Core Elements

PART II
ITERATIONS
&
FUNCTIONS

Today's Objectives

- Introduction to Objects
- Maintainable code
- Importing libraries
- Iteration: **for** loops
- Functions: **def**
  - parameters

What are Objects?

- Objects are composed of structural and behavioural constituents.
- Data field enable an object to maintain state.
  - a.k.a *properties*, *fields*, *data members*, or ***attributes***
- **Methods**, enable the behaviour of Objects.

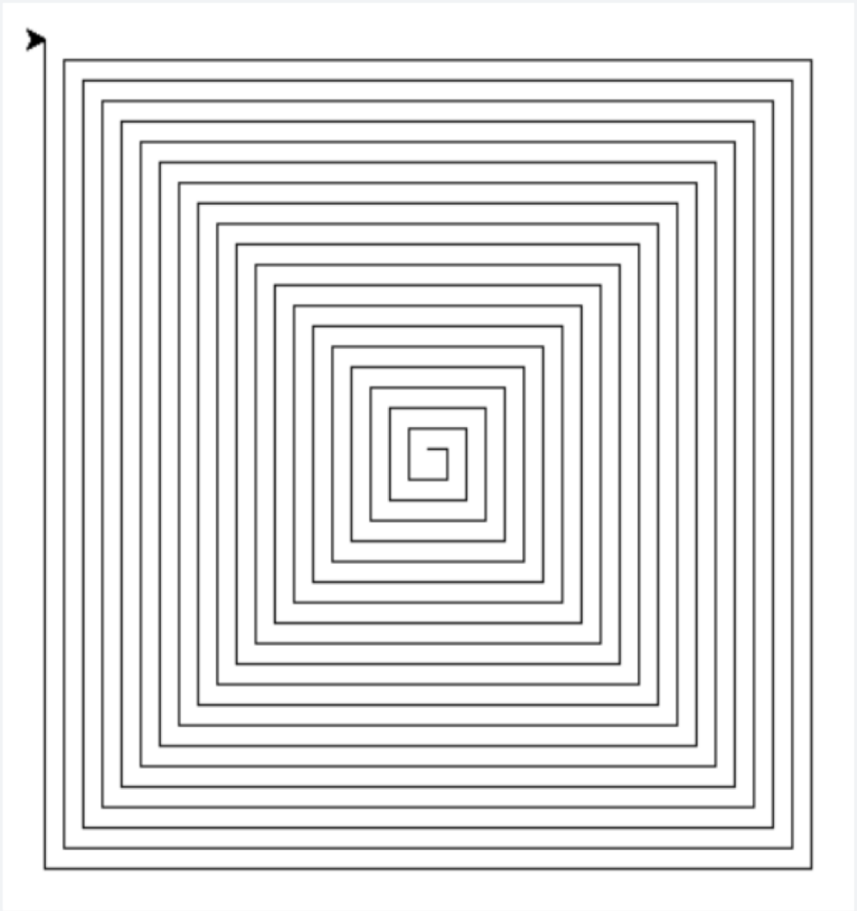
An example of object

- Code Demo:

