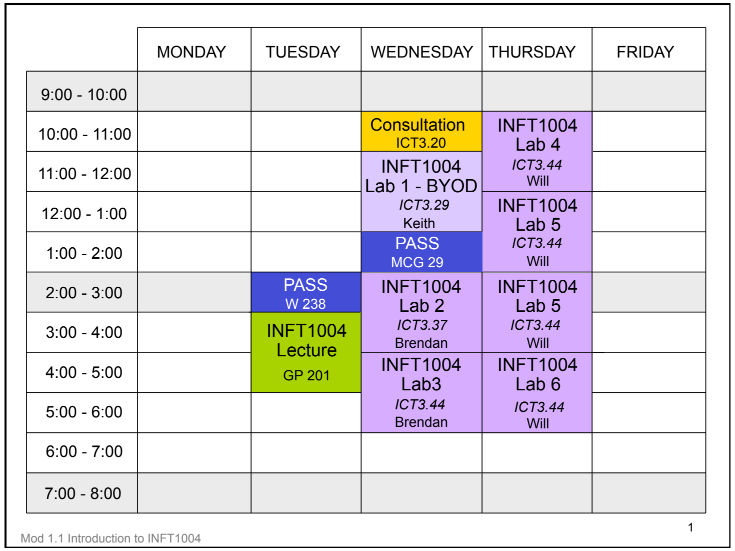
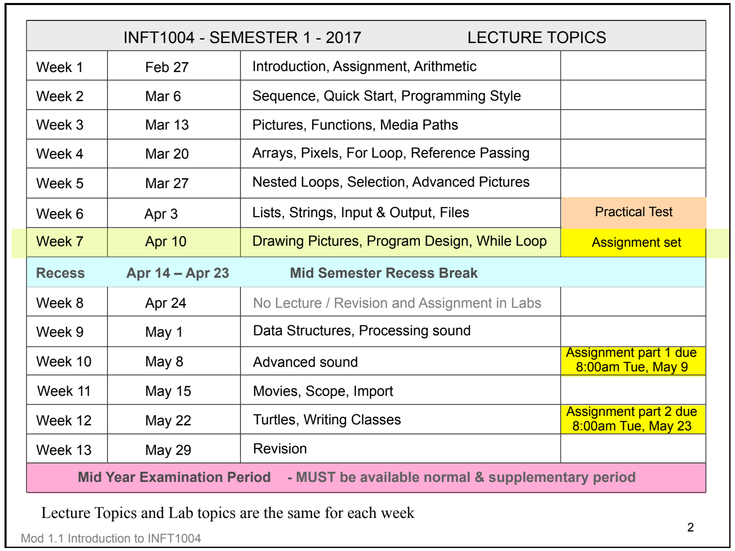


	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
9:00 - 10:00					
10:00 - 11:00			Consultation ICT3.20	INFT1004 Lab 4 *ICT3.44* Will	
11:00 - 12:00			INFT1004 Lab 1 - BYOD *ICT3.29* Keith		
12:00 - 1:00				INFT1004 Lab 5 *ICT3.44* Will	
1:00 - 2:00			PASS MCG 29		
2:00 - 3:00		PASS W 238	INFT1004 Lab 2 *ICT3.37* Brendan	INFT1004 Lab 5 *ICT3.44* Will	
3:00 - 4:00		INFT1004 Lecture GP 201			
4:00 - 5:00			INFT1004 Lab3 *ICT3.44* Brendan	INFT1004 Lab 6 *ICT3.44* Will	
5:00 - 6:00					
6:00 - 7:00					
7:00 - 8:00					

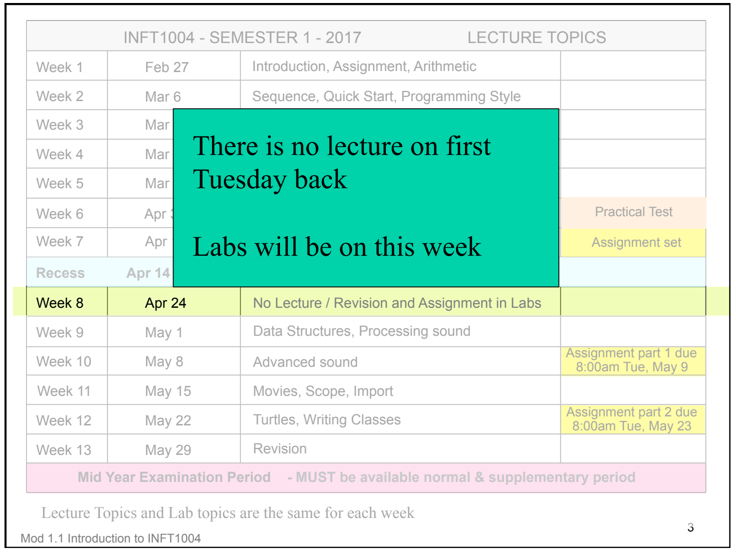
Mod 1.1 Introduction to INFT1004 1



INFT1004 - SEMESTER 1 - 2017		LECTURE TOPICS	
Week 1	Feb 27	Introduction, Assignment, Arithmetic	
Week 2	Mar 6	Sequence, Quick Start, Programming Style	
Week 3	Mar 13	Pictures, Functions, Media Paths	
Week 4	Mar 20	Arrays, Pixels, For Loop, Reference Passing	
Week 5	Mar 27	Nested Loops, Selection, Advanced Pictures	
Week 6	Apr 3	Lists, Strings, Input & Output, Files	Practical Test
Week 7	Apr 10	Drawing Pictures, Program Design, While Loop	Assignment set
Recess	**Apr 14 – Apr 23**	**Mid Semester Recess Break**	
Week 8	Apr 24	No Lecture / Revision and Assignment in Labs	
Week 9	May 1	Data Structures, Processing sound	
Week 10	May 8	Advanced sound	Assignment part 1 due 8:00am Tue, May 9
Week 11	May 15	Movies, Scope, Import	
Week 12	May 22	Turtles, Writing Classes	Assignment part 2 due 8:00am Tue, May 23
Week 13	May 29	Revision	
Mid Year Examination Period - MUST be available normal & supplementary period			

Lecture Topics and Lab topics are the same for each week

Mod 1.1 Introduction to INFT1004 2



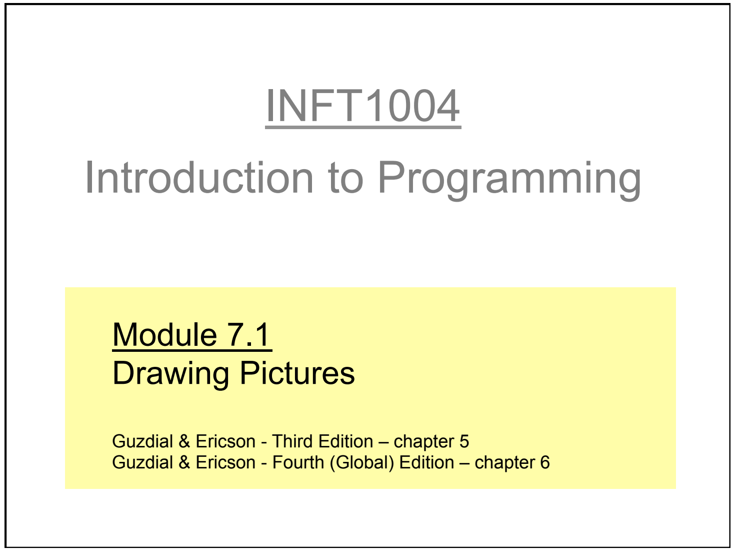
INFT1004 - SEMESTER 1 - 2017		LECTURE TOPICS	
Week 1	Feb 27	Introduction, Assignment, Arithmetic	
Week 2	Mar 6	Sequence, Quick Start, Programming Style	
Week 3	Mar		
Week 4	Mar		
Week 5	Mar		
Week 6	Apr		Practical Test
Week 7	Apr		Assignment set
Recess	**Apr 14**		
Week 8	Apr 24	No Lecture / Revision and Assignment in Labs	
Week 9	May 1	Data Structures, Processing sound	
Week 10	May 8	Advanced sound	Assignment part 1 due 8:00am Tue, May 9
Week 11	May 15	Movies, Scope, Import	
Week 12	May 22	Turtles, Writing Classes	Assignment part 2 due 8:00am Tue, May 23
Week 13	May 29	Revision	
Mid Year Examination Period - MUST be available normal & supplementary period			

There is no lecture on first Tuesday back

Labs will be on this week

Lecture Topics and Lab topics are the same for each week

Mod 1.1 Introduction to INFT1004 3

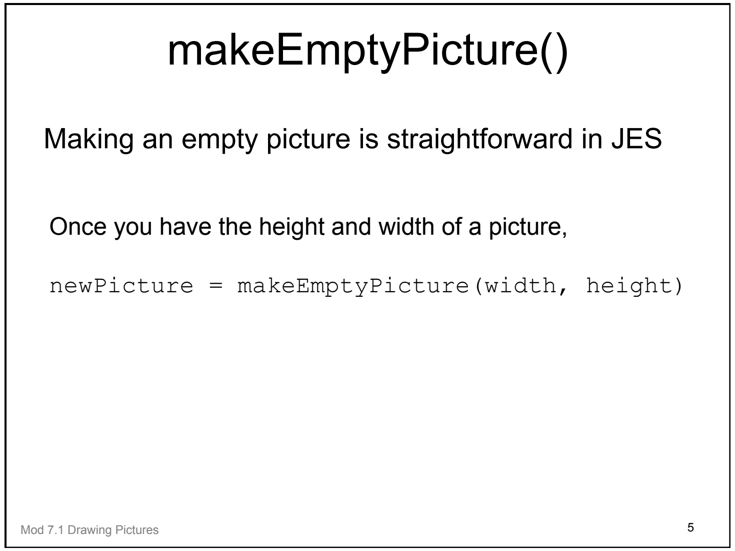


INFT1004

Introduction to Programming

Module 7.1
Drawing Pictures

Guzdial & Ericson - Third Edition – chapter 5
Guzdial & Ericson - Fourth (Global) Edition – chapter 6

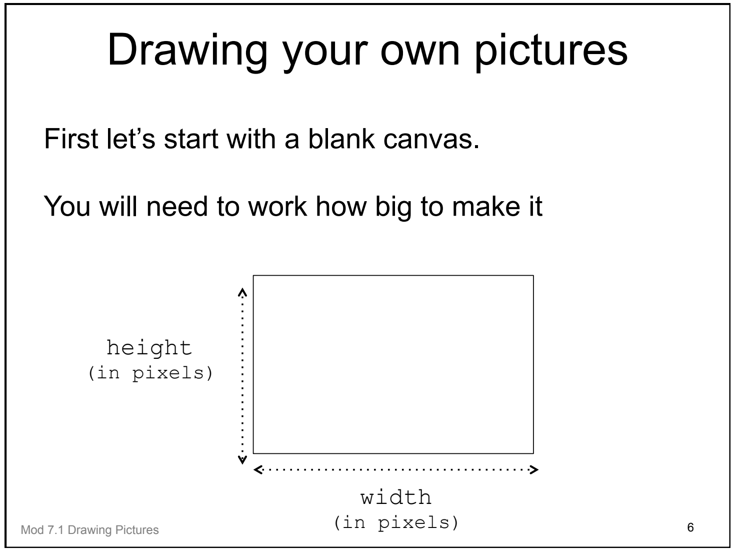


makeEmptyPicture()

Making an empty picture is straightforward in JES

Once you have the height and width of a picture,

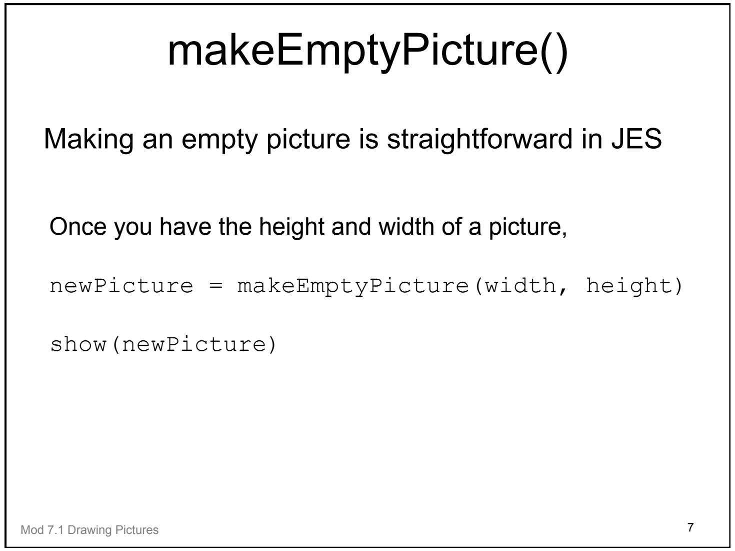
```
newPicture = makeEmptyPicture(width, height)
```



Drawing your own pictures

First let’s start with a blank canvas.

You will need to work how big to make it

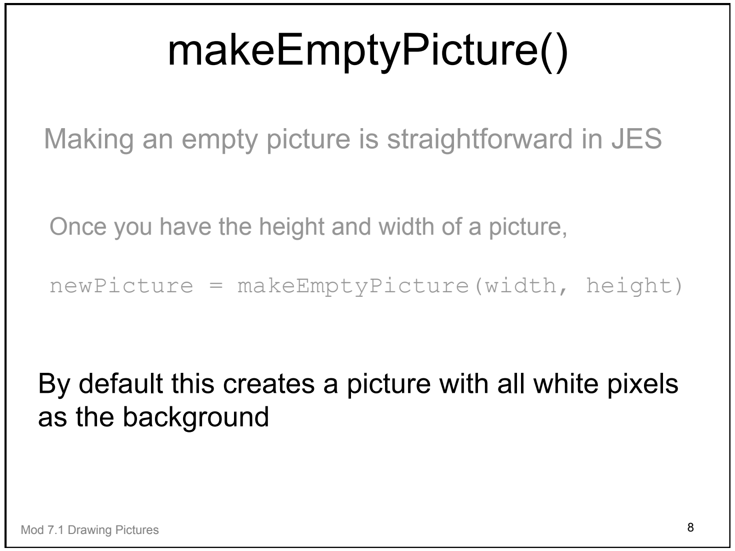


makeEmptyPicture()

Making an empty picture is straightforward in JES

Once you have the height and width of a picture,

```
newPicture = makeEmptyPicture(width, height)

show(newPicture)
```



makeEmptyPicture()

Making an empty picture is straightforward in JES

Once you have the height and width of a picture,

```
newPicture = makeEmptyPicture(width, height)
```

By default this creates a picture with all white pixels as the background

makeEmptyPicture()

Making an empty picture is straightforward in JES

Once you have the height and width of a picture,

```
newPicture = makeEmptyPicture(width, height)
```

You can choose a color to make the background

```
newPicture = makeEmptyPicture(width, height, red)
```

Default arguments

This is known as a default argument.

