Teach A level Computing: Algorithms and Data Structures

Eliot Williams

@MrEliotWilliams

Queen Mary University of London · Supported by Mayor of London · Computing at School Educate · Engage · Encourage · King's College London

Course Outline

1	Representations of data structures: Arrays, tuples, Stacks, Queues,Lists
2	Recursive Algorithms (& lists as we didn't quite get there last time…)
3	Searching and Sorting - EW will be late!
4	Hashing and Dictionaries, Graphs and Trees
5	Depth and breadth first searching ; tree traversals

Queen Mary University of London · Supported by Mayor of London · Computing at School Educate · Engage · Encourage · King's College London

Procedures

To write recursive programs we need a good understanding of procedures

Why Procedures?

- Code organisation
  - Procedures allow code to be organised in parts
  - Top-down development
- Code reuse
  - The library
  - Procedures with parameters: existing code, your data
- Recursion

Queen Mary University of London SUPPORTED BY MAYOR OF LONDON COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE KING'S College LONDON

Procedures & Functions

- Procedures:
  - 0 or more inputs
  - 0 or more outputs
  - Side effects (print statements, global variables etc)
- Functions
  - 1 or more inputs
  - 1 output
  - No side effects

Creating a procedure in python

```
def fun1(p):
    p = p + 1
    return p

a = 1
b = fun1(a)
print("argument a =", a)
print("return value b =", b)
```

Name

Parameter

Return expression

Function call

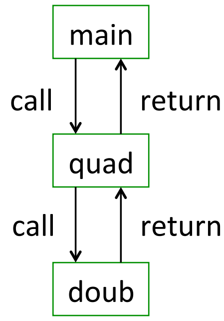
Argument

- What is the result?
- Is variable a changed?

Procedures Call Tree

- One function calls another

```
def doub(p):
    return p*2

def quad(p):
    d = doub(p)
    return doub(d)

n = int(input("Number: "))
print("4 x", n, "=", quad(n))
```



Procedures Scope

- Scope: dictionary of variables

```
def doub(p):
    return p*2

def quad(p):
    d = doub(p)
    return doub(d)

n = int(input("Number: "))
print("4 x", n, "=", quad(n))
```

p → integer (6)

p → integer (3)
d → integer (6)

n → integer (3)
doub → ... code
quad → ... code

Pass by reference or by value

```
def fun1(thelist):
    thelist.append(41)

myl = [2,3,4]
fun1(myl)
print("myl =", myl)
```

Just like assignment a list:
… variable refers to the list
… parameter refers to the list

No global: reference is read

Any mutable object will will behave this way…….

- What is the result?
- Is the list variable a changed?

big

- Define a function big that takes 2 inputs and returns the biggest!

```
def big(a,b):
    if



print(big(2,3))
print(big(4,4))
```

big

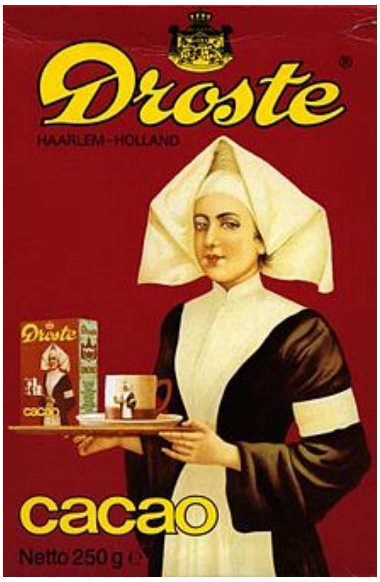
- Define a function big that takes 2 inputs and returns the biggest!

```
def big(a,b):
    if a>b:
        return a
    return b

print(big(2,3))
print(big(4,4))
```


big3

```
def big3(a,b,c):
    if a>b:
        if a>c:
            return a
        else:
            return c
    else:
        if b>c:
            return b
        else:
            return c
```

```
def big(a,b):
    if a>b:
        return a
    return b

def big3(a,b,c):
    d=big(a,b)
    return big(d,c)
```



Recursion

Recursion

Recursion

- **Recursion** in computer science is a method where the solution to a problem depends on solutions to smaller instances of the same problem (as opposed to iteration).[1] The approach can be applied to many types of problems, and recursion is one of the central ideas of computer science.[2]

Queen Mary University of London
SUPPORTED BY MAYOR OF LONDON
COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE
KING'S College LONDON

What do you need to write every possible computer program.....

- Complex question – Babbage's analytical Engine had:
  - The arithmetic functions +, −, × where − indicates "proper" subtraction $x - y = 0$ if $y \geq x$
  - Any sequence of operations is an operation
  - Iteration of an operation (repeating n times an operation P)
  - Conditional iteration (repeating n times an operation P conditional on the "success" of test T)
  - Conditional transfer (i.e. conditional "goto").

