We organize computations to be reused.

1. Questions.

2. We can gather commands into a file or script. Here, mage.py:

```python
# A script to compute the phase of the moon.
# Typical usage: python3 phoon.py <month> <day> <year>
from sys import argv
month = int(argv[1])
day = int(argv[2])
year = int(argv[3])
assert (1900 <= year <= 2099), "Year not in range"
# Step i. start a sum
s = day+month+30
# Step ii. correct for century
cent = year//100
yy = year % 100
s = s - (8 if cent == 20 else 4)
# Steps iii & iv. correct for year within century
# modified from the original
d = yy%19
if d > 9:
    d = d-19 # e.g. 2017 yields 17-19 == -2
# absolute value of d
ad = d if d >= 0 else -d
adMod3 = ad%3
if d < 0:
    d = d-10*adMod3
else:
    d = d+10*adMod3
# Step v. compute final age
s = (s + d)%30
# The result: the age of the moon. Print it.
print("Age of the moon is: {}".format(s))
```

To execute this script, we can type:

```
python3 mage.py 2 8 2017
```

We can now reuse the code in the script, without re-typing the commands.

3. New dataset: grades15-16.dat in the datasets folder
4. Functions (sometimes *procedures* or *subroutines*). Introduced with the keyword *def*.

(a) We use functions to gather together statements that perform (at some level) a single, logical action. Some functions produce results. Others produce side-effects. This encapsulation allows us, again, to reuse statements whenever we wish.

(b) Functions are *parameterized by arguments* or *parameters*. For example, the trigonometry `math.sin` function takes an angle. The `print` takes an object. Within the function, expressions refer to these arguments as *formal parameters*. When calling the function, the values we actually use are called *actual parameters*.

5. Example: A function to detect odd integers.

6. Example: A simple, but strange function, `syr`.

7. Example: Printing the orbit of `syr`.

8. Example: The length of the orbit of `syr`.

9. Example: Sorting a string.

10. Example: Getting lines or words in a file.

⋆