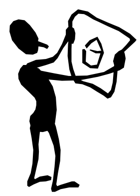
Divide, conquer, glue

Loops, structures, program design



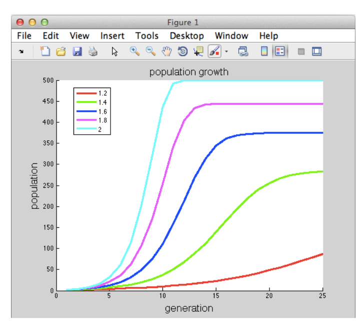
CS112 Scientific Computation
Department of Computer Science
Wellesley College

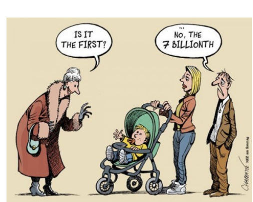
Simulating population growth

Goal: define a function that generates a figure with curves for different rates of population growth over multiple generations, using the *logistic growth* model for population growth:

$$p_{t+1} = r * p_t * (K - p_t)/K$$

p_t: current population
p_{t+1}: population in the next generation
r: growth rate
K: carrying capacity





Guidelines & tips

Define a function named popGrowth with four inputs:

- vector of growth rates to simulate
    - (default [1.2 1.4 1.6 1.8 2.0])
- initial population (default 2)
- number of generations (default 25)
- carrying capacity (default 1000)

For each growth rate:

- create a vector to store the population for each generation, and store initial population in the first location of the vector
- for each new generation, apply the formula to calculate the new population size and store it in the vector
- plot the populations for this growth rate

Add figure embellishments at the end

3

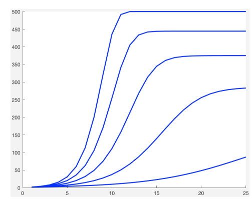
```
function  popGrowth (rates,  generations,  initPop,  K)
% all input parameters are optional
```

```
if  (nargin < 4)
    K = 1000;
end
if  (nargin < 3)
    initPop = 2;
end
if  (nargin < 2)
    generations = 25;
end
if  (nargin < 1)
    rates = [1.2  1.4  1.6  1.8  2.0];
end
. . .
```

```
if  (nargin == 3)
    K = 1000;
elseif  (nargin == 2)
    initPop = 2;
    K = 1000;
elseif  (nargin == 1)
    generations = 25;
    initPop = 2;
    K = 1000;
elseif  (nargin == 0)
    rates = [1.2  1.4  1.6  1.8  2.0];
    generations = 25;
    initPop = 2;
    K = 1000;
end
. . .
```

4

```
% for each growth rate
for rate = rates
    % create a vector to store population for each generation
    pops = zeros(1, generations);
    % store initial population in the first location of vector
    pops(1) = initPop;
    % for each new generation
    for gen = 2:generations
        % apply formula to calculate new population size ...
        %  ... and store it in the vector
        pops(gen) = rate * pops(gen–1) * (K – pops(gen–1))/K;
    end
    % plot the populations for this growth rate
    plot(pops, 'b')
end
```



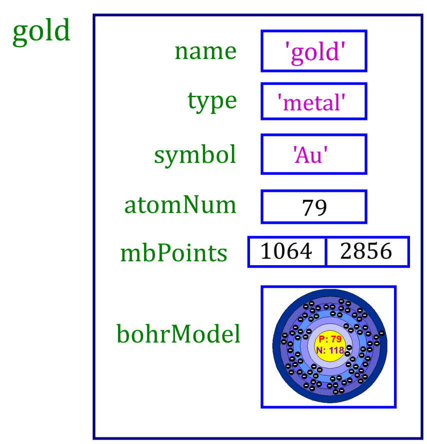
5

Structures

A structure can store multiple values of different types

```
gold.name = 'gold';
gold.type = 'metal';
gold.symbol = 'Au';
gold.atomNum = 79;
gold.mbPoints = [1064  2856];
gold.bohrmodel = goldPict;
```

structure name | field name | field value



6

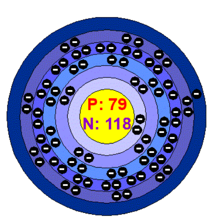
Structures make sharing easy

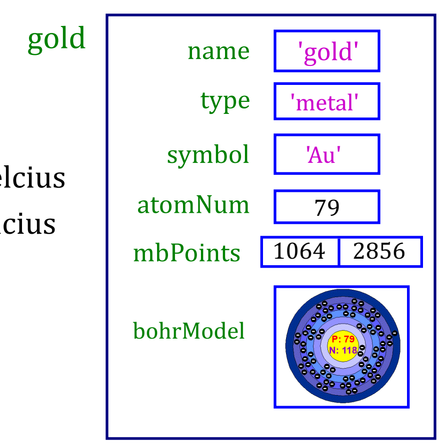
```
function describeElement (element)
% shows properties stored in the input element structure

disp(['name of element: '  element.name]);
disp(['type of element: '  element.type]);
disp(['atomic symbol: '  element.symbol]);
disp(['atomic number: '  num2str(element.atomNum)]);
disp(['melting point: '  num2str(element.mbPoints(1)) ...
        ' degrees Celcius' ]);
disp(['boiling point: '  num2str(element.mbPoints(2)) ...
        '  degrees Celcius']);
imshow(element.bohrModel);
```