- Another answer is procedures, simple arithmetic with the comparisons, and "if" statements

Queen Mary University of London | SUPPORTED BY MAYOR OF LONDON | COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE | KING'S College LONDON

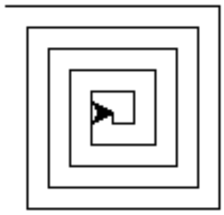
Predict & Explain what will this do

```
import turtle
myTurtle = turtle.Turtle()
myWindow = turtle.Screen()

def foo(bar):
  if bar > 0:
    myTurtle.forward(bar)
    myTurtle.right(90)
    foo(bar-5)

foo(100)
myWindow.exitonclick()
```

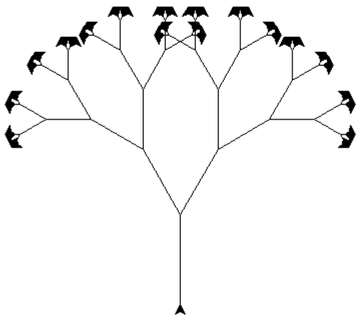
Queen Mary University of London | SUPPORTED BY MAYOR OF LONDON | COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE | KING'S College LONDON

Our first Example - Spiral

```
import turtle
myTurtle = turtle.Turtle()
myWindow = turtle.Screen()

def spiral(side):
    if side > 0:
        myTurtle.forward(side)
        myTurtle.right(90)
        spiral(side-5)

spiral(100)
myWindow.exitonclick()
```



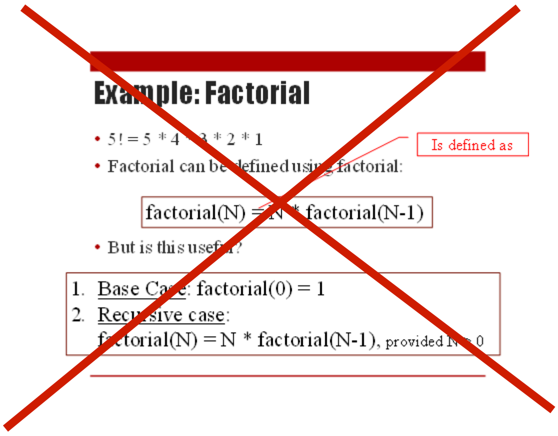
Modify



- Change the angle between the branches
- Change the thickness of the branches so that as the branchLen gets smaller, the line gets thinner.
- Change the colour of the branches so that they start brown and as the branchLen gets very short it is green.
- Change the recursive call branchLen so that instead of always subtracting the same amount you subtract a random amount in some range

Hints:
turtle.width(width)
turtle.color("blue")

Our first recursive program: Factorial



- In my opinion number sequences tend to be too complex for a first example
- Recursion is a threshold concept
  - New concept
  - Different way of thinking about repetition
  - New terminology
- We want to keep it simple at first
  - Cognitive load theory

Our first recursive program

```
def wibbler():
        print("wibble“)
        return wibbler()

wibbler()

#whats going to happen?
```

- what's going to happen?
- Does this work like a loop?
- What type of loop does it work like?

A slight modification

```
def wibbler(n):
    print("Wibble",n)
    return wibbler(n+1)

print(wibbler(1))
```

- How many times does it run?

An improved wibbler……

```
def wibbler(n):
 if (n>0):
    print ("wibble")
    return wibbler(n-1)

wibbler(3)
```

- What's going to happen
- What loop structure is this like?

What's the difference?

```
def wibbler(n):
 if (n>0):
   print ("wibble")
   return wibbler(n-1)
```

```
def wibble(n):
        if (n==1):
                return ("wibble")
        return ("wibble "+wibble(n-1))
```


What's the difference?

```
def wibbler(n):
 if (n>0):
   print ("wibble")
   return wibbler(n-1)
```

```
def wibble(n):
        if (n==1):
                return ("wibble")
        return ("wibble "+wibble(n-1))
```


```
wibbler(3)
```

```
print(wibble(3))
```

```
def wibble(n):
        if (n==1):
                return ("wibble")
        return ("wibble "+wibble(n-1))
```

Base Case – stop condition

Recursive Case

- Define something in terms of itself
  - Recursive Case –simplifies the problem and moves towards the base case with a recursive call
  - Base Case –smallest instance (You may need more than one base case)

Queen Mary University of London · SUPPORTED BY MAYOR OF LONDON · COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE · KING'S College LONDON

Task - allstar

- Given a string, compute recursively a new string where all the adjacent chars are now separated by a "*".
- allStar("hello") → "h*e*l*l*o"
- allStar("ab") → "a*b"
- allStar("a")→"a"

Identify a way to break a problem up recursively….

Look for a Recursive Case and a Base Case

```
def allStar(string):
        if (??????):
                return ???????
        return (?????????????)
```

Queen Mary University of London · SUPPORTED BY MAYOR OF LONDON · COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE · KING'S College LONDON

Task time

- Reverse

What's going on............................

