you solve problems, but the best tool is having the right attitude and approach.

6.0.1 Remember:

- Nobody knows how to code 'naturally'
- It is a skill to be learned
- Errors and mistakes are a sign you are learning, not that you're failing.

6.0.2 This means you should:

- Embrace mistakes
- Be willing to try things out
- Search for solutions

Coding is not about memorising the correct answer, it is about knowing how to find the answers yourself.

7 Tracebacks

Traceback messages are reports that Python provides if something goes wrong. The report tries to tell you where the error occurred, and what the likely cause was.

Traceback should be read from the bottom up as often the most important and specific information is shown last.

[ ]:
```
'The meaning of life is ' + 42
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-32-09889c4fe956> in <module>
----> 1 'The meanining of life is ' + 42

TypeError: can only concatenate str (not "int") to str
```

In this message, reading from the bottom up, we can see that the type of error thrown is a `TypeError`. This means that the wrong type of object has been used in an operation.

The Traceback also gives us an explanation if it can. Here it tells us that you can only concatenate string objects together, not a string and an integer object.

Above the last line, the Traceback has attempted to tell us where in the code the error was so that you can more easily find the part that needs correcting.

7.1 Exceptions

`TypeError` is a type of 'exception', and one of the more common built-in error types in Python. Other common types include: - `NameError` - Indicates a Python can't find a variable you're referring to. - `SyntaxError` - Something is wrong with the grammar, such as a comma or colon is missing. - `ValueError` - The wrong type of value has been given, such as a number when it expects a string.

There are many more types of Exception, some of which are specific to particular tools and packages.

7.2 Exceptions workbook

Open up the 3_exceptions_workbook.ipynb downloaded from Moodle. Run each cell, examine the Traceback message and see if you can fix the code so that it runs. Use Google and other resources to help you.