[QI;MP]

Basic Numerical Structures

Lee James O'Riordan

OIST

Programming fundamentals

[QI;MP]

- Compiled languages: (Fortran, C/C++, Objective-C, Java)
  - Write code, compile, wait… run fast!
  - Long development time, but best performance.
- Interpreted languages: (Python, Javascript, Perl)
  - Write code, run!
  - Shorter development time, but not as fast as compiled languages.

Algorithm development

[QI;MP]

- Write down everything you know about the problem!
- Don't try to reinvent the wheel.
  - Unless you think you can do better --- but be honest!
- "Premature optimisation is the root of all evil", D. Knuth.
  - i.e. get it to work first, then worry about making it work better. Fast garbage is still garbage!
- Everything that you can do in software can be done in hardware (and vice versa)

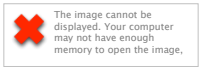
MATLAB

[QI;MP]

- High level interpreted language (but can make use of compiled code)
- Written in C/C++, Java, and MATLAB
- Numerical computing and prototyping language
- Many (more than we can discuss today!) toolboxes and user-submitted codes for all types of purposes

Number formats

[QI;MP]

- Integers: (int8…int64, uint8…uint64)
  - Exact precision in integer operations --- no round-off error.
  - Must explicitly specify: int16(x)
  - Most likely you won't need this!
- Double: (default number representation)
  - MAX≈10^{308}, MIN≈10^{-308}, ε≈10^{-16}
- Complex double: (default complex number type)
  - As above, but with format 

Syntax, notation & operators

[QI;MP]

- For computer to understand you, must format question in language it can understand.
  - Infix notation: Operand Operator Operand (like a calculator!)
- Assignment: (=) Arithmetic: (^ * / + -)

  e.g. `a=3+5;` `b=6*4;` `c=a/b;` `c^3;`
- Relational: Equivalent (==), Not equivalent (~=),

  LT (<), GT (>), LTE (<=), GTE (>=)

  e.g. `a <= b;` `e = c > d;` `f ~= 3`
- Logical: AND (&&) OR (||) NOT (~)

```
e.g.isItTrue = (2 < 3) && (4 > 3);
    isItTrue2 = (a==2) || (b~=4);
```

Syntax, notation & operators

[QI;MP]

- Scalars, vectors, and matrices

```
s = 2;    vec_row=[1,2,3]; vec_col=[1;2;3]

%Comments are written like this

%You can transpose col to row like this:

vec_col=vec_col';

mat=[[1,2,3];[4,5,6];[7,8,9]]
```

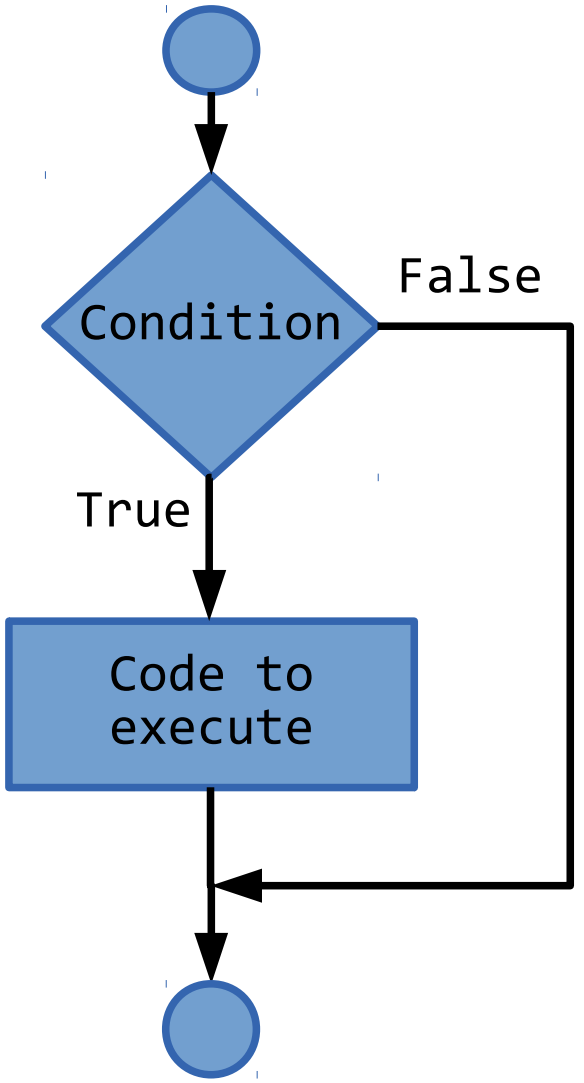
- Pointwise (element-wise):
  e.g. `A.*B  D./F  Z.^2`
- Indexing: `A(i,j)`