A child couldn't sleep, so her mother told her a story about a little frog,

who couldn't sleep, so the frog's mother told her a story about a little bear,

who couldn't sleep, so the bear's mother told her a story about a little weasel...

who fell asleep

and the little bear fell asleep;

and the little frog fell asleep;

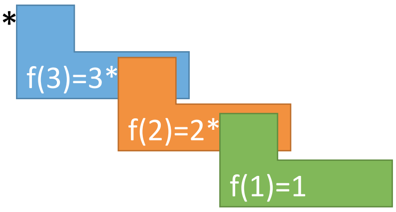
and the child fell asleep.

Queen Mary University of London | SUPPORTED BY MAYOR OF LONDON | COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE | KING'S College LONDON

Factorial revisted

- f(n)=n*f(n-1)
- f(1)=1

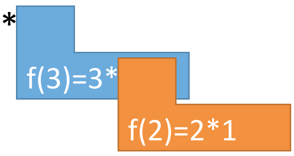
- f(4)=4*



Factorial revisted

- f(n)=n*f(n-1)
- f(1)=1

- f(4)=4*



Queen Mary University of London
SUPPORTED BY MAYOR OF LONDON
COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE
KING'S College LONDON

Factorial revisted

- f(n)=n*f(n-1)
- f(1)=1

- f(4)=4*

f(3)=3*

f(2)=2

Factorial revisted

- f(n)=n*f(n-1)
- f(1)=1

- f(4)=4*

f(3)=3*2

Queen Mary University of London · SUPPORTED BY MAYOR OF LONDON · COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE · KING'S College LONDON

Factorial revisted

- f(n)=n*f(n-1)
- f(1)=1

- f(4)=4*

f(3)=6

Queen Mary University of London · SUPPORTED BY MAYOR OF LONDON · COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE · KING'S College LONDON

Factorial revisited

- f(n)=n*f(n-1)
- f(1)=1

- f(4)=4*6

Queen Mary University of London · SUPPORTED BY MAYOR OF LONDON · COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE · KING'S College LONDON

Factorial revisted

- f(n)=n*f(n-1)
- f(1)=1

- f(4)=24

Why use recursion

- Recursion is a method of solving problems based on the divide and conquer mentality
- Sometimes its easier to solve think of a problem in terms of itself

- Q: Does using recursion usually make your code faster?
- A: No.
- Q: Does using recursion usually use less memory?
- A: No.
- Q: Then why use recursion?
- A: It sometimes makes your code much simpler!

Queen Mary University of London
SUPPORTED BY MAYOR OF LONDON
COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE
KING'S College LONDON

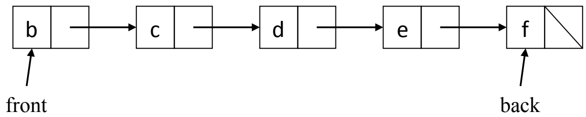
Predict-Explain-Modify-Create

- Predict – given a working program, what do you think it will do? (at a high level of abstraction)
- Run – run it and test your prediction
- Explain/Articulate – get into the nitty gritty. What does each line of code mean? (low level of abstraction). Lots of activities here: trace, annotate, explain, talk about, identify parts, etc….
- Modify – edit the program to make it do different things (high and low levels of abstraction) Design/Create – design a new program that uses the same nitty gritty but that solves a new problem.

Linked Lists

An implementation of the list abstractions

Linked List Concept

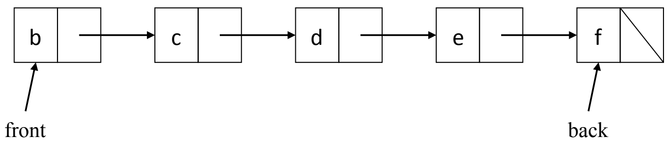


- Each entry has
  - A value
  - A pointer to the next entry
- Keep a pointer to the front entry
- The pointer of the last entry is None

- We could represent this in python by
- list=[['James',2],['Bob',0],['Sarah',-1]]
- startpos=1
- length=3

- We would need procedures to
- find a values position
- Find an pointers position
- Add
- Delete (and update the previous pointer – using find a pointer)
- Print in order
- Insert a new value

Exercise



- Redraw list after:
  - appending a new entry at the end
  - inserting before entry zero
  - inserting before entry 3

Linked List Index

```
myList.index(i)
```

- Count along the list to entry i
- Return the value

```
pointer = front
count = 0
while count < i:
   pointer ← next of current entry
   count = count + 1
return the value of current entry
```

Linked List Update

`myList.update(idx, newValue)`

- Count along the list to entry index
- Replace value with new value

```
pointer = front
count = 0
while count < idx:
   pointer ← next of currentEntry
   count = count = 1
currentEntry.value = newValue
```


Linked List Insert

`myList.insert(idx, newValue)`

- Count along the list to entry idx-1
- Insert a new entry
  - Next pointer of current entry points to new entry
  - Next pointer of new entry points to following entry

Queen Mary University of London · SUPPORTED BY MAYOR OF LONDON · COMPUTING AT SCHOOL EDUCATE · ENGAGE · ENCOURAGE · KING'S College LONDON