Sharing structures

```
>> describeElement(gold)
name of element: gold
type of element: metal
atomic symbol: Au
atomic number: 79
melting point: 1064 degrees Celcius
boiling point: 2856 degrees Celcius
```





Play it again, Sam…

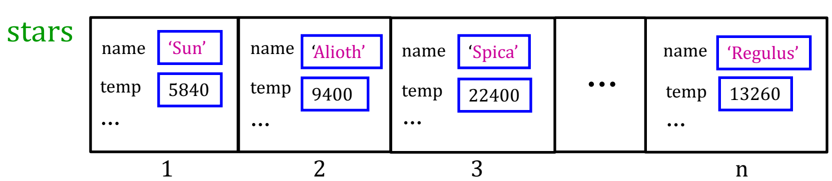

```
for i = 1:100            % for loop
   disp('Play it once, Sam, for old times'' sake');
   again = input('Play it again? (yes:1, no:0) ');
   if (again == 0)
      break
   end
end
```

```
while conditional expression
      statements to repeat if conditional
          expression is true
end
```

```
again = 1;
while (again == 1)            % while loop
   disp('Play it once, Sam, for old times'' sake');
   again = input('Play it again? yes(1) or no(0): ');
end
```

9

Vector of structures



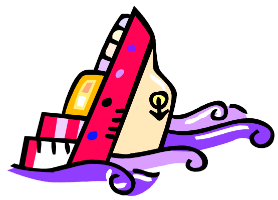

```
function printTemps (allStars)
% print temperature of all the stars
for i = 1:length(allStars)
    disp([allStars(i).name ' ' num2str(allStars(i).temp)])
end
```

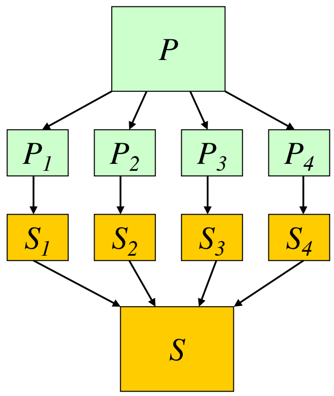
```
function temp = getTemp (allStars, starName)
% return temperature of input star
i = 1;
while (~strcmp(allStars(i).name, starName) & (i < length(allStars))
    i = i + 1;
end
temp = allStars(i).temp;
```

10

Program complexity

Designing large scale programs is fraught with peril





Divide, conquer & glue is a simple but powerful design strategy that helps us avoid danger

Tools of the trade

We have used functions and scripts to help divide problems into manageable chunks:

lineFit, poleVault
rotate, spin
displayGrid, virus

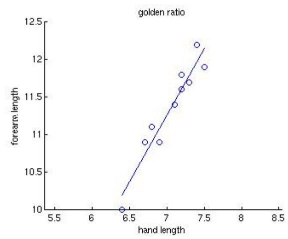
What kinds of subtasks are performed by these individual functions in these programs, and ...

... why did we divide the programming task in this way?

Functions may…

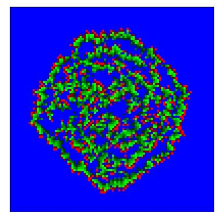
Perform a general function that is useful in many contexts

- lineFit function can be used for any linear regression
- visualize displays many kinds of data



Apply or test other functions

- poleVault tests the lineFit function



Hide details of tasks like plotting or displaying data

- displayGrid displays current state of the virus

13

Functions help to avoid repetitious code

Consider a function with the following structure

```
function outputs = myFunction (inputs)
  statements a
  statements b
  statements c
  statements b
  statements d
  statements b
```

similar statements

Encapsulate repetitious statements in a separate function

14

Test, test, test!

"If there is no way to check the output of your program, in using that program, you have left the realm of scientific computation and entered that of mysticism, numerology, and the occult."

Daniel Kaplan, Introduction to Scientific Computation & Programming

Tips on testing:

- Test & debug each function on its own
- Create test data for simple cases where expected intermediate results and final answer can be easily verified
- Be thorough! Construct examples to test all cases considered by program

15

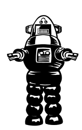
Functions versus scripts

Functions usually have one or more inputs that provide data or control aspects, and one or more outputs

Scripts perform a specific set of actions and do not have inputs or outputs

Execution of a ***function*** creates a private, temporary environment of variables

Scripts have access to variables defined in the environment within which the script is called*

*  Danger Will Robinson!!!

16

Subfunctions

An M-file can only contain one function that can be called from the Command Window or from another code file

This function must be placed at the beginning of the file and its name must be the same as the file name

Other *subfunctions* can be defined in an M-File, but can only be called by functions in the same M-File

17

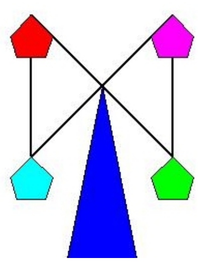
Subfunctions for a ferris wheel movie

```
function ferrisWheel
% displays an animation of a rotating ferris wheel
for frame = 1:36
    drawBase;
    hold on
    spokeCoords = drawWheel(10*frame);
    drawCars(spokeCoords);
    pause(0.1), hold off
end

function drawBase
% draw the blue base of the ferris wheel

function spokeCoords = drawWheel (angle)
% draw the black spokes at the input angle and return
% the coordinates of the endpoints of the spokes

function drawCars (spokeCoords)
% draw a colored car at each location in spokeCoords
```



18