Assignment 1, Part A: Building Blocks

(19%, due 11:59pm Sunday, April 16th)

Overview

This is the first part of a two-part assignment. This part is worth 19% of your final grade for IFB104. Part B will be worth a further 6%. Part B is intended as a last-minute extension to the assignment, thereby testing the maintainability of your code, and the instructions for completing it will not be released until Week 7. Whether or not you complete Part B you will submit only one file, and receive only one assessment, for the whole 25% assignment.

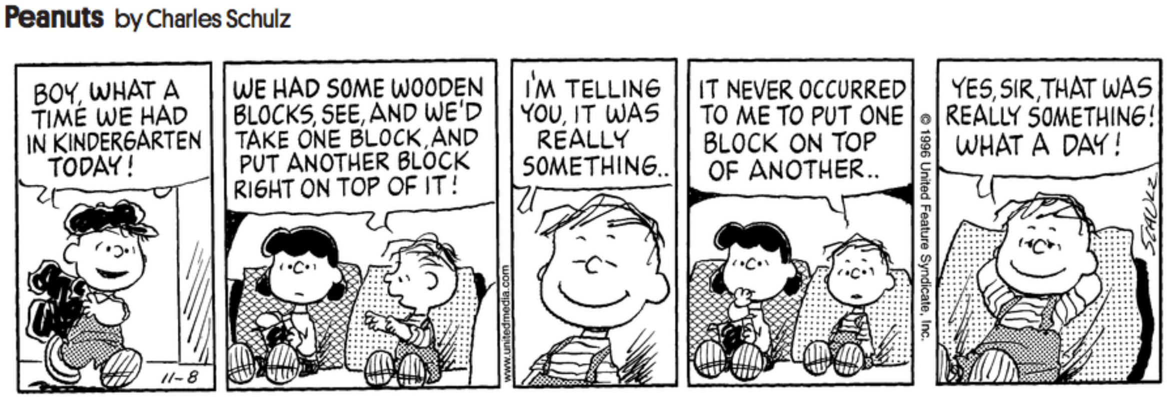
Motivation

One of the most basic functions of any IT system is to process a given data set to produce some form of human-readable output. This assignment requires you to produce a visual image by following instructions stored in a list. It tests your abilities to:

- Process lists of data values;
- Design a solution to a computational problem;
- Display information in a visual form; and
- Produce reusable code.

Goal

Building blocks are a familiar, traditional toy for pre-schoolers, as illustrated by the following *Peanuts* strip (8/11/96) featuring Lucy and her youngest brother 'Rerun' (no, it's *not* Linus!).


Peanuts by Charles Schulz
BOY, WHAT A TIME WE HAD IN KINDERGARTEN TODAY!
11-8
WE HAD SOME WOODEN BLOCKS, SEE, AND WE'D TAKE ONE BLOCK, AND PUT ANOTHER BLOCK RIGHT ON TOP OF IT!
www.unitedmedia.com
I'M TELLING YOU, IT WAS REALLY SOMETHING..
IT NEVER OCCURRED TO ME TO PUT ONE BLOCK ON TOP OF ANOTHER..
© 1996 United Feature Syndicate, Inc.
YES, SIR, THAT WAS REALLY SOMETHING! WHAT A DAY!
SCHULZ

In this assignment you are required to develop a Python program which processes data stored in a list to display a specific arrangement of up to four blocks. Each block must have a distinct part of an overall image on its face. When the blocks are stacked correctly the entire image must be produced. Blocks can be stacked up to two layers deep and can be placed in any of four possible orientations, upright, upside down, turned left and turned right.

To complete this assignment you will need to use basic Python features and the Turtle graphics module. You must also design four blocks which, when arranged in the correct order, produce a single picture. The picture must be non-trivial, and must span all four pieces, but otherwise you have a free choice of what to draw, e.g.,

- cartoon, game or science fiction characters,
- household objects,
- corporate or sporting logos,
- buildings or vehicles,
- animals or pets,
- landscapes, etc.

