1. **Documentation Strings:**
   1. Python supports an additional comment form called documentation strings.
   2. Docstrings are simply strings that show up at the top of program files and some statements. Python ignores their contents, but they are automatically attached to objects at runtime and may be displayed with documentation tools like PyDoc. Docstrings are part of Python’s larger documentation strategy.
   3. \_\_doc\_\_ attribute
   4. dir()
   5. help()
2. This captures the essence of programming for me - “we do things with stuff”!
   1. “Things” take the form of operations like addition and concatenation, and “stuff” refers to the objects on which we perform those operations.
   2. Everything is an object in a Python script. Objects are like alphabets for Python.
   3. More formally, objects are essentially just pieces of memory, with values and sets of associated operations.
   4. e.g. a = 90 creates an object ‘a’ as type Integer, value ‘90’ and supported operations of addition, subtraction etc.
3. Hierarchy of Python:
   1. Python programs can be decomposed into modules, statements, expressions, and objects, as follows:
   2. Programs are composed of modules.
   3. Modules contain statements.
   4. Statements contain expressions.
   5. Expressions create and process objects.
4. Built-in Object types:

| Object type | Example literals/creation |
| --- | --- |
| Numbers | 1234, 3.1415, 3+4j, 0b111, Decimal(), Fraction() |
| Strings | 'spam', "Bob's", b'a\x01c', u'sp\xc4m' |
| Lists | [1, [2, 'three'], 4.5],list(range(10)) |
| Dictionaries | {'food': 'spam', 'taste': 'yum'},dict(hours=10) |
| Tuples | (1, 'spam', 4, 'U'),tuple('spam'),namedtuple |
| Files | open('eggs.txt'),open(r'C:\ham.bin', 'wb') |
| Sets | set('abc'),{'a', 'b', 'c'} |
| Other core types | Booleans, types, None |
| Program unit types | Functions, modules, classes |
| Implementation-related types | Compiled code, stack tracebacks |

**Python REPL**

1. Open Windows Powershell/Ubuntu Terminal.
2. Type python to enter python shell.
3. Useful to test out code snippets
4. **REPL = Read Evaluate Print Loop** - helps interact with Python code
   1. A simple interactive loop

while True:

reply = input('Enter text:')

if reply == 'stop':

break

print(reply.upper())

* 1. This small piece of code is quite powerful and has many things going on.
     1. while loop: Python’s most general looping statement
        1. it consists of the word while, followed by an expression that is interpreted as a true or false result, followed by a nested block of code that is repeated while the test at the top is true (the word True here is considered always true).
        2. while test: # Loop test

statements # Loop body

else: # Optional else

statements # Run if didn't exit loop with break

* + 1. input: built-in function for taking user input from console
    2. if statement: tests the condition following if keyword which is evaluated to True or False. If evaluated to True, the code in the block beneath the if statement is executed. If statement consists of the word if followed by a test and an associated block of code, one or more optional elif (“else if”) tests and code blocks, and an optional else part, with an associated block of code at the bottom to serve as a default. Python runs the block of code associated with the first test that is true, working from top to bottom, or the else part if all tests are false.
    3. break statement: break and exit the loop
    4. print: output the content following the print statement
  1. In effect, this combination of statements essentially means:
     1. Read a line from the user
     2. Print it in uppercase until the user enters the word ‘stop’.