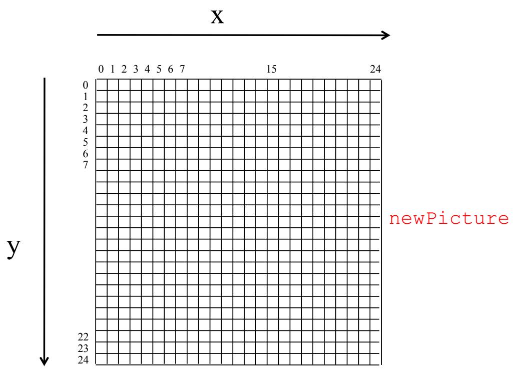
If you call the function with only 2 arguments (width, height) it makes a white picture. If you provide a color it makes the empty picture all that color

```
newPicture = makeEmptyPicture(width, height)

newPicture = makeEmptyPicture(width, height, red)
```

optional argument

Lets do some drawing

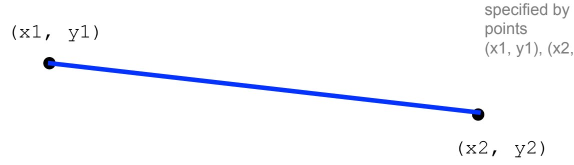
```
newPicture = makeEmptyPicture(25, 25)
```



addLine(…)

Like most programming languages, JES lets us add certain features directly to a picture

`addLine(pic, x1, y1, x2, y2)`	a line from (x1, y1) to (x2, y2)

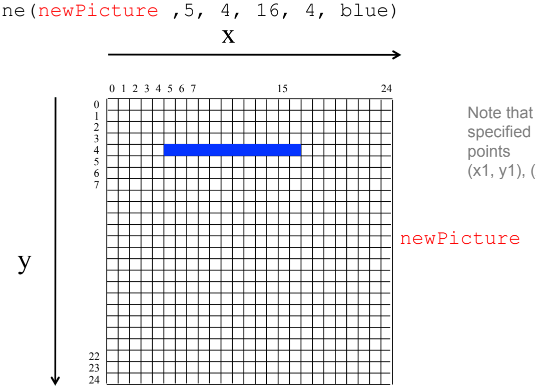


Note that the line is specified by two points (x1, y1), (x2, y2)

All of these can take an additional argument, colour

addLine(…)

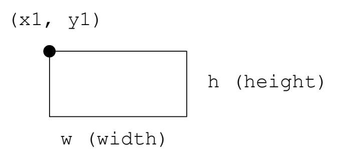
```
newPicture = makeEmptyPicture(25, 25)
addLine(newPicture ,5, 4, 16, 4, blue)
```



Note that the line is specified by two points
(x1, y1), (x2, y2)

addRect(…)

Like most programming languages, JES lets us add certain features directly to a picture

`addRect(pic, x1, y1, w, h)`	a rectangle of width w and height h starting at (x1, y1)
	Note that the position is specified by the top left coordinate (x1, y1) It uses the width and height to determine the size of rectangle

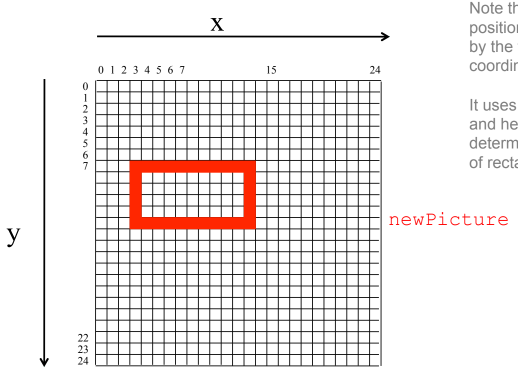
All of these can take an additional argument, colour

addRect(…)

```
newPicture = makeEmptyPicture(25, 25)
addRect(newPicture , 3, 7, 11, 6, red)
```

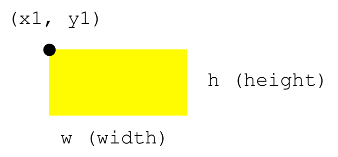
Note that the position is specified by the top left coordinate (x1, y1)

It uses the width and height to determine the size of rectangle



addRectFilled(…)

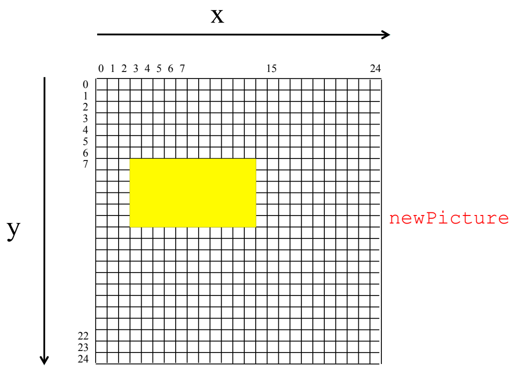
Like most programming languages, JES lets us add certain features directly to a picture

`addRectFilled(pic, x1, y1, w, h)`	a filled rectangle
	Note that the position is specified by the top left coordinate (x1, y1) It uses the width and height to determine the size of filled rectangle (there is no outline)

All of these can take an additional argument, colour

addRectFilled(…)

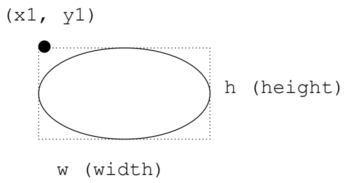
```
newPicture = makeEmptyPicture(25, 25)
addRectFilled(newPicture, 3, 7, 11, 6, yellow)
```



Drawing on pictures

Like most programming languages, JES lets us add certain features directly to a picture

`addOval(pic, x1, y1, w, h)`	an oval that fits in a rectangle of width w and height h starting at (x1, y1)



Note that the position is specified by the top left coordinate (x1, y1)

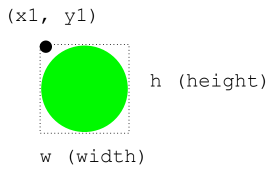
It uses the width and height to determine the size of oval – it touches the sides of an imaginary rectangle

All of these can take an additional argument, colour

Drawing on pictures

Like most programming languages, JES lets us add certain features directly to a picture

`addOvalFilled(pic, x1, y1, w, h)`	a filled oval



Note that the position is specified by the top left coordinate (x1, y1)

It uses the width and height to determine the size of oval – it touches the sides of an imaginary rectangle
(Note if width is the same as height it makes a circle.
(no outline is drawn)

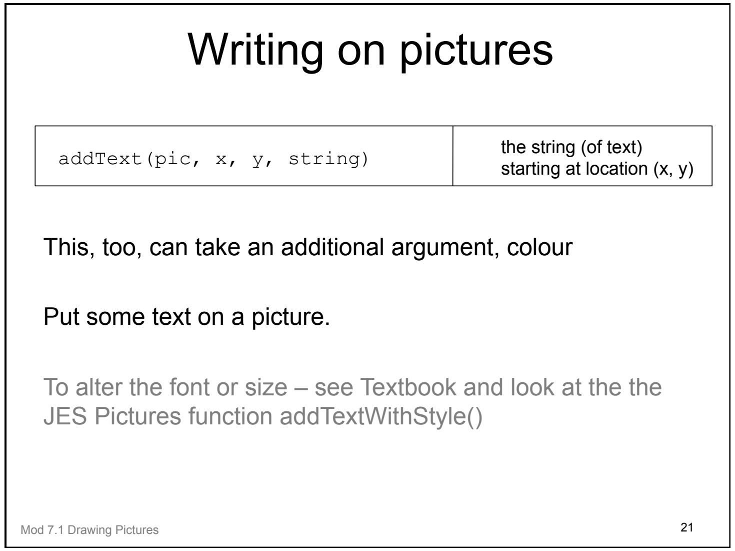
All of these can take an additional argument, colour

Drawing on pictures

Like most programming languages, JES lets us add certain features directly to a picture

`addLine(pic, x1, y1, x2, y2)`	a line from (x1, y1) to (x2, y2)
`addRect(pic, x1, y1, w, h)`	a rectangle of width w and height h starting at (x1, y1)
`addRectFilled(pic, x1, y1, w, h)`	a filled rectangle
`addOval(pic, x1, y1, w, h)`	an oval that fits in a rectangle of width w and height h starting at (x1, y1)
`addOvalFilled(pic, x1, y1, w, h)`	a filled oval

All of these can take an additional argument, colour



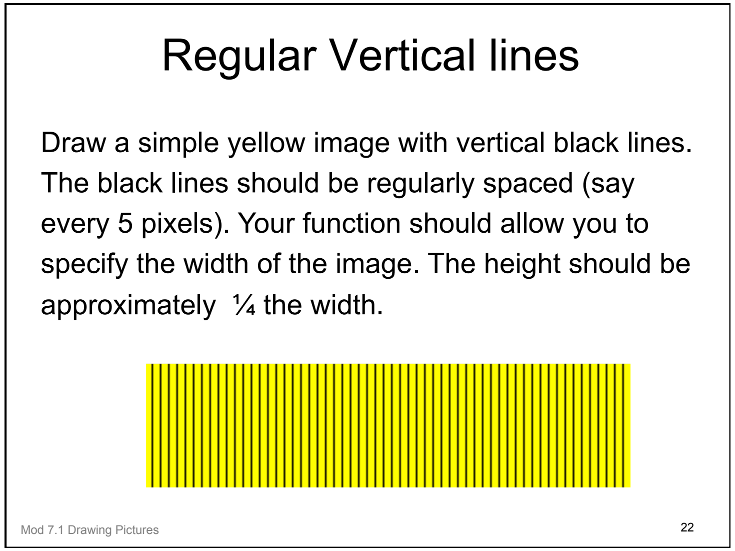
Writing on pictures

`addText(pic, x, y, string)`	the string (of text) starting at location (x, y)

This, too, can take an additional argument, colour

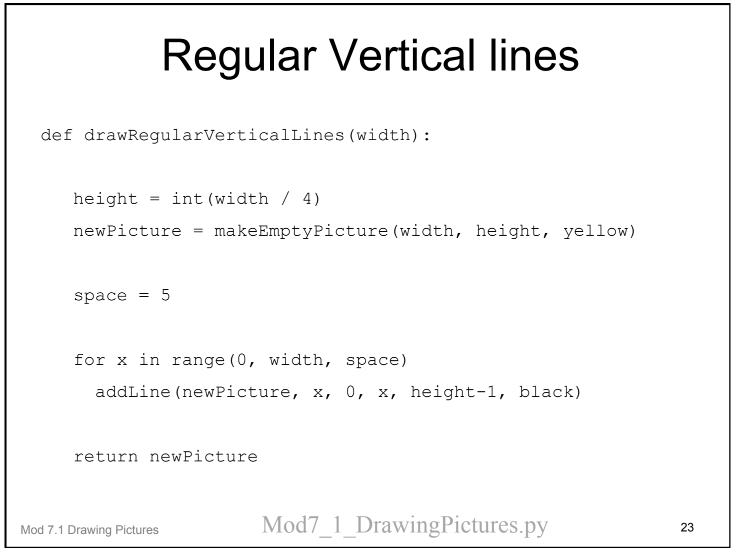
Put some text on a picture.

To alter the font or size – see Textbook and look at the the JES Pictures function addTextWithStyle()



Regular Vertical lines

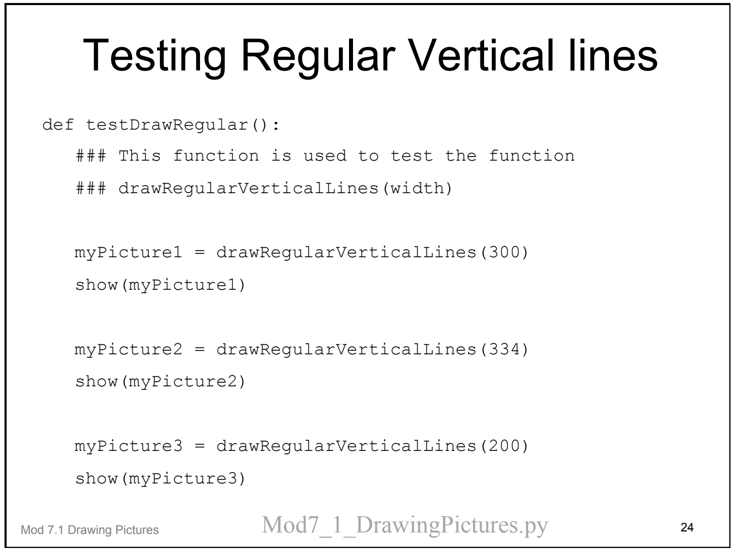
Draw a simple yellow image with vertical black lines. The black lines should be regularly spaced (say every 5 pixels). Your function should allow you to specify the width of the image. The height should be approximately ¼ the width.