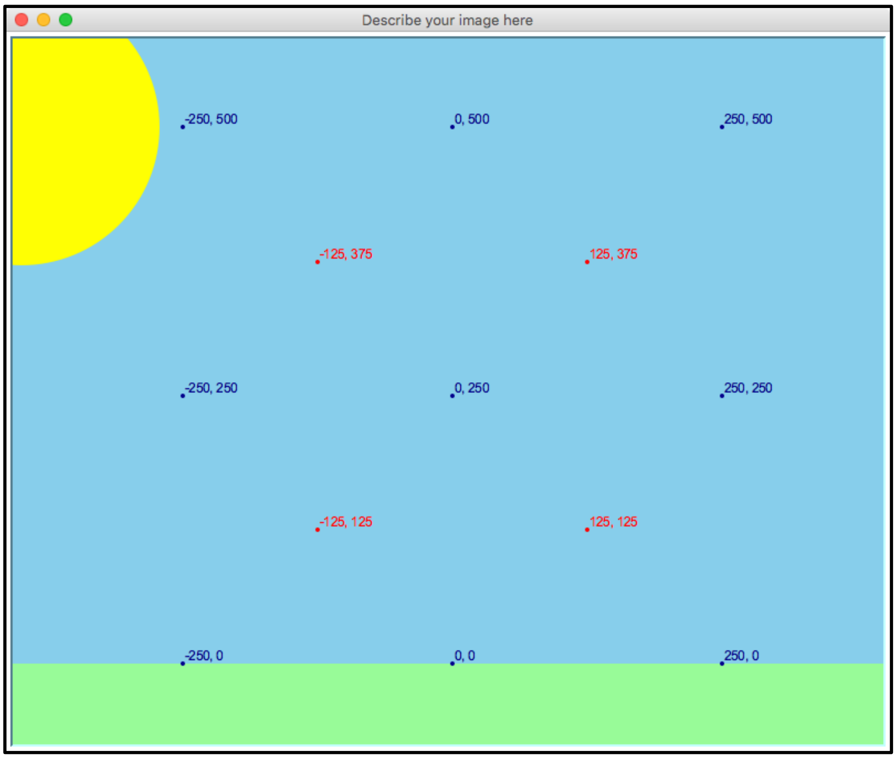
To stack the blocks you must develop your code so that each individual block can be drawn in any of four different locations on the screen and in any of four different orientations.

Resources provided

A template Python program, `building_blocks.py`, is provided with these instructions. This template:

- Creates a drawing canvas and displays a simple background image; and
- Optionally draws the cartesian coordinates at which the blocks must be drawn.

When you first run the program it will create the following drawing canvas.



The red dots mark the centres of the locations where the blocks must be drawn and the dark blue dots mark their corners. Each block must be a square measuring 250 × 250 pixels. The coordinates are shown to help you position your images while you develop your program. When you have completed your solution you can turn them off by changing some parameters in the template's main program. Notice that in this canvas the "home" coordinate (0, 0) is at the bottom, marking the ground on which the blocks will be stacked.

The provided template file also contains several data sets, in the form of lists, which specify how you must arrange the blocks when drawing them. These 'arrangment' lists each contain instructions in four parts:

- The identity of the block to draw, from '`Block A`' to '`Block D`'.
- The place where the block must be drawn, either '`Top left`', '`Top right`', '`Bottom left`' or '`Bottom right`'. Notice that the lists are ordered so that each block is placed either on the ground or on top of an existing one, never in mid air.
- The orientation of the block, either '`Up`', '`Down`', '`Left`' or '`Right`', denoting upright, upside down, on its left side or on its right side, respectively.
- A mystery value, either '`O`' or '`X`', whose purpose will be revealed only in the second part of the assigment.

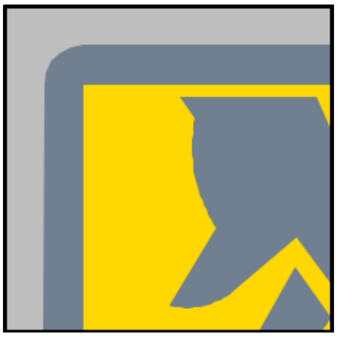
The template file also contains a dummy function definition, `stack_blocks`. Your task is to complete this function definition so that it draws the blocks at the positions and orientations specified by any of the given data sets (or any other similar data sets in the same format). When arranged the right way your blocks must produce a single, complete picture.

Illustrative example

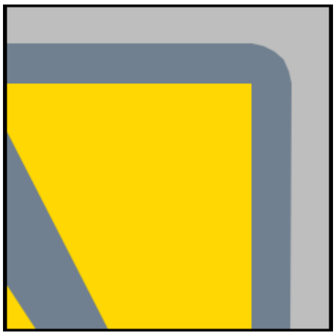
Here we present a sample solution to illustrate the requirement. (You should *not* copy our example. Develop your own idea! Be imaginative!)

Firstly you will need to design each of your blocks. In our case we have created four blocks, as shown below in their upright orientation.