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

Conditionals

[QI;MP]

- Allow choices to be made in program
  - if…elseif…else:
    - 'if' statement will execute following code provided condition is true, otherwise skips
    - 'elseif' is checked when 'if' fails. Can be used multiple times for different values.
    - 'else' is the catch-all for unspecified conditions.
  - switch case:
    - Allows condition to be compared to a known set of values.



Example 1 - Conditionals

[QI;MP]

- You have 1000 Yen. You want to go to Naha. The 120 bus costs 800 Yen, while limousine bus costs 2000 Yen.

```
>> money = 1200;

>> if money >= 2000
       disp('limousine bus!');
     elseif (money >= 800) && (money <= 2000)
       disp('120 bus!');
     else
      disp('Start walking!')
     end
```

Example 1 - Conditionals

[QI;MP]

- You are ordering a meal, and are trying to decide which portion size to go for:

```
>> hungry = ‘very’;

>> switch hungry
            case ‘not very’
            disp(‘Regular burger’)
       case ‘a little’
            disp(‘Yokubari burger’)
       case {‘very’, ‘super’, ‘mondo-extremely’}
            disp(‘Heart-attack murder burger’)
       otherwise
            disp(‘Go to FamilyMart!’)
      end
```

Loops

[QI;MP]

- For loop:
  - Loops over a specified range of values (walk from here until 5km, and drink water every 1km).

```
>> for i = start : increment : end    %[1,2,3,4,5]
     disp(i);
   end
```

- While loop:
  - Loop continues while the specified condition is true (walk until you are tired, then stop).

```
>> while i <= 100
     disp(i);
     i=i+1;
   end
```

- To forcefully exit loop use "`break;`"

Function

[QI;MP]

- Block of code designed for a particular purpose

```
function [output_args] = myFunction (input_args)
  %Machinery of function goes here
end
```

- MATLAB functions must be saved as **.m** files.
- Single MATLAB function per **.m** file. Filename must match function name!
- Function must be in same directory (or included in MATLAB path --- same directory is easier)
- Callable from command prompt like:

```
>> function_name(args)
```

Examples 2

[QI;MP]

- Create function that calculates Fibonacci series: each respective element is the sum of the previous 2 (for-loop, function, conditionals)
- Click “New” => “Function”

```
function [result] = fibonacci(n)
%FIBONACCI calculates Fibonacci series up to the
n-th index.

result(1)=0;       result(2)=1;
for i = 3:1:n
     result(i) = result(i-1) + result(i-2)
end
```

- Save file in check current folder and type at prompt:

```
>> a=fibonacci(10); [b] = fibonacci(10);
```

Examples 2

[QI;MP]

- Create function that calculates n-factorial (n!) while loop, function, conditionals
- Click “New” => “Function”

```
function [result] = fac(n)
f = 1;
while n >= 1
      f = f* n;
      counter = counter – 1;
end
result = f;
```

- As previously, save and type:

```
>> a = fac(5);     [b] = fac(3);     fac(16);
```

Script

[QI;MP]

- A recipe for carrying out a task.
  - After defining necessary functions, most time will be spent writing scripts (until you start to execute).
- Can comprise a collection of loops, conditionals, function calls.
- Must be saved as a .m file (similar to functions)
  - Functions and scripts **are different!**
  - Cannot define a function in a script! Calling function is OK.

Examples 3

[QI;MP]

- Create a simple script using conditionals, loops and functions.
- Click “New” => “Script”

```
mat=[[1,2,3];[4,5,6];[7,8,9]]
path=1;
for i=1:1:3
      for j=1:1:3
            if(path==1)
                  disp(fibonacci(mat(i,j)))
            elseif(path==2)
                  disp(factorial(mat(i,j)))
            else
                  disp('1:Fibonacci; 2:Factorial')
            end
      end
end
```

- Questions?