Regular Vertical lines

```
def drawRegularVerticalLines(width):

  height = int(width / 4)
  newPicture = makeEmptyPicture(width, height, yellow)

  space = 5

  for x in range(0, width, space)
    addLine(newPicture, x, 0, x, height-1, black)

  return newPicture
```

Mod7_1_DrawingPictures.py

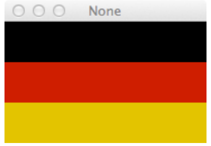


Testing Regular Vertical lines

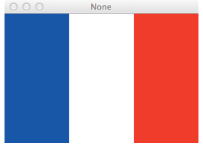
```
def testDrawRegular():
  ### This function is used to test the function
  ### drawRegularVerticalLines(width)

  myPicture1 = drawRegularVerticalLines(300)
  show(myPicture1)

  myPicture2 = drawRegularVerticalLines(334)
  show(myPicture2)

  myPicture3 = drawRegularVerticalLines(200)
  show(myPicture3)
```

Mod7_1_DrawingPictures.py

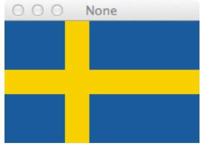
Drawing Some Flags



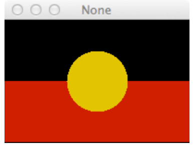
German



French

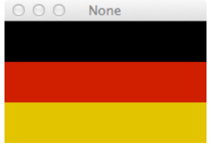


Swedish



Aboriginal

Drawing German Flag

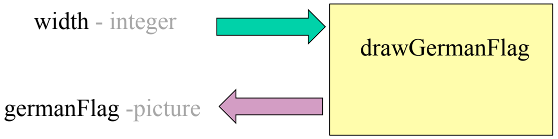


German

Write a function that creates and returns a picture of the German Flag.

The function requires one parameter which is the width of the flag.

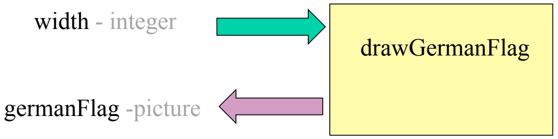
Drawing German Flag



Write a function that creates and returns a picture of the German Flag.

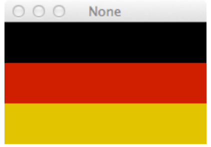
The function requires one parameter which is the width of the flag.

Function definiton



```
def createGermanFlag(width):

   .
   .

   return flag   # a picture
```

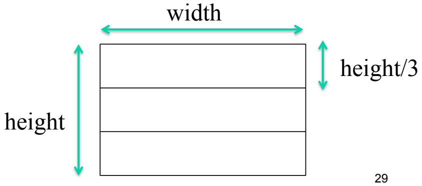
Drawing German Flag



German

I looked on the internet (not sure how reliable that is) but found that the height of the German flag is three fifths of the width.

height = width * 3/5

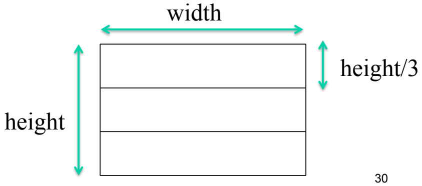


Function definiton

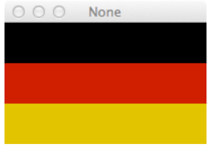
```
def createGermanFlag(width):

   #use float division for the 5
   height = (width * 3) / 5.0
   thirdHeight = height / 3
   height = int(height)
   .
   .
   return flag # a picture
```

Work out correct sizes for flag and stripes

height = width * 3/5



Drawing German Flag



German

I looked on the internet (not sure how reliable that is) I found the colours (in Pantone which is how they colour flags apparently) and I worked out the red, green and blue values from another web site (since I don't trust the internet I checked a few different sites to make sure they agreed)

```
Black:  r=0,g=0, b=0)
Red :   r=216,g=30,b=5       #Pantone 485
Yellow: r=230,g=196,b=20     #Pantone 7405
```

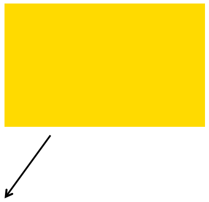
Empty German Flag

```
def createGermanFlag(width):

   cBlack = makeColor(0,0,0)
   cRed = makeColor(216,30,5)
   cGold = makeColor(230,196,20)

   height = (width * 3) / 5.0
   thirdHeight = int(height / 3)
   height = int(height)



   .
   .

   return flag
```

Set up the colors I need

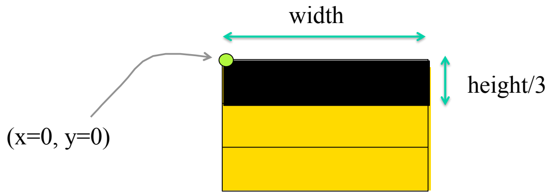
Empty German Flag

```
def createGermanFlag(width):

   cBlack = makeColor(0,0,0)
   cRed = makeColor(216,30,5)
   cGold = makeColor(230,196,20)

   height = (width * 3) / 5.0
   thirdHeight = int(height / 3)
   height = int(height)

   flag = makeEmptyPicture(width, height, cGold)

   .
   .

   return flag
```



This makes a blank picture (in gold) that is the right size

Drawing German Flag



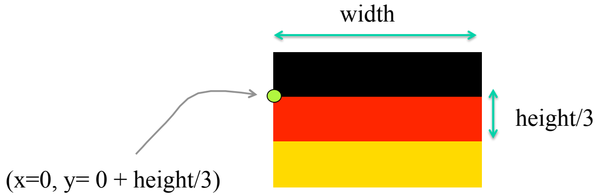
Work out the top left corner (x, y) of the black rectangle (check function definition)

Work out the width and height of the rectangle

Drawing German Flag

```
def createGermanFlag(width):

   cBlack = makeColor(0,0,0)
   cRed = makeColor(216,30,5)
   cGold = makeColor(230,196,20)

   height = (width * 3) / 5.0
   thirdHeight = int(height / 3)
   height = int(height)

   flag = makeEmptyPicture(width, height, cGold)

   addRectFilled(flag, 0,0, width, thirdHeight, cBlack)

   .
   return flag
```

Drawing German Flag



Work out the top left corner (x, y) of the red rectangle (check function definition)

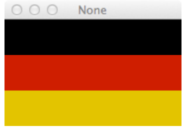
Work out the width and height of the rectangle

Drawing German Flag

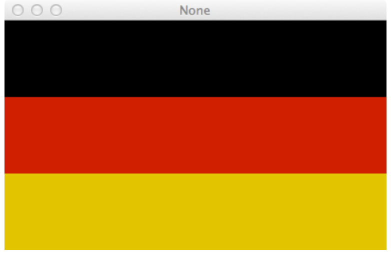
```
def createGermanFlag(width):

   cBlack = makeColor(0,0,0)
   cRed = makeColor(216,30,5)
   cGold = makeColor(230,196,20)

   height = (width * 3) / 5.0
   thirdHeight = int(height / 3)
   height = int(height)

   flag = makeEmptyPicture(width, height, cGold)

   addRectFilled(flag, 0,0, width, thirdHeight, cBlack)
   addRectFilled(flag, 0, thirdHeight, width, thirdHeight, cRed)

   return flag
```

Mod7_1_DrawingPictures.py

Test German Flag

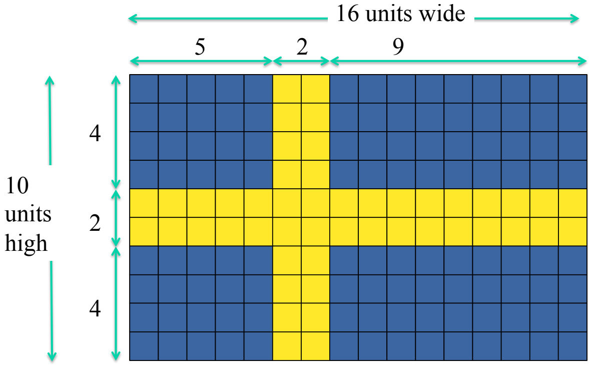


```
>>> createGermanFlag(200)
```



```
>>> createGermanFlag(200)
```

Drawing a Swedish Flag



Drawing Swedish Flags

```
def createSwedenFlag(width):
  cBlue   = makeColor(0, 91, 153)
  cYellow = makeColor(252, 209, 22)

  height = int((width / 16.0) * 10)
  flag = makeEmptyPicture(width, height, colourBlue)

  oneSixtenth = width / 16.0
  stripeSize = int(2 * oneSixtenth)
  xPos = int(one16*5)
  yPos = 0

  addRectFilled(flag, xPos , yPos, stripeSize, height, cYellow)

  oneTenth = height / 10.0
  xPos = 0
  yPos = int(oneTenth *4)
  addRectFilled(flag, xPos, yPos, width, stripeSize, cYellow)

  return flag
```

Mod7_1_DrawingPictures.py

Testing my Flags

```
def testSimpleFlags():

    germanFlag = createGermanFlag(500)
    show(germanFlag)

    #frenchFlag = createFrenchFlag(300)
    #show(frenchFlag)

    swedishFlag = createSwedenFlag(280)
    show(swedishFlag)

    #aboriginalFlag = createAboriginalFlag(700)
    #show(aboriginalFlag)
```

Mod7_1_DrawingPictures.py

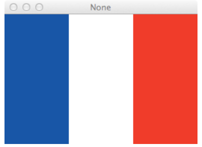
Testing my Flags

```
def testSimpleFlags():

    germanFlag = createGermanFlag(500)
    show(germanFlag)

    #frenchFlag = createFrenchFlag(300)
    #show(frenchFlag)

    swedishFlag = createSwedenFlag(280)
    show(swedishFlag)

    #aboriginalFlag = createAboriginalFlag(700)
    #show(aboriginalFlag)
```

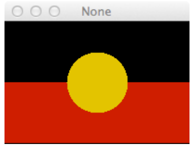
Mod7_1_DrawingPictures.py

Now you try some

```
def testSimpleFlags():

   germanFlag = createGermanFlag(500)
   show(germanFlag)

   #frenchFlag = createFrenchFlag(300)
   #show(frenchFlag)

   swedishFlag = createSwedenFlag(280)
   show(swedishFlag)

   #aboriginalFlag = createAboriginalFlag(700)
   #show(aboriginalFlag)
```



French



Aboriginal

For this weeks tut

INFT1004

Introduction to Programming

Module 7.2

Program Design

Guzdial & Ericson - Third Edition – chapter 9
Guzdial & Ericson - Fourth (Global) Edition – chapter 10

Assignment

Please read the assignment requirements carefully.

I will reward people who follow instructions (these is very important in IT generally!)

Specifications/requirement are important!