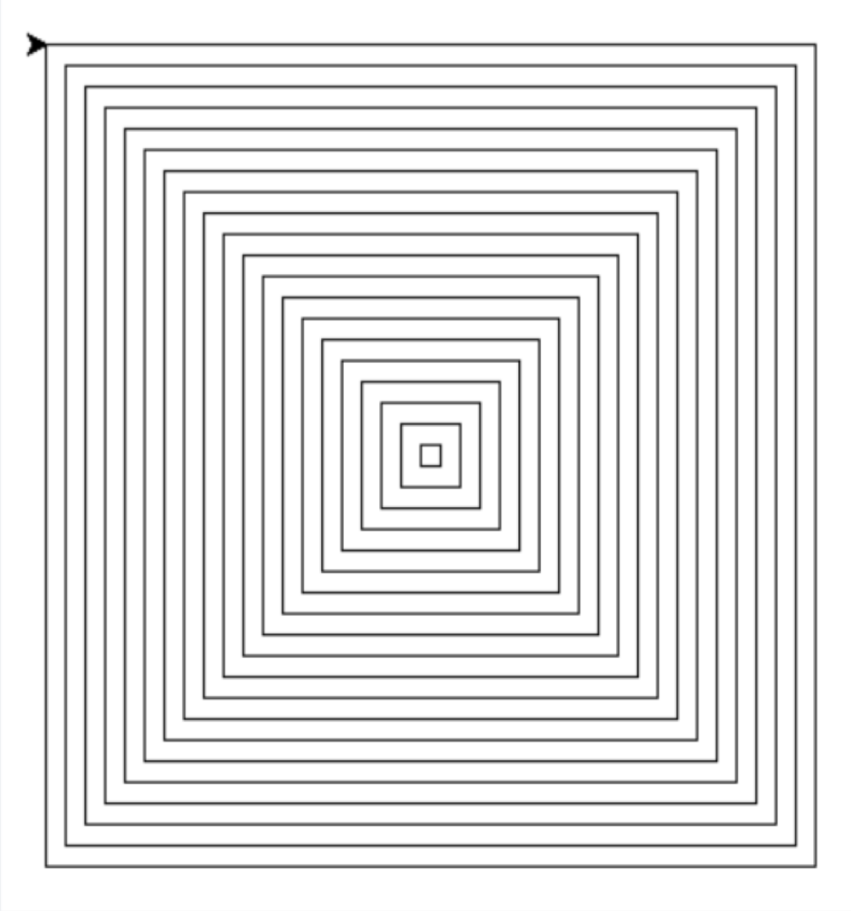
```
>>>
>>> a_string = 'a short sentence.'
>>> type(a_string)
<type 'str'>
>>> len(a_string)
17
>>> a_string.upper()
'A SHORT SENTENCE.'
>>>
```

The Programs Outputs

Program 1



Program 2



First Attempt at Program 1

- Code sample 1

Iteration

7

DEFINITE LOOPS

Iteration

- We need a structure to execute a sequence of statements multiple times in succession
- So Far, we have to write the sequence of statement *n* times to achieve *n* repetition
- How can we proceed if we are not sure how many times it needs to be repeated?

Definite Loop: For statement

- The simplest kind of loop
- It will execute a definite amount of times
- The for loop statement

```
for <var> in <iterable>:
    <body>
```

  - Iterable can return its element one at a time
  - The body of the loop can be any sequence of Python statements
  - The variable after the keyword `for` is called the ***loop index***.

Example

Code

```
print "Before the for loop"

for val in [1,2,3]:
    square_val = val * val
    print "--> inside loop: square of", str(val), "is",
          str(square_val)

print "After the for loop"
```

Python shell

```
Before the for loop
--> inside loop: square of 1 is 1
--> inside loop: square of 2 is 4
--> inside loop: square of 3 is 9
After the for loop
```

Modularisation

- Monolithic Programs cannot be maintained, nor reused easily
- Decomposition of complex problem into smaller sub-problem
- Modularisation
  - Separation of concerns
  - Semantically coherent

Why Function?

- having similar code in two places has some drawbacks
- failing to keep related part of the code in sync is a common problem in program maintenance
- function can be used to reduce code duplication

Why Function?

- make program more understandable
- easier to maintain
- easier to reuse

In Summary

- Advantages
  - code reuse
  - facilitate team work
  - modularisation
  - maintainability

- monolithic code
  - huge collection of statement
    - no modularisation
    - no code reuse (cut & paste is not code reuse!)
    - no parallel implementation

Functions

- New Keywords
  - def

- Function declaration and Function calls

Declaring a Function

- template:

```
def <function_name> (<formal_parameters>):
    <body>
```

  - `<function_name>`: an identifier
  - `<formal_parameters>`: comma-separated identifiers
  - `<body>`: any number of indented statements.

Summary

- Introduction to Objects
- Maintainable code
  - Importing libraries
    - how could we import mathematics tools?
    - Randomness?
  - Using iterations and functions to modularise code
  - Separation of concerns
- Iterations and Functions will be revisited next week
  - There is much more to what we have seen

Reading

- Python Code Style, Comments and DocString
  - http://www.python.org/dev/peps/pep-0008/
  - http://www.python.org/dev/peps/pep-0257/
- Must be applied to your code!!!

Exercises

- Use what we have seen today to draw the following image: