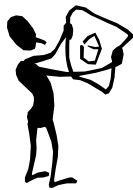


Divide, conquer, glue

Program design



CS112 Scientific Computation

Department of Computer Science
Wellesley College

Play it again, Sam...

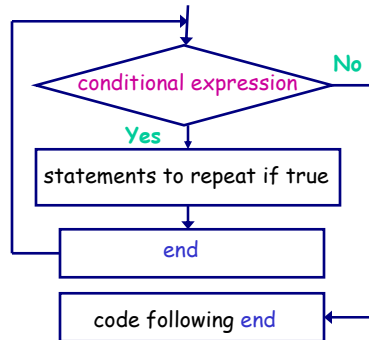
```
for i = 1:100           % bit of a kludge...
    disp('Play it once, Sam, for old times'' sake');
    again = input('Play it again? (yes:1, no:0) ');
    if ~again
        break
    end
end

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



The while statement

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



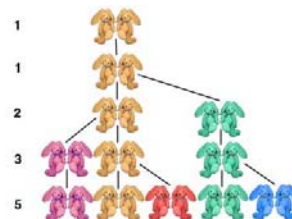
3

Fibonacci numbers, or multiplying rabbits?

Fibonacci numbers first appear around 500 B.C. in writings of a Sanscrit grammarian named Pingala who studied rhythm in language



Leonardo Fibonacci studied these numbers around 1200 in the context of multiplying rabbits



- In the first month, there's one newborn pair
- Newborn pairs become fertile in their second month
- Each month, every fertile pair begets a new pair
- Rabbits never die

1 1 2 3 5 8 13 21 34 ...

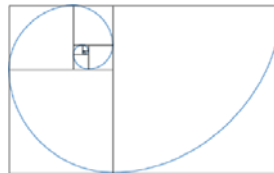
4

Finding first Fibonacci number > 100

```
fibo = [1 1];  
while (fibo(end) < 100)  
    fibo(end+1) = fibo(end) + fibo(end-1);  
end  
disp(['first Fibonacci number > 100: ' num2str(fibo(end))]);
```



Bet none of this lot know what 1, 1, 2, 3, 5, 8, 13 has in common with the snail I've got in my pocket!



5

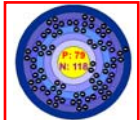
Structures

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

gold

name	'gold'
type	'metal'
symbol	'Au'
atomNum	79
mbPoints	1064 2856
bohrModel	

6

Structures make sharing easy

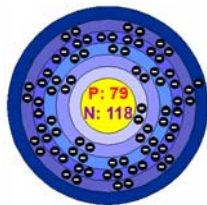
```
function describeElement (element)
% shows the properties stored in the input element structure

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);
```

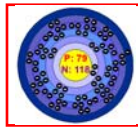
7

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
```



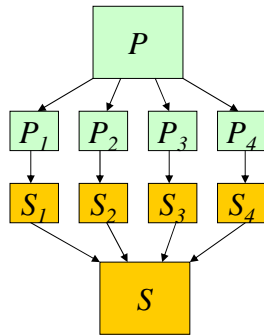
gold

name	'gold'
type	'metal'
symbol	'Au'
atomNum	79
mbPoints	1064 2856
bohrModel	

8

Program complexity

Designing large scale programs
is fraught with peril



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

9

Tools of the trade

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

`makeDots`, `illusions`
`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?

10

Our goal is ...

... to design programs that:

- are free of errors
- run efficiently
- require no more memory than necessary
- are easy to understand and use
- can be used in a variety of situations
- are easy to maintain and modify if necessary



*... and to do all of this within time and budget

11

Functions may...

Perform a general function that is useful in many contexts

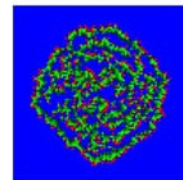
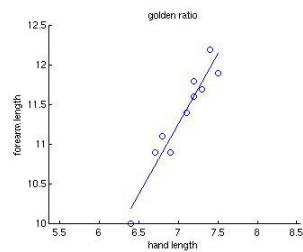
e.g. `lineFit` function can be used for any linear regression

Apply or test other functions

e.g. `poleVault` tests the `lineFit` function

Hide details of tasks like plotting or displaying data

e.g. `displayGrid` displays the current state of the virus



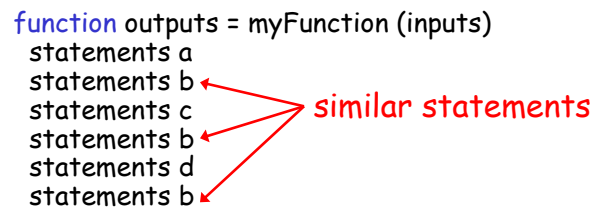
12

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

A list of code statements is shown: 'statements a', 'statements b', 'statements c', 'statements b', 'statements d', and 'statements b'. Red arrows point from each of the four 'statements b' lines to a red label 'similar statements' on the right.

Encapsulate repetitious statements in a separate function

13

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 and Programming



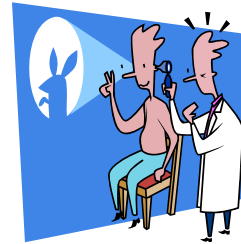
14

General tips on testing

Test and 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 your program



15

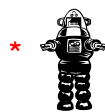
Functions versus scripts

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