How the customer explained it

How the Project Leader understood it

How the Analyst designed it

How the Programmer wrote it

How the Business Consultant described it

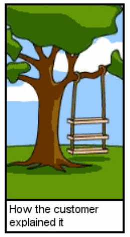
How the project was documented

What operations installed

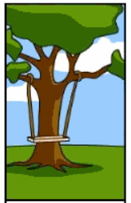
How the customer was billed

How it was supported

What the customer really needed



How the customer explained it



How the Project Leader understood it

How the Analyst designed it

How the Programmer wrote it

How the Business Consultant described it

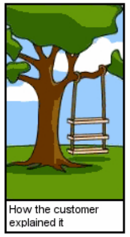
How the project was documented

What operations installed

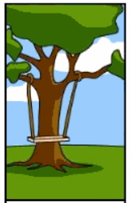
How the customer was billed

How it was supported

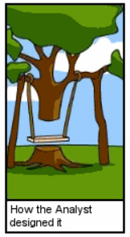
What the customer really needed



How the customer explained it



How the Project Leader understood it



How the Analyst designed it

How the Programmer wrote it

How the Business Consultant described it

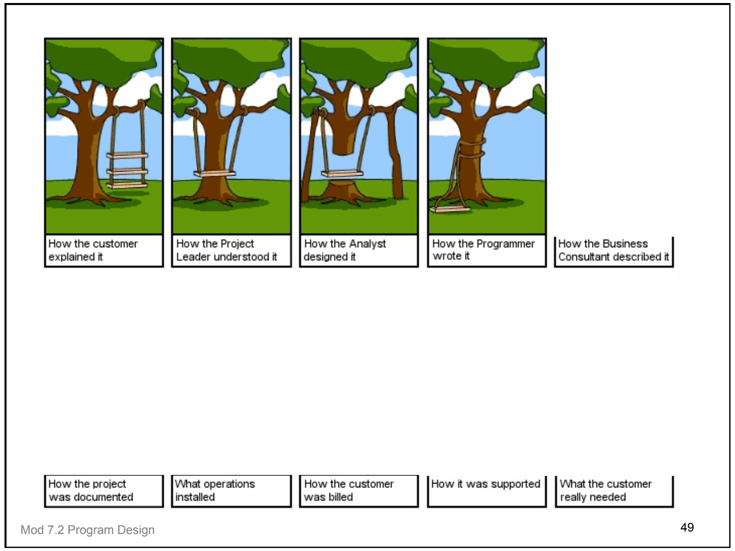
How the project was documented

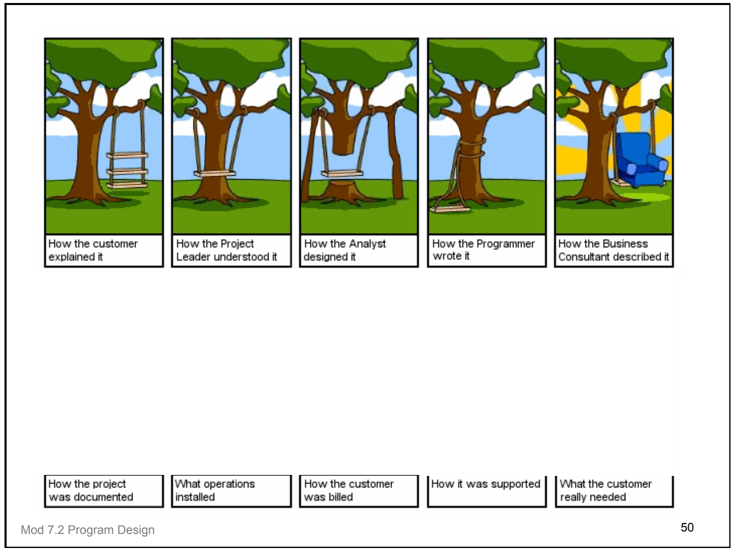
What operations installed

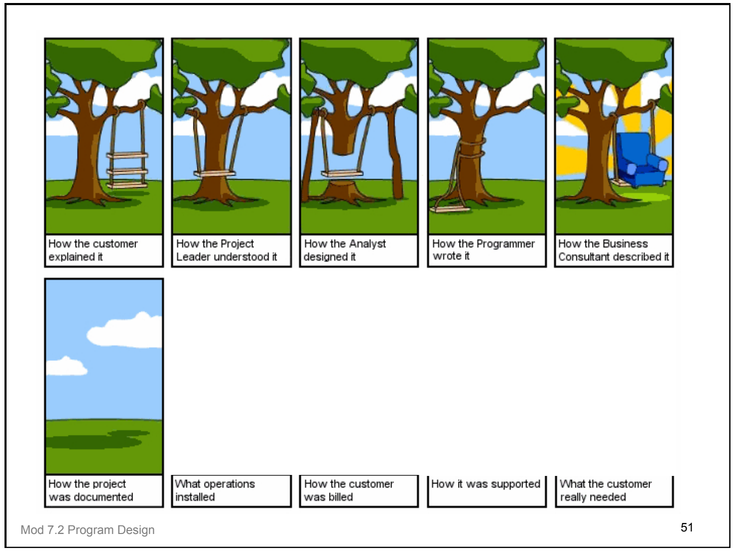
How the customer was billed

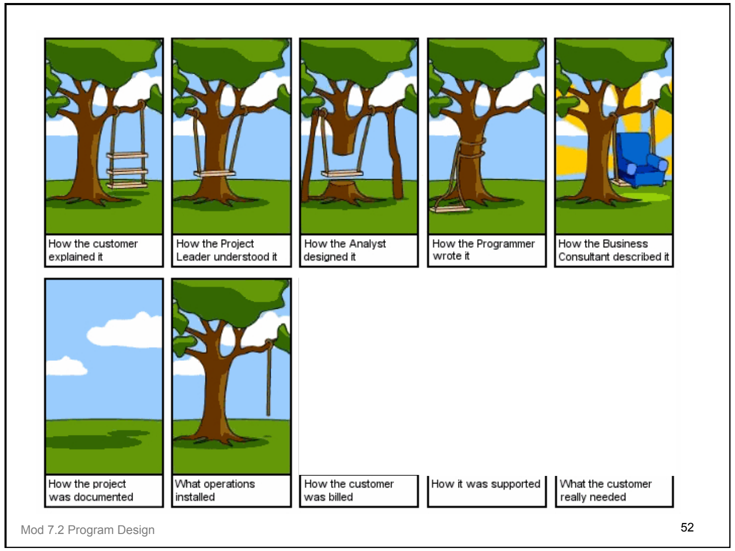
How it was supported

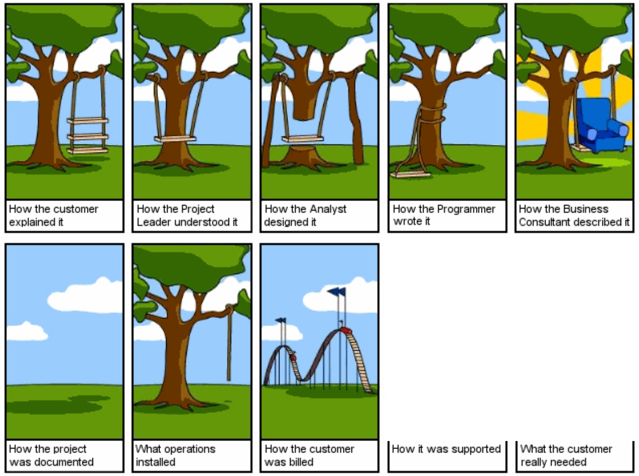
What the customer really needed


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

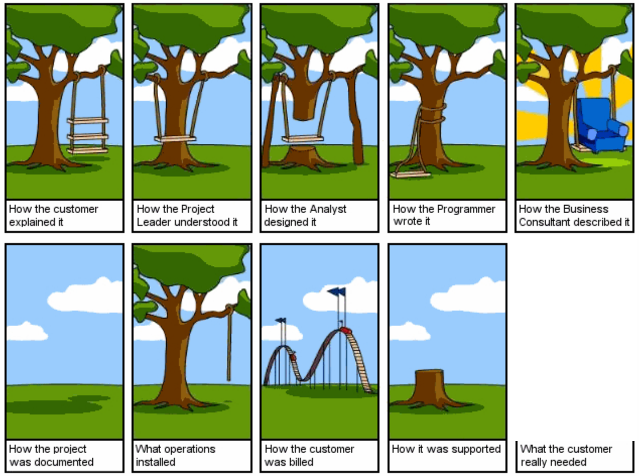

How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

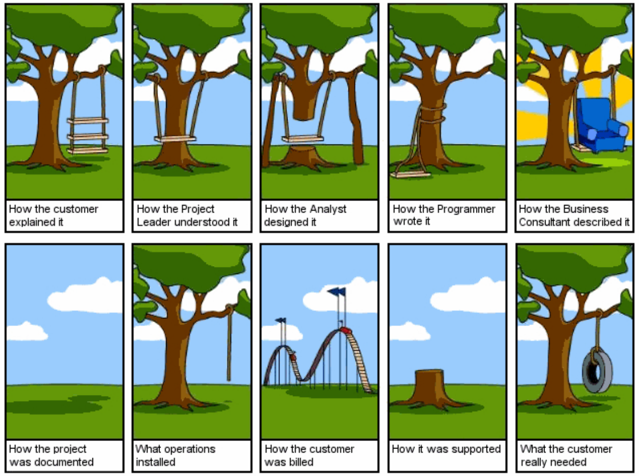

How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed



How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed



How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

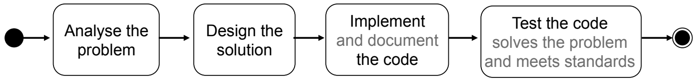


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

INFT1004 Visual Programming

This course will teach you to

- comprehend a programming problem and design a solution algorithm
- code the solution algorithm in a specific programming language (Python)
- test and document your program solutions according to suitable standards.



Analyse the problem

Design the solution

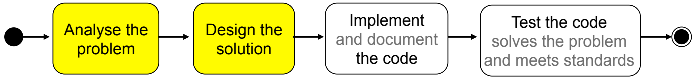
Implement and document the code

Test the code solves the problem and meets standards

Problem solving

It's not enough to be able to write code;

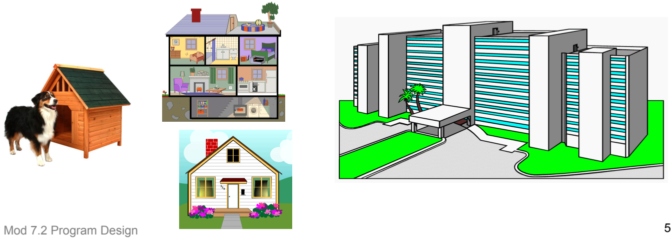
You need to be able to look at the problem, work out how to solve it, and then write the code that implements your solution


Analyse the problem
Design the solution
Implement and document the code
Test the code solves the problem and meets standards

Program Design

Program design can be difficult to understand as it often requires considerable experience to appreciate the benefits of good design.

Furthermore "good" design may depend on the size of the project.



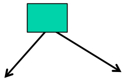
Program design

There are two 'standard' approaches to program design: top-down and bottom-up

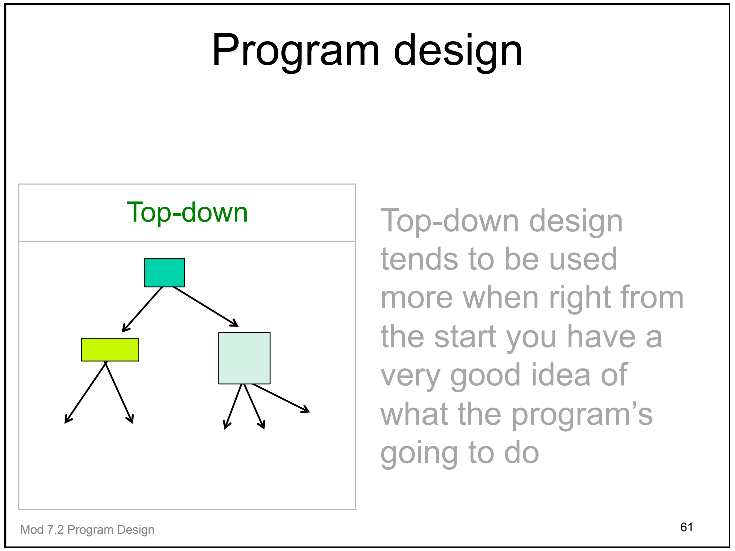
Top-down	Bottom-up

Program design

Top-down



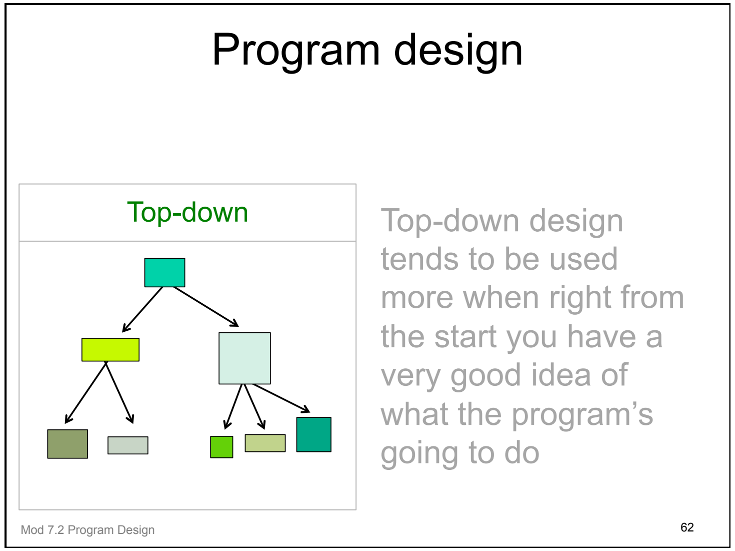
Top-down design tends to be used more when right from the start you have a very good idea of what the program's going to do



Program design

Top-down

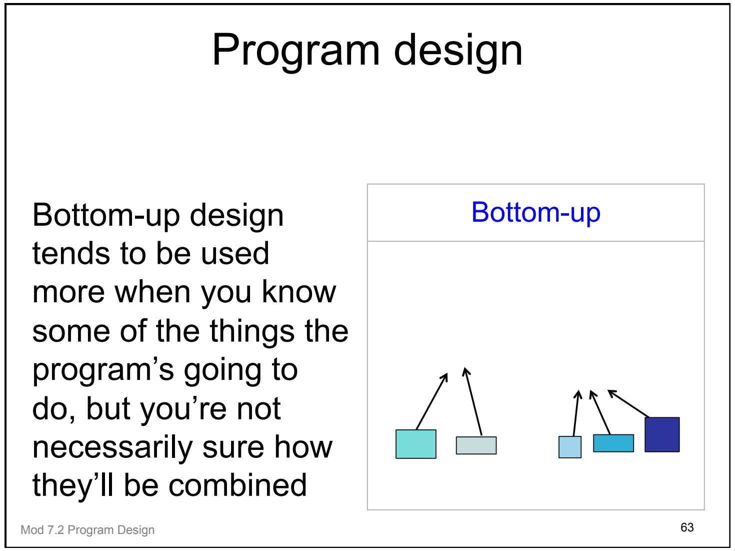
Top-down design tends to be used more when right from the start you have a very good idea of what the program's going to do



Program design

Top-down

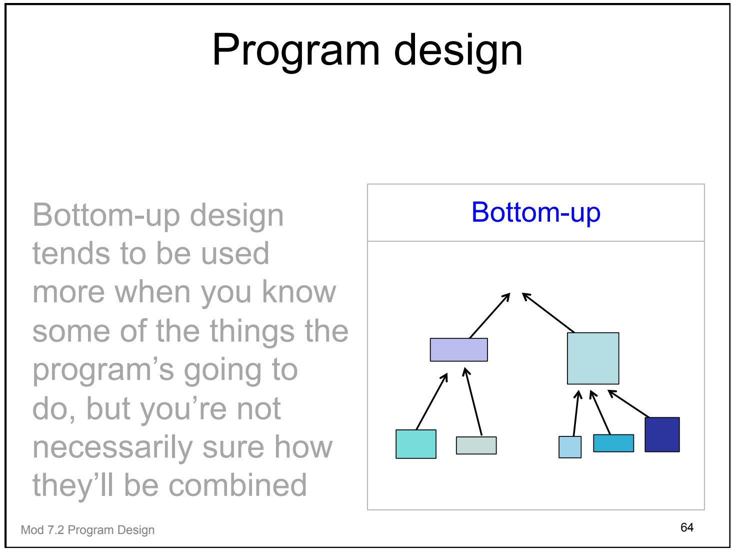
Top-down design tends to be used more when right from the start you have a very good idea of what the program's going to do



Program design

Bottom-up design tends to be used more when you know some of the things the program's going to do, but you're not necessarily sure how they'll be combined

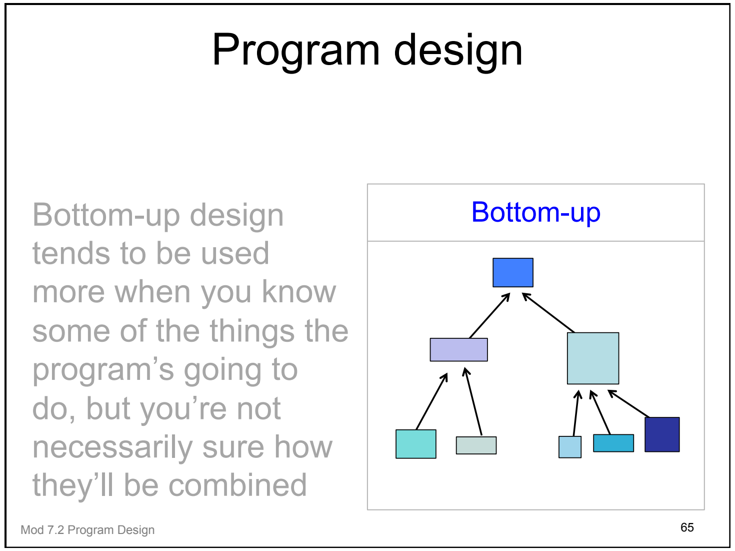
Bottom-up



Program design

Bottom-up design tends to be used more when you know some of the things the program's going to do, but you're not necessarily sure how they'll be combined

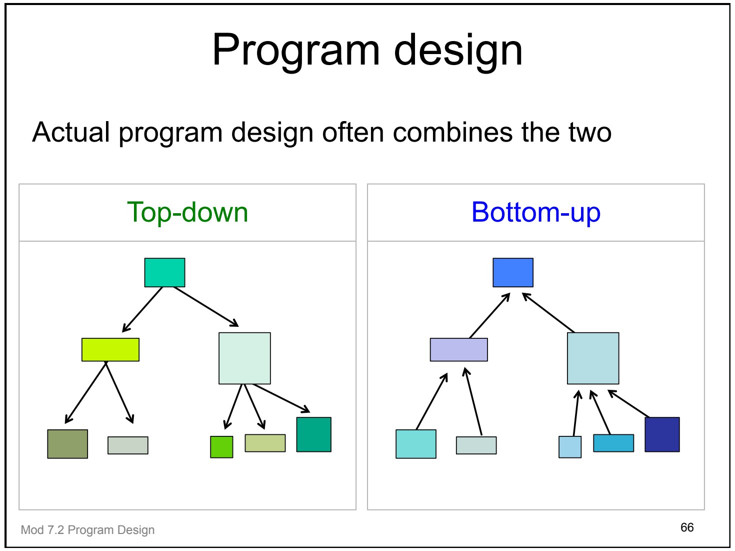
Bottom-up



Program design

Bottom-up design tends to be used more when you know some of the things the program's going to do, but you're not necessarily sure how they'll be combined

Bottom-up

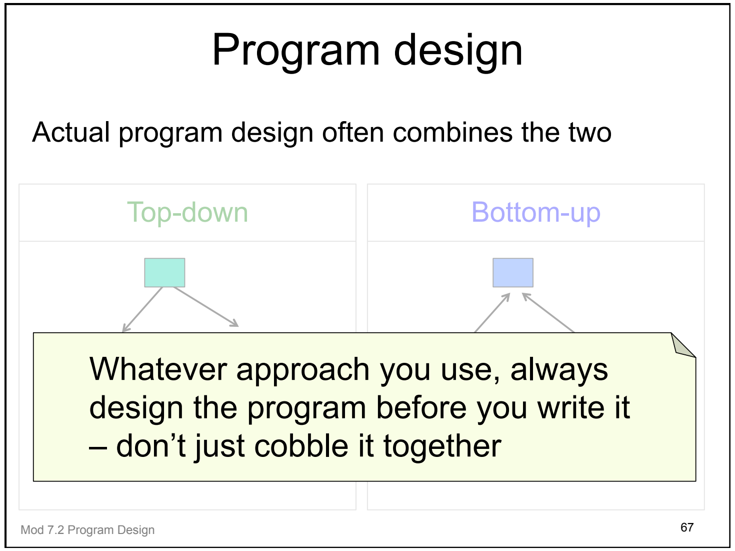


Program design

Actual program design often combines the two

Top-down

Bottom-up



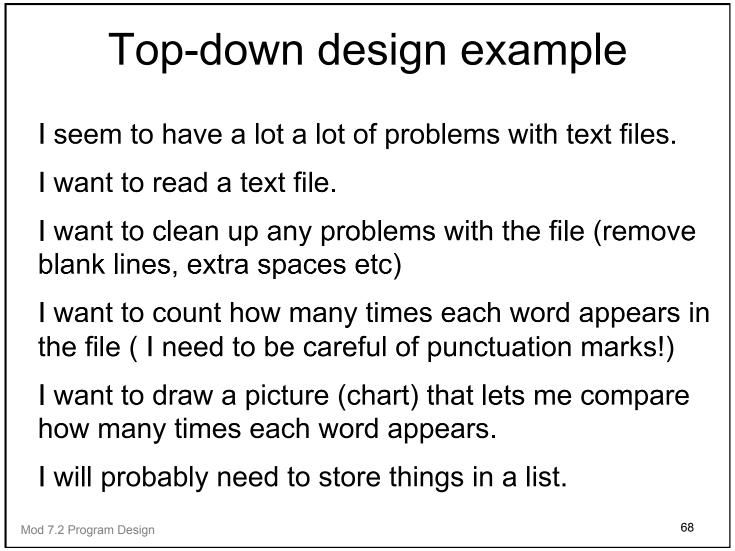
Program design

Actual program design often combines the two

Top-down

Bottom-up

Whatever approach you use, always design the program before you write it – don't just cobble it together



Top-down design example

I seem to have a lot a lot of problems with text files.

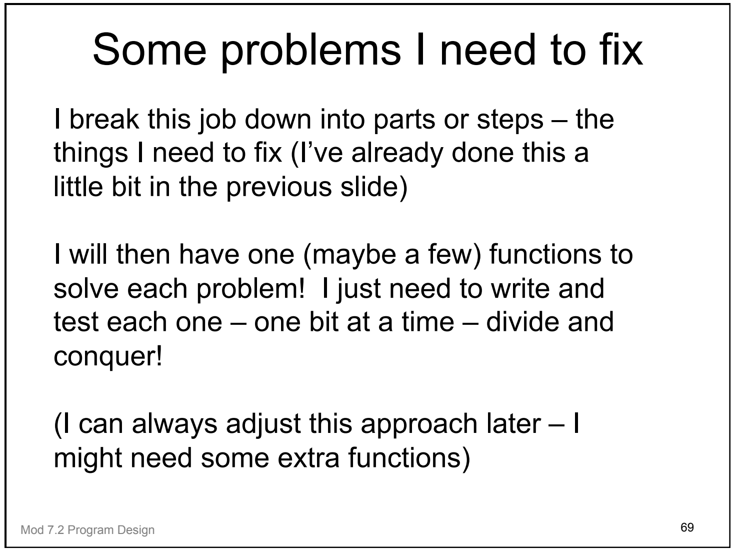
I want to read a text file.

I want to clean up any problems with the file (remove blank lines, extra spaces etc)

I want to count how many times each word appears in the file (I need to be careful of punctuation marks!)

I want to draw a picture (chart) that lets me compare how many times each word appears.

I will probably need to store things in a list.

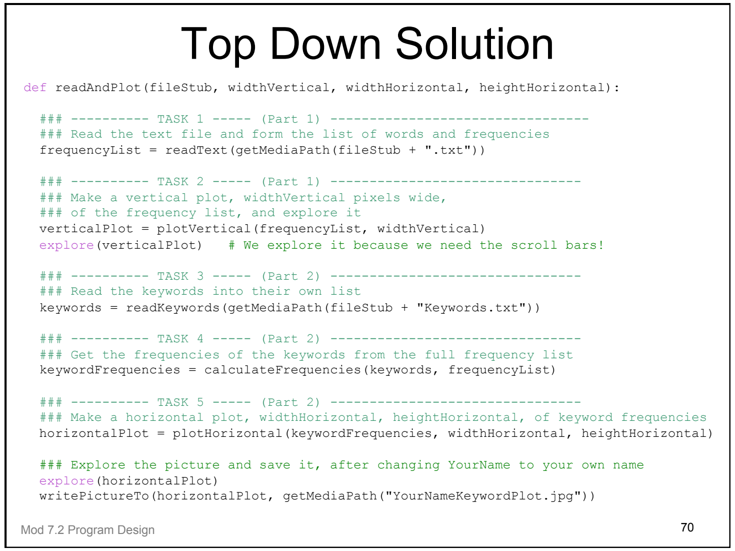


Some problems I need to fix

I break this job down into parts or steps – the things I need to fix (I've already done this a little bit in the previous slide)

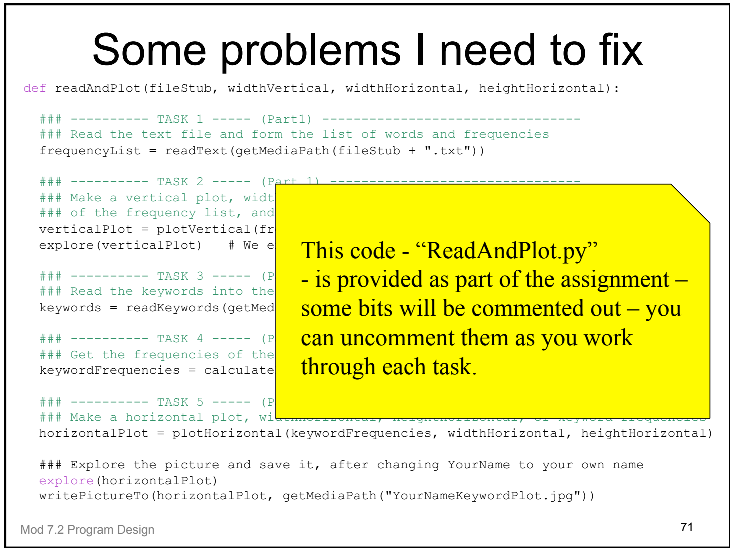
I will then have one (maybe a few) functions to solve each problem! I just need to write and test each one – one bit at a time – divide and conquer!

(I can always adjust this approach later – I might need some extra functions)



Top Down Solution

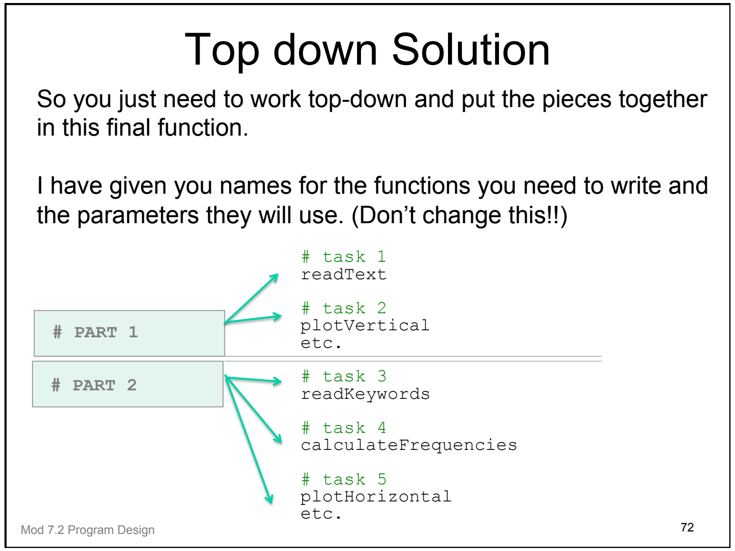
```
def readAndPlot(fileStub, widthVertical, widthHorizontal, heightHorizontal):

  ### ---------- TASK 1 ----- (Part 1) ---------------------------------
  ### Read the text file and form the list of words and frequencies
  frequencyList = readText(getMediaPath(fileStub + ".txt"))

  ### ---------- TASK 2 ----- (Part 1) --------------------------------
  ### Make a vertical plot, widthVertical pixels wide,
  ### of the frequency list, and explore it
  verticalPlot = plotVertical(frequencyList, widthVertical)
  explore(verticalPlot)   # We explore it because we need the scroll bars!

  ### ---------- TASK 3 ----- (Part 2) --------------------------------
  ### Read the keywords into their own list
  keywords = readKeywords(getMediaPath(fileStub + "Keywords.txt"))

  ### ---------- TASK 4 ----- (Part 2) --------------------------------
  ### Get the frequencies of the keywords from the full frequency list
  keywordFrequencies = calculateFrequencies(keywords, frequencyList)

  ### ---------- TASK 5 ----- (Part 2) --------------------------------
  ### Make a horizontal plot, widthHorizontal, heightHorizontal, of keyword frequencies
  horizontalPlot = plotHorizontal(keywordFrequencies, widthHorizontal, heightHorizontal)

  ### Explore the picture and save it, after changing YourName to your own name
  explore(horizontalPlot)
  writePictureTo(horizontalPlot, getMediaPath("YourNameKeywordPlot.jpg"))
```



Some problems I need to fix

```
def readAndPlot(fileStub, widthVertical, widthHorizontal, heightHorizontal):

  ### ---------- TASK 1 ----- (Part1) ---------------------------------
  ### Read the text file and form the list of words and frequencies
  frequencyList = readText(getMediaPath(fileStub + ".txt"))

  ### ---------- TASK 2 ----- (Part 1) ---------------------------------
  ### Make a vertical plot, widt
  ### of the frequency list, and
  verticalPlot = plotVertical(fr
  explore(verticalPlot)   # We e

  ### ---------- TASK 3 ----- (P
  ### Read the keywords into the
  keywords = readKeywords(getMed

  ### ---------- TASK 4 ----- (P
  ### Get the frequencies of the
  keywordFrequencies = calculate

  ### ---------- TASK 5 ----- (P
  ### Make a horizontal plot, wi
  horizontalPlot = plotHorizontal(keywordFrequencies, widthHorizontal, heightHorizontal)

  ### Explore the picture and save it, after changing YourName to your own name
  explore(horizontalPlot)
  writePictureTo(horizontalPlot, getMediaPath("YourNameKeywordPlot.jpg"))
```

This code - "ReadAndPlot.py" - is provided as part of the assignment – some bits will be commented out – you can uncomment them as you work through each task.



Top down Solution

So you just need to work top-down and put the pieces together in this final function.

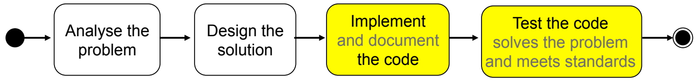
I have given you names for the functions you need to write and the parameters they will use. (Don't change this!!)

```
# PART 1
    # task 1
    readText
    # task 2
    plotVertical
    etc.

# PART 2
    # task 3
    readKeywords
    # task 4
    calculateFrequencies
    # task 5
    plotHorizontal
    etc.
```

Problem solving

Of course the code needs to be implemented and tested!

I would implement one bit at a time – testing it as I go.


Analyse the problem
Design the solution
Implement and document the code
Test the code solves the problem and meets standards

Write dummy functions

You will see in the code I provide that you can comment out some tasks – probably best to leave these as comments until you are ready to do them.

```
### ---------- TASK 2 ----- (Part 1) ----------------------
### Make a vertical plot, widthVertical pixels wide,
### of the frequency list, and explore it

#verticalPlot = plotVertical(frequencyList, widthVertical)
#explore(verticalPlot)
```

Although another option is to write dummy functions that don't anything.

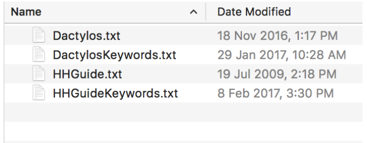
Test top level function

Either way you can at least test the top level function runs eg.

```
>>> readAndPlot("Dactylos", 500, 800, 600)
```

The file "Dactylos.txt" (and others have been provided to help you test your assignment – I'll use different ones for marking)

This numbers you should pick to something sensible – I will pick different ones for marking.



Name	Date Modified
Dactylos.txt	18 Nov 2016, 1:17 PM
DactylosKeywords.txt	29 Jan 2017, 10:28 AM
HHGuide.txt	19 Jul 2009, 2:18 PM
HHGuideKeywords.txt	8 Feb 2017, 3:30 PM

Write and test each function

The approach is to start writing each of the dummy functions with the full working code. Work on one at a time

Test as you go. Finishing each function in turn.

Eventually all the parts are complete and you can test the top-level function (try different width, heights)

```
>>> readAndPlot("Dactylos", 500, 800, 600)

>>> readAndPlot("HHGuide", 700, 920, 740)
```

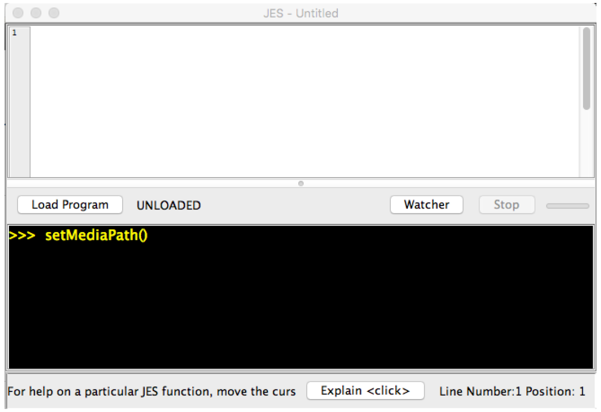
Write and test each function

Make sure you comment and follow our other style standards such as naming conventions.

Remember the pictures (plots) should end up in the current media path – you must set the media path correctly before you run your program.

(BUT DO NOT SET THE MEDIAPATH IN YOUR PROGRAM – DO IT IN THE COMMAND WINDOW)

(BUT DO NOT SET THE MEDIAPATH IN YOUR PROGRAM – DO IT IN THE COMMAND WINDOW)



```
>>> setMediaPath()
```

For help on a particular JES function, move the curs

Only part One for Part One

The assignment is in 2 parts.

When you hand in the first part – only the first part should run (so we can mark it)

If you've done more – just comment out any part 2 functions

Only part One for Part One

```
def readAndPlot(fileStub, widthVertical, widthHorizontal, heightHorizontal):

  ### ---------- TASK 1 ----- (Part 1) ---------------------------------
  ### Read the text file and form the list of words and frequencies
  frequencyList = readText(getMediaPath(fileStub + ".txt"))

  ### ---------- TASK 2 ----- (Part 1) --------------------------------
  ### Make a vertical plot, widthVertical pixels wide,
  ### of the frequency list, and explore it
  verticalPlot = plotVertical(frequencyList, widthVertical)
  explore(verticalPlot)   # We explore it because we need the scroll bars!

  ### ---------- TASK 3 ----- (Part 2) --------------------------------
  ### Read the keywords into their own list
  #keywords = readKeywords(getMediaPath(fileStub + "Keywords.txt"))

  ### ---------- TASK 4 ----- (Part 2) --------------------------------
  ### Get the frequencies of the keywords from the full frequency list
  #keywordFrequencies = calculateFrequencies(keywords, frequencyList)

  ### ---------- TASK 5 ----- (Part 2) --------------------------------
  ### Make a horizontal plot, widthHorizontal, heightHorizontal, of keyword frequencies
  horizontalPlot = plotHorizontal(keywordFrequencies, widthHorizontal, heightHorizontal)

  ### Explore the picture and save it, after changing YourName to your own name
  #explore(horizontalPlot)
  #writePictureTo(horizontalPlot, getMediaPath("YourNameKeywordPlot.jpg"))
```

Debugging

A lot of programming seems to involve getting rid of the mistakes you put in

If you get stuck:

- Try desk-checking your code
- Ask someone else to look at your code
- Use `print` statements (or `printNow`) to try and work out what is going on

Problem solving

Once the high level function is complete and working properly

Remember to test it in the labs

Double check the assignment requirements. Check again.

One more check..
(does it run with any file I might use, any width or height I might supply ?)

Keep the comments flowing!

Remember, the comments in a program are to explain to the reader what the code is meant to be doing

Don’t wait until the program is complete before adding the comments!

Add each comment before you write the code that it refers to.

That way, the comments remind you of the design, and help to keep you faithful to it

Tidying up at the end

When the program is doing what it was designed to do, it’s still not finished

Now’s the time to work through it line by line: removing any commented code that’s no longer needed;

Tidying up at the end

Add further comments where appropriate;

Replace repeated chunks of code with another function;

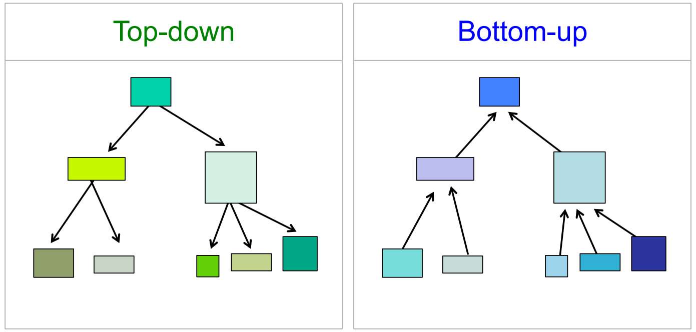
Revising bits of code to make them clearer;

Generally make it easier to read and understand, and therefore a better program.

Check spacing and naming standard

Program design

Actual program design often combines the two



Which approach is better?

There's no simple answer to whether it's better to design and write a program top-down or bottom-up

Top-down ensures the structure of the program, and is good if you know from the start what it's required to do

Which approach is better?

Bottom-up concentrates on the components, so it's possible that the structure will be harder to synthesise

Bottom-up is sometimes preferred by programmers when they can't grasp the overall design at the beginning

INFT1004

Introduction to Programming

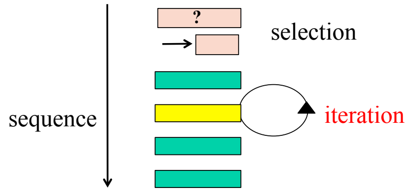
Module 7.3
Iteration – While Loop

Guzdial & Ericson - Third Edition – chapter 9
Guzdial & Ericson - Fourth (Global) Edition – chapter 10

Sequence, selection, iteration

Programming has three essential building blocks

Iteration determines which code to execute depending on specified conditions



Types of Loops

While loop — Tests some condition - If the condition is true it executes some statements
Repeats until the test condition is false

For loop — Repeats some statements a predetermined number of times

Nested loop — One loop inside another loop

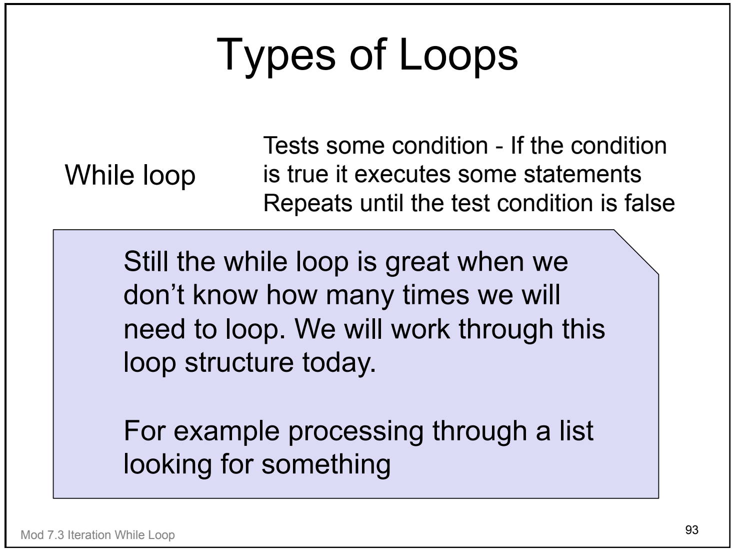
Types of Loops

We will focus mostly on the for loop in this course.

(Mostly we will know how many times we need to loop)

For loop — Repeats some statements a predetermined number of times

Nested loop — One loop inside another loop



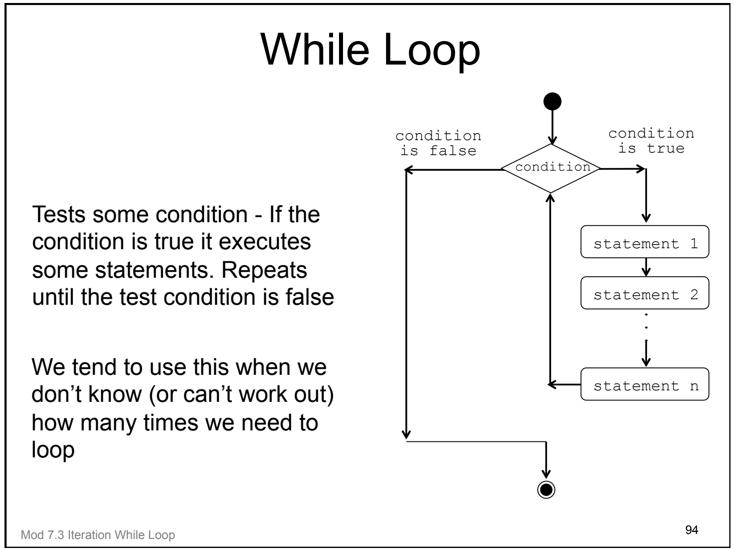
Types of Loops

While loop

Tests some condition - If the condition is true it executes some statements Repeats until the test condition is false

Still the while loop is great when we don't know how many times we will need to loop. We will work through this loop structure today.

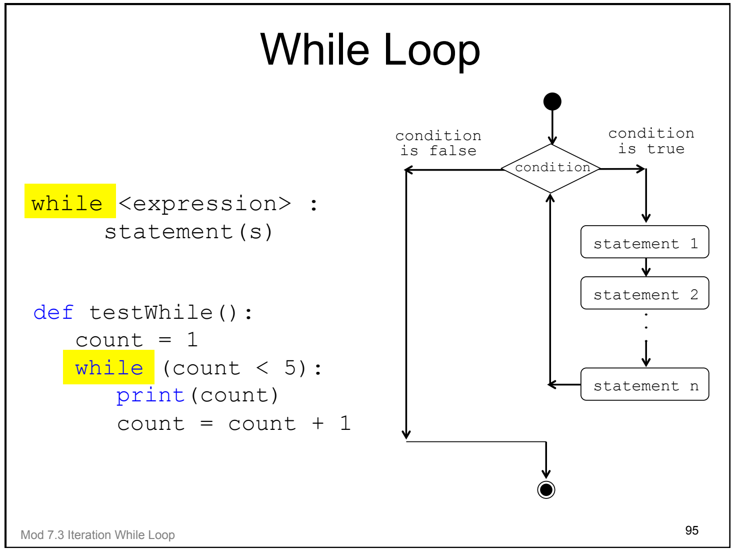
For example processing through a list looking for something



While Loop

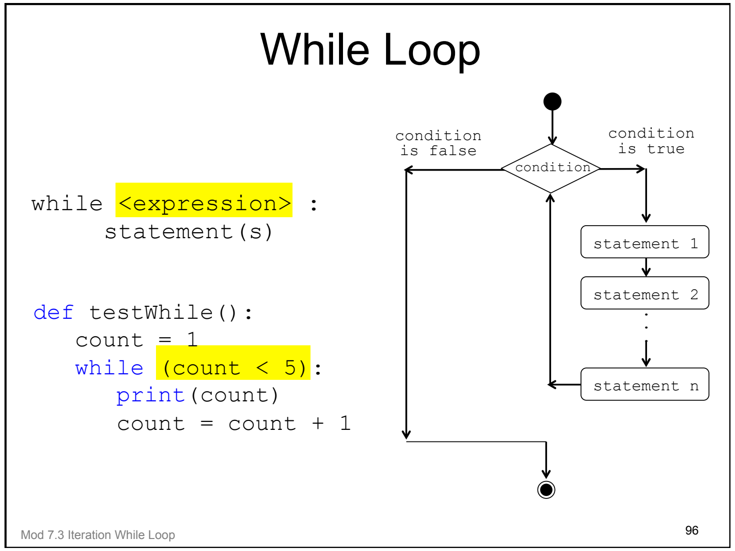
Tests some condition - If the condition is true it executes some statements. Repeats until the test condition is false

We tend to use this when we don't know (or can't work out) how many times we need to loop



While Loop

```
while <expression> :
      statement(s)

def testWhile():
   count = 1
   while (count < 5):
      print(count)
      count = count + 1
```



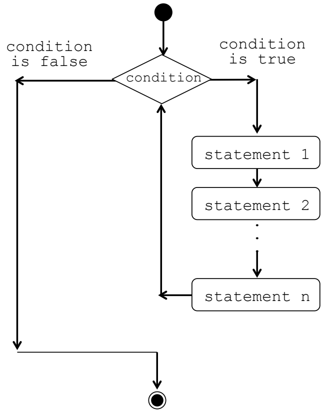
While Loop

```
while <expression> :
      statement(s)

def testWhile():
   count = 1
   while (count < 5):
      print(count)
      count = count + 1
```

While Loop

```
while <expression> :
      statement(s)
```

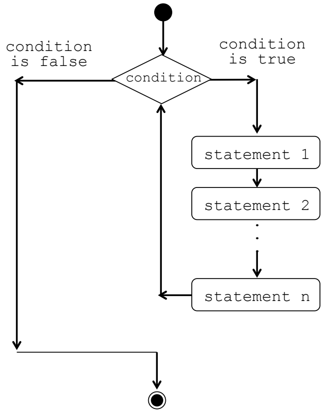
```
def testWhile():
    count = 1
    while (count < 5):
        print(count)
        count = count + 1
```



While Loop

```
while <expression> :
      statement(s)
```

```
def testWhile():
    count = 1
    while (count < 5):
        print(count)
        count = count + 1
```



Desk Check

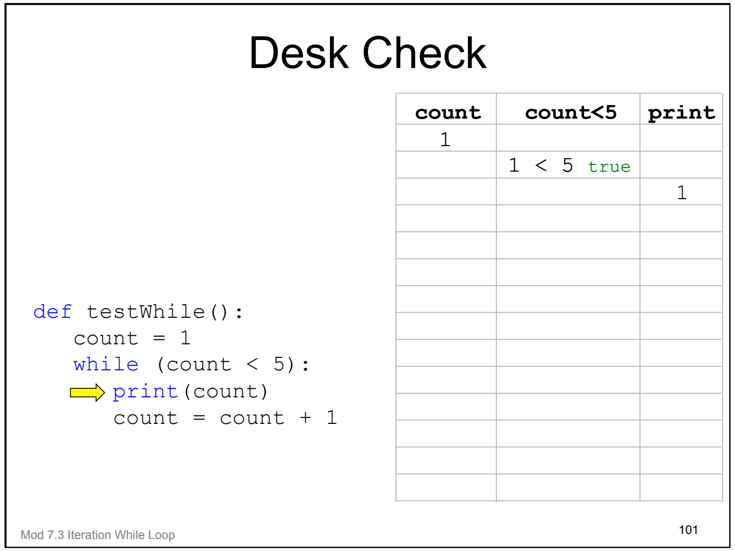
```
def testWhile():
⇨   count = 1
    while (count < 5):
        print(count)
        count = count + 1
```

count	count<5	print
1		

Desk Check

```
def testWhile():
    count = 1
⇨   while (count < 5):
        print(count)
        count = count + 1
```

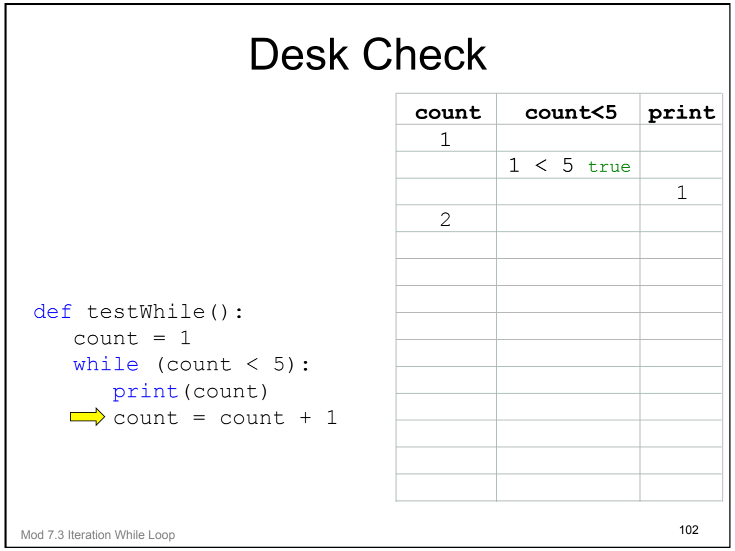
count	count<5	print
1		
	1 < 5 true	



Desk Check

count	count<5	print
1		
	1 < 5 true	
		1

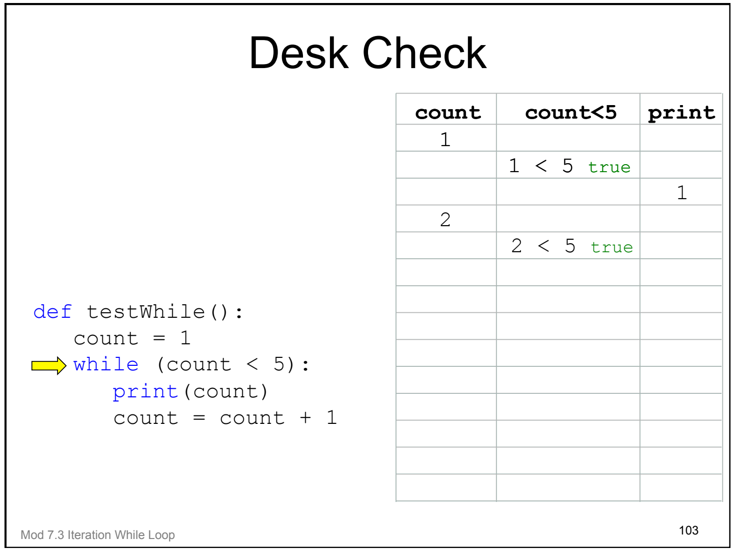
```
def testWhile():
    count = 1
    while (count < 5):
  ⇨     print(count)
        count = count + 1
```



Desk Check

count	count<5	print
1		
	1 < 5 true	
		1
2		

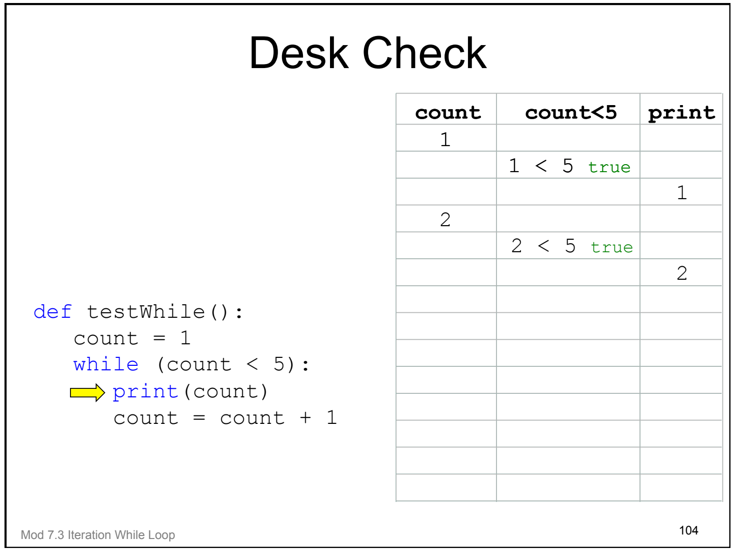
```
def testWhile():
    count = 1
    while (count < 5):
        print(count)
  ⇨     count = count + 1
```



Desk Check

count	count<5	print
1		
	1 < 5 true	
		1
2		
	2 < 5 true	

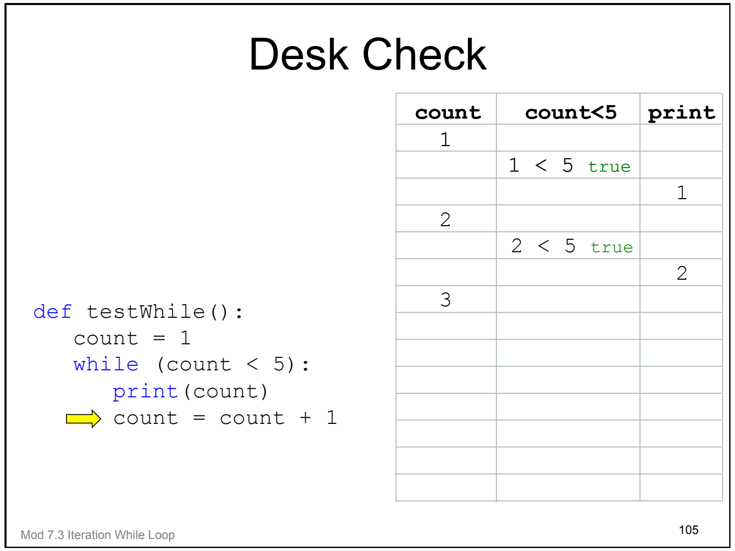
```
def testWhile():
    count = 1
⇨   while (count < 5):
        print(count)
        count = count + 1
```



Desk Check

count	count<5	print
1		
	1 < 5 true	
		1
2		
	2 < 5 true	
		2

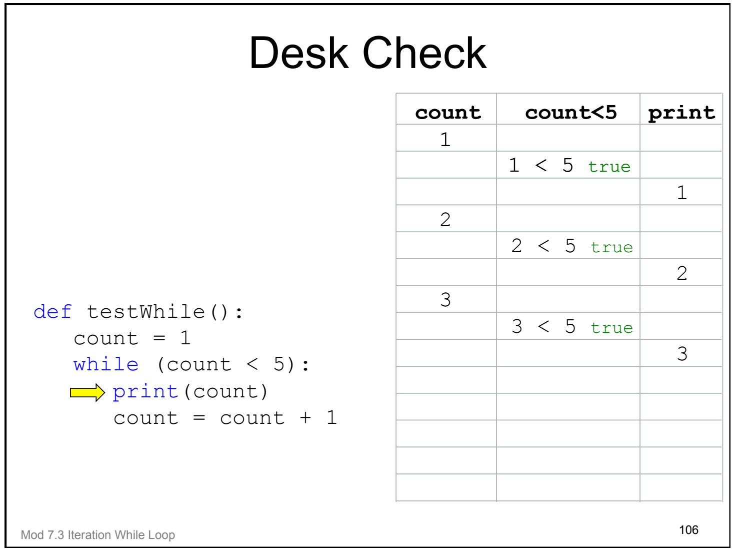
```
def testWhile():
    count = 1
    while (count < 5):
  ⇨     print(count)
        count = count + 1
```



Desk Check

```
def testWhile():
    count = 1
    while (count < 5):
        print(count)
  ⇨     count = count + 1
```

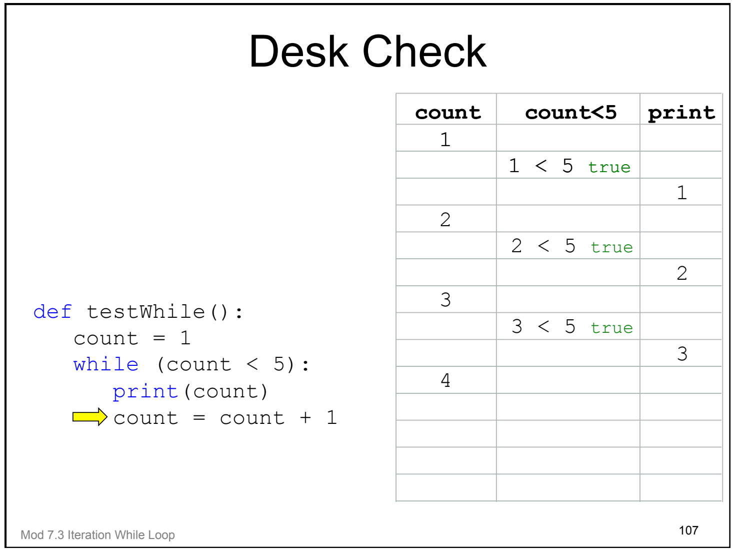
count	count<5	print
1		
	1 < 5 true	
		1
2		
	2 < 5 true	
		2
3		



Desk Check

```
def testWhile():
    count = 1
    while (count < 5):
  ⇨     print(count)
        count = count + 1
```

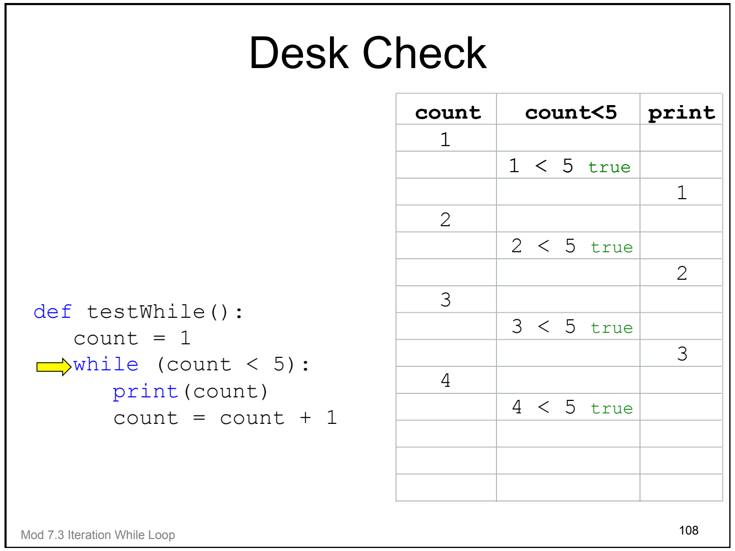
count	count<5	print
1		
	1 < 5 true	
		1
2		
	2 < 5 true	
		2
3		
	3 < 5 true	
		3



Desk Check

```
def testWhile():
    count = 1
    while (count < 5):
        print(count)
  ⇨     count = count + 1
```

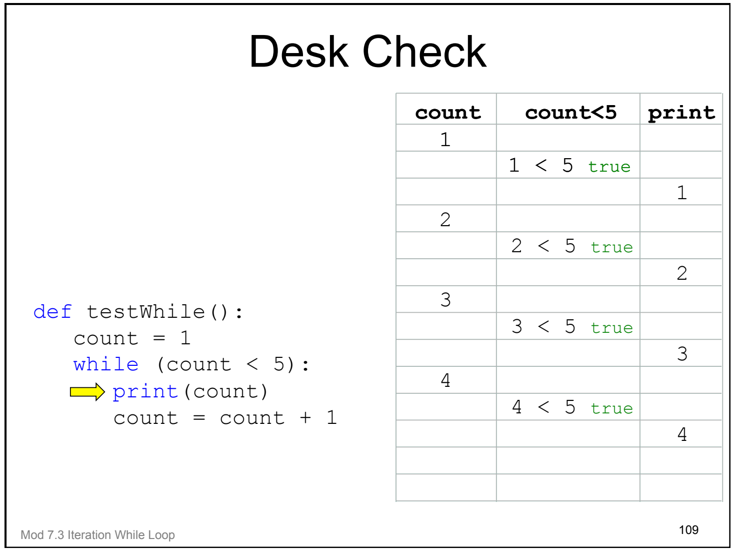
count	count<5	print
1		
	1 < 5 true	
		1
2		
	2 < 5 true	
		2
3		
	3 < 5 true	
		3
4		



Desk Check

```
def testWhile():
    count = 1
  ⇨ while (count < 5):
        print(count)
        count = count + 1
```

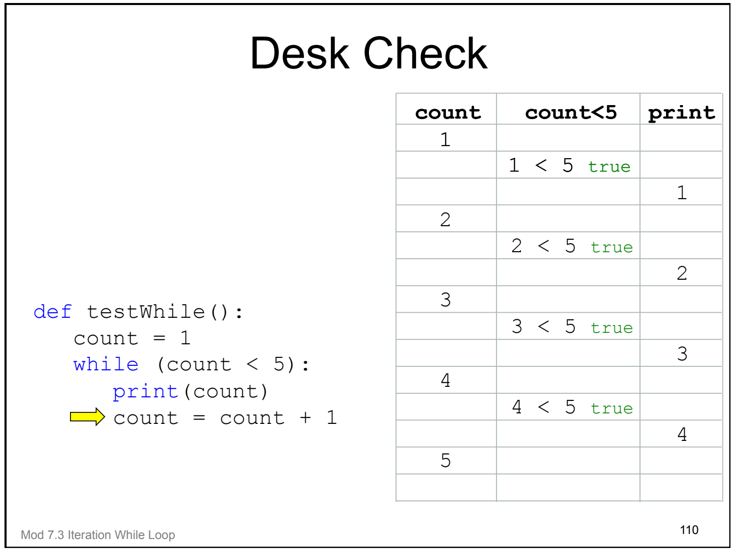
count	count<5	print
1		
	1 < 5 true	
		1
2		
	2 < 5 true	
		2
3		
	3 < 5 true	
		3
4		
	4 < 5 true	



Desk Check

```
def testWhile():
    count = 1
    while (count < 5):
  ⇨     print(count)
        count = count + 1
```

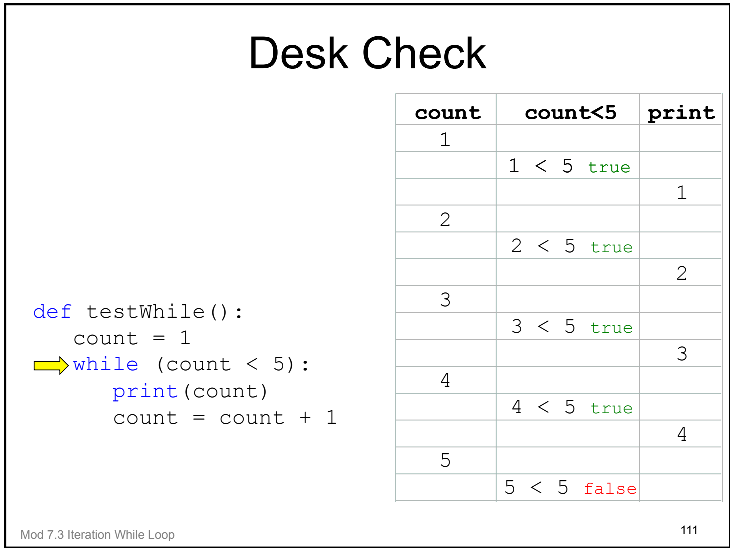
count	count<5	print
1		
	1 < 5 true	
		1
2		
	2 < 5 true	
		2
3		
	3 < 5 true	
		3
4		
	4 < 5 true	
		4



Desk Check

```
def testWhile():
    count = 1
    while (count < 5):
        print(count)
  ⇨     count = count + 1
```

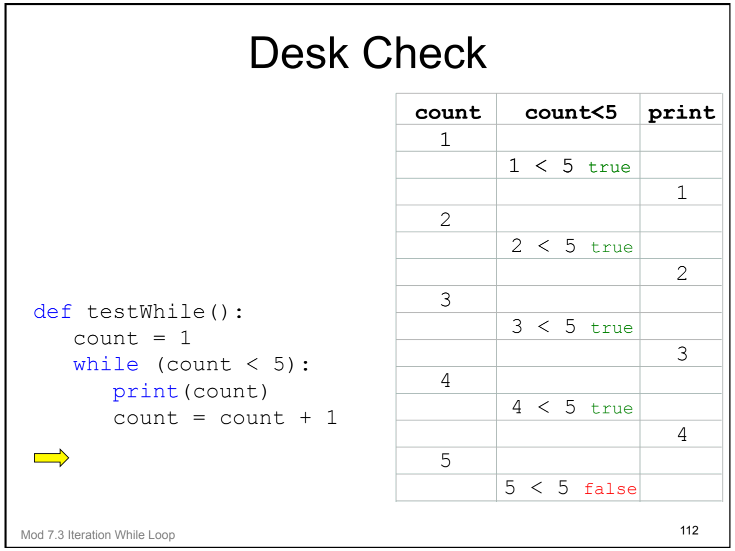
count	count<5	print
1		
	1 < 5 true	
		1
2		
	2 < 5 true	
		2
3		
	3 < 5 true	
		3
4		
	4 < 5 true	
		4
5		



Desk Check

```
def testWhile():
    count = 1
⇨   while (count < 5):
        print(count)
        count = count + 1
```

count	count<5	print
1		
	1 < 5 true	
		1
2		
	2 < 5 true	
		2
3		
	3 < 5 true	
		3
4		
	4 < 5 true	
		4
5		
	5 < 5 false	



Desk Check

```
def testWhile():
    count = 1
    while (count < 5):
        print(count)
        count = count + 1

⇨
```

count	count<5	print
1		
	1 < 5 true	
		1
2		
	2 < 5 true	
		2
3		
	3 < 5 true	
		3
4		
	4 < 5 true	
		4
5		
	5 < 5 false	

Desk Check

```
def testWhile():
   count = 1
   while (count < 5):
      print(count)
      count = count + 1
```

count	count<5	print
1		
	1 < 5 true	
		1
2		
	2 < 5 true	
		2
3		
	3 < 5 true	
		3
4		
	4 < 5 true	
		4
5		
	5 < 5 false	

Example 1

```
def testWhileMax():
   # This function uses While statement to print
   # the numbers from 1 to whatever number the user
   # selects

   max = requestInteger("Please enter a number:")
   count = 1

   while (count <= max):
     print(count)
     count = count + 1
```

Mod7 3 testIterationWhile.py

Example 2 - Find a red pixel

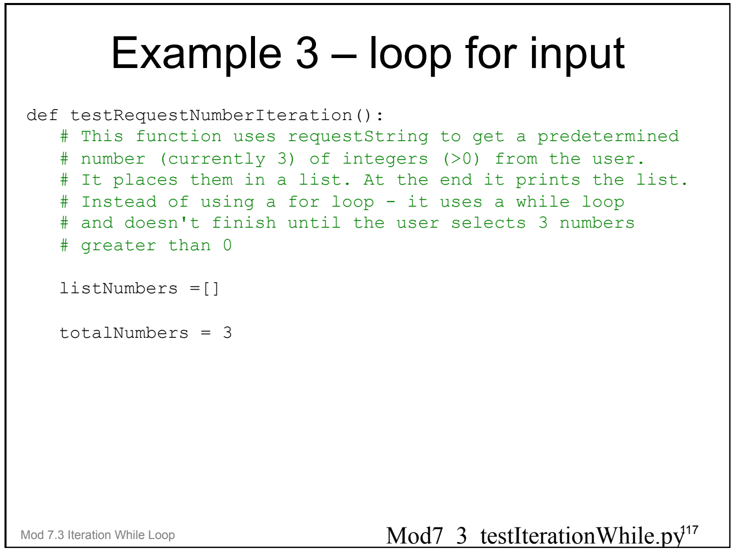
```
def redPixelInRow(image, rowNumber):
   # This function test for a pixel that is red (or almost red)
   # in a specified row of an image
   # it returns true if there is a redish pixel in the row
   # or false otherwise

   result = false #return value

   # set up a boolean loop variable - it becomes true
   # if a redish pixel is found
   redFound = false

   width = getWidth(image)
   x = 0 #start looking in first column

   while ((not redFound) and (x < width) ):
      pixel = getPixel(image, x, rowNumber)
      colourPixel = getColor(pixel)
      if (distance(colourPixel, red) < 100): #100 seems to work OK
         redFound = true #we have found a redish pixel
         result = true
      else:
         x = x + 1 #check the next column

   return result
```

Mod7 3 testIterationWhile.py

Test find a red pixel

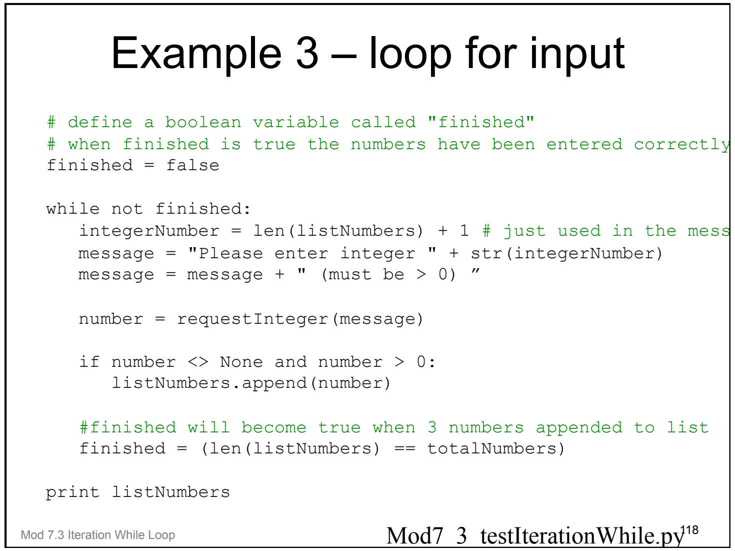
```
def testForRedPixel():
   file = pickAFile()    #try the redDoor.jpg
   picture = makePicture(file)

   # test row y = 137
   # with "redDoor.jpg there is a reddish pixel in this row)

   if redPixelInRow(picture, 137):
      print("red pixel found")
   else:
      print ("No red pixel found")

   # test row y = 450
   # with "redDoor.jpg there is no red pixel in this row)

   if redPixelInRow(picture, 450):
       print("red pixel found")
   else:
       print ("No red pixel found")
```

Mod7 3 testIterationWhile.py



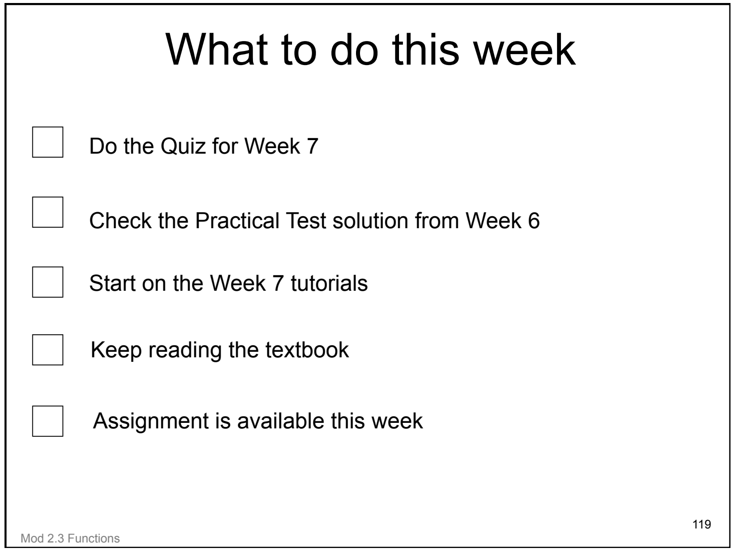
Example 3 – loop for input

```
def testRequestNumberIteration():
   # This function uses requestString to get a predetermined
   # number (currently 3) of integers (>0) from the user.
   # It places them in a list. At the end it prints the list.
   # Instead of using a for loop - it uses a while loop
   # and doesn't finish until the user selects 3 numbers
   # greater than 0

   listNumbers =[]

   totalNumbers = 3
```



Example 3 – loop for input

```
# define a boolean variable called "finished"
# when finished is true the numbers have been entered correctly
finished = false

while not finished:
   integerNumber = len(listNumbers) + 1 # just used in the mess
   message = "Please enter integer " + str(integerNumber)
   message = message + " (must be > 0) ”

   number = requestInteger(message)

   if number <> None and number > 0:
      listNumbers.append(number)

   #finished will become true when 3 numbers appended to list
   finished = (len(listNumbers) == totalNumbers)

print listNumbers
```



What to do this week

□ Do the Quiz for Week 7

□ Check the Practical Test solution from Week 6

□ Start on the Week 7 tutorials

□ Keep reading the textbook

□ Assignment is available this week