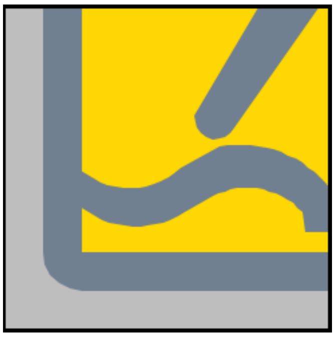
Block A:



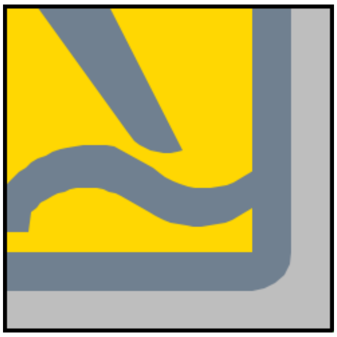
Block B:



Block C:



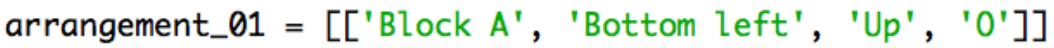
Block D:



Each of the four blocks must contain a non-trivial image clearly distinct from all the others, regardless of their orientation, i.e., no block can be identical to another block when rotated. Also, the images must be asymmetric so that it's easy to tell which way the block is pointing. When arranged correctly the four blocks must combine precisely to produce a single composite picture.

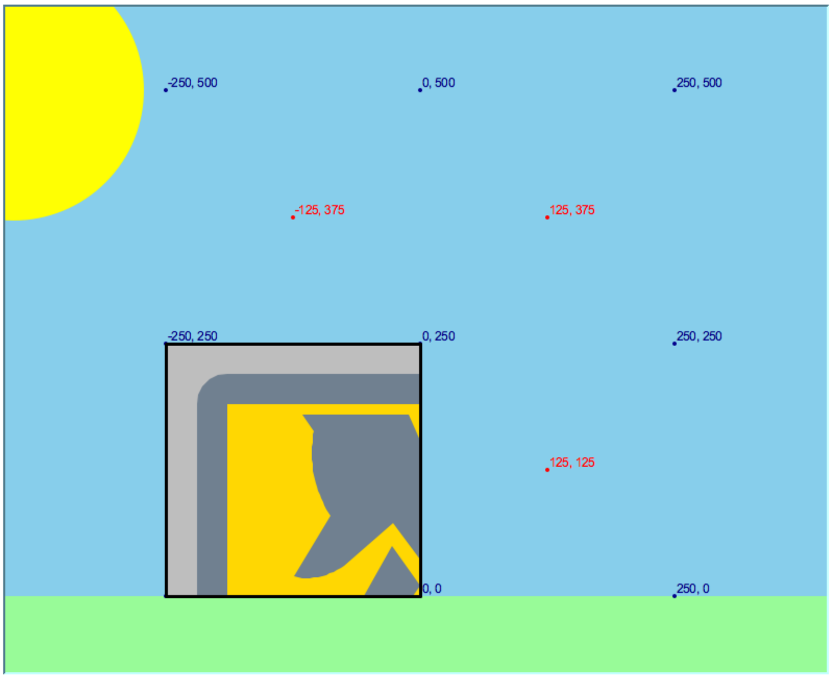
Although it's difficult to specify the artistic requirements for this assignment, given the wide range of images that could be chosen, it's expected the assembled picture will involve a number of different shapes and colours and must be immediately recognisable. Simple geometric shapes would not be considered sufficiently challenging. Similarly, four unrelated images, one per block, would be unacceptable. A good example of an image that would *not* be considered suitable for this assignment is the *Commonwealth Bank* logo, which is largely just a yellow square. Even apart from the logo's simplicity, the top left and bottom left blocks would be identical when rotated in this instance.

In the Python template file there are several data sets in the form of lists, each describing a particular arrangement of one or more blocks. For instance, one of the simplest such lists is as follows:



```
arrangement_01 = [['Block A', 'Bottom left', 'Up', 'O']]
```

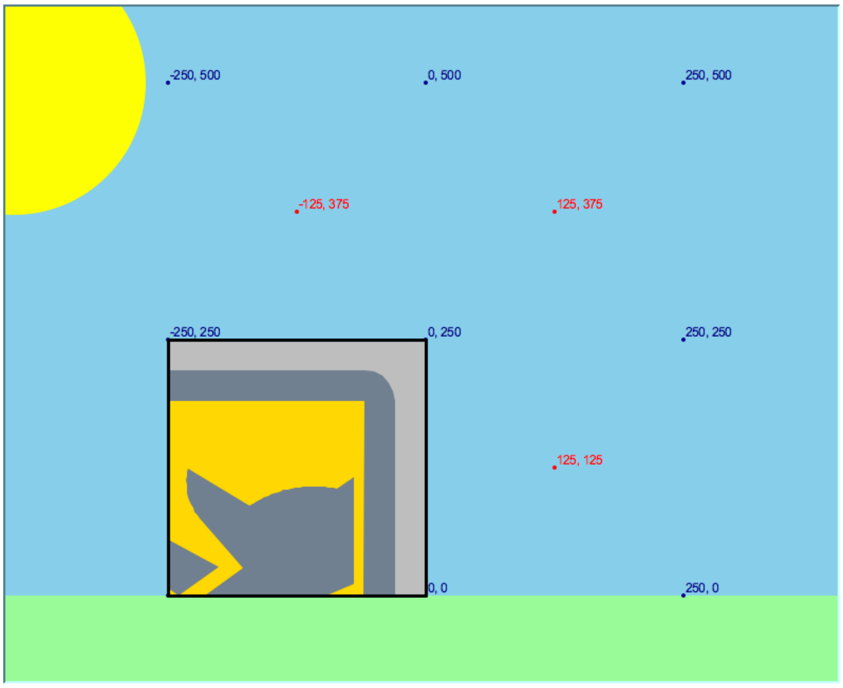
This list tells us that we are required to position Block A in the bottom left location, in its upright orientation. When our `stack_blocks` function is called with this list as the argument it produces the image displayed overleaf.



Another such data set is as follows.

```
arrangement_12 = [['Block A', 'Bottom left', 'Right', 'O']]
```

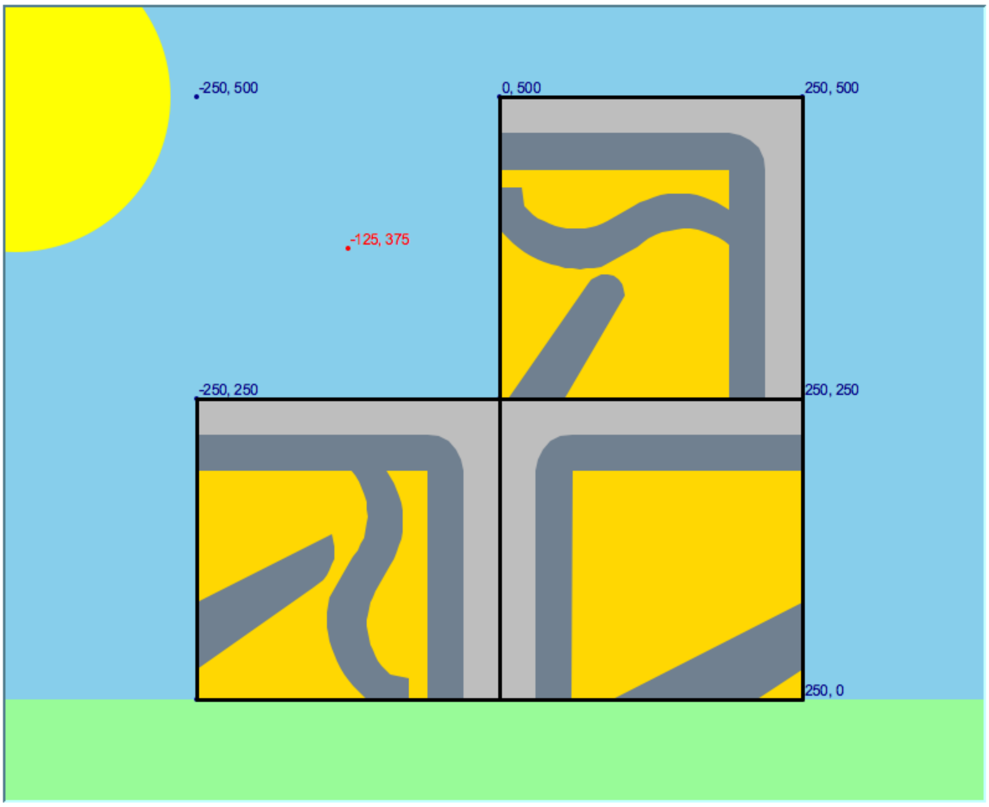
In this case Block A is to be drawn in the same location but lying on its right side, which produces the following image.



Other data sets require multiple blocks to be stacked. For instance, the following data set specifies that three blocks must be shown, Block B in the bottom right location lying on its left side, Block D in the bottom left also on its left side, and Block C stacked in the top right position and upside down.

```
arrangement_42 = [['Block B', 'Bottom right', 'Left', 'O'],
                  ['Block D', 'Bottom left', 'Left', 'O'],
                  ['Block C', 'Top right', 'Down', 'O']]
```

