Eric Roberts
CSCI 121

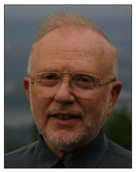
Handout #9
September 12, 2018

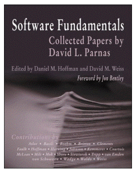
Decomposition

Decomposition

Eric Roberts
CSCS 121
September 12, 2018

David Parnas





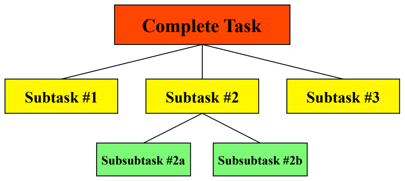
David Parnas is Professor Emeritus of Computer Science at Limerick University in Ireland, where he was director of the Software Quality Research Laboratory, and has also taught at universities in Germany, Canada, and the United States.

Parnas's most influential contribution to software engineering is his groundbreaking 1972 paper "On the criteria to be used in decomposing systems into modules," which laid the foundation for modern structured programming. This paper appears in many anthologies and is available on the web at

`http://portal.acm.org/citation.cfm?id=361623`

Stepwise Refinement

- The most effective way to solve a complex problem is to break it down into successively simpler subproblems.
- You start by breaking the whole task down into simpler parts.
- Some of those tasks may themselves need subdivision.
- This process is called ***stepwise refinement*** or ***decomposition***.



Criteria for Choosing a Decomposition

1. ***The proposed steps should be easy to explain.*** One indication that you have succeeded is being able to find simple names.
2. ***The steps should be as general as possible.*** Programming tools get reused all the time. If your methods perform general tasks, they are much easier to reuse.
3. ***The steps should make sense at the level of abstraction at which they are used.*** If you have a method that does the right job but whose name doesn't make sense in the context of the problem, it is probably worth defining a new method that calls the old one.

Passing Information between Functions

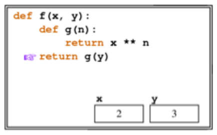
- There are several strategies you can use to pass information from one function to the functions that it calls. These include
  - ***Defining constants.*** Constants defined at the top level of a module provide a convenient way to share information that is likely to remain constant for the lifetime of a program.
  - ***Passing arguments.*** Arguments are the best mechanism for passing information that is likely to change from call to call.
  - ***Inheriting variables from enclosing functions.*** As described on the next slide, functions in Python can be ***nested***, which means that one function is defined inside another. The interior functions have access to the local variables in the enclosing function. A common example of this model is the `gw` variable.
- There is no hard-and-fast rule for deciding which technique is best. A single application will often use all three depending on how the information is used.

Nested Functions

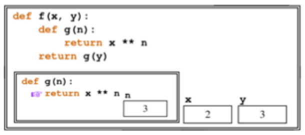
- Python allows you define one function inside another, as in the following example from the text:

```
def f(x, y):
    def g(n):
        return x ** n
    return g(y)
```

- Calling `f(2, 3)` creates the following frames:



```
def f(x, y):
    def g(n):
        return x ** n
    return g(y)
```

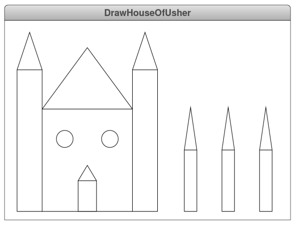


```
def f(x, y):
    def g(n):
        return x ** n
    return g(y)
```

```
def g(n):
    return x ** n
```

Exercise: The House of Usher

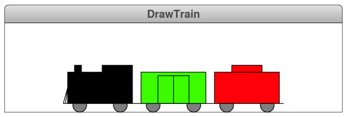
- Exercise 5 on page 108 of the text asks you to draw a stylized rendering of Edgar Allan Poe's House of Usher:



- For the next few minutes, think about how you would divide this problem into smaller parts without writing any code.

The `DrawTrain` Program

- The primary example of decomposition in Chapter 4 is the `DrawTrain` program, which asks you to draw trains that look like this:



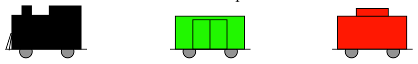
Parameters for Drawing Train Cars

- The `DrawTrain` program in the text makes the following assumptions:
  - The caller will always want to supply the location of each car.
  - All train cars are the same size and have the same structure.
  - Engines are always black.
  - Boxcars come in many colors.
  - Cabooses are always red.
- These assumptions imply that the headers for `drawEngine`, `drawBoxcar`, and `drawCaboose` will look like this:

```
def drawEngine(x, y):
def drawBoxcar(x, y, color):
def drawCaboose(x, y):
```

Looking for Common Features

- Another useful strategy in choosing a decomposition is to look for features that are shared among several different parts of a program. Such common features can be implemented by a single function.
- In the `DrawTrain` program, every train car has a common structure that consists of the frame for the car, the wheels on which it runs, and a connector to link it to its neighbor.
  - The engine adds a smokestack, cab, and cowcatcher.
  - The boxcar is colored as specified by the caller and adds doors.
  - The caboose is red and adds a cupola.



- You can use a single `drawCarFrame` function to draw the common parts of each car, as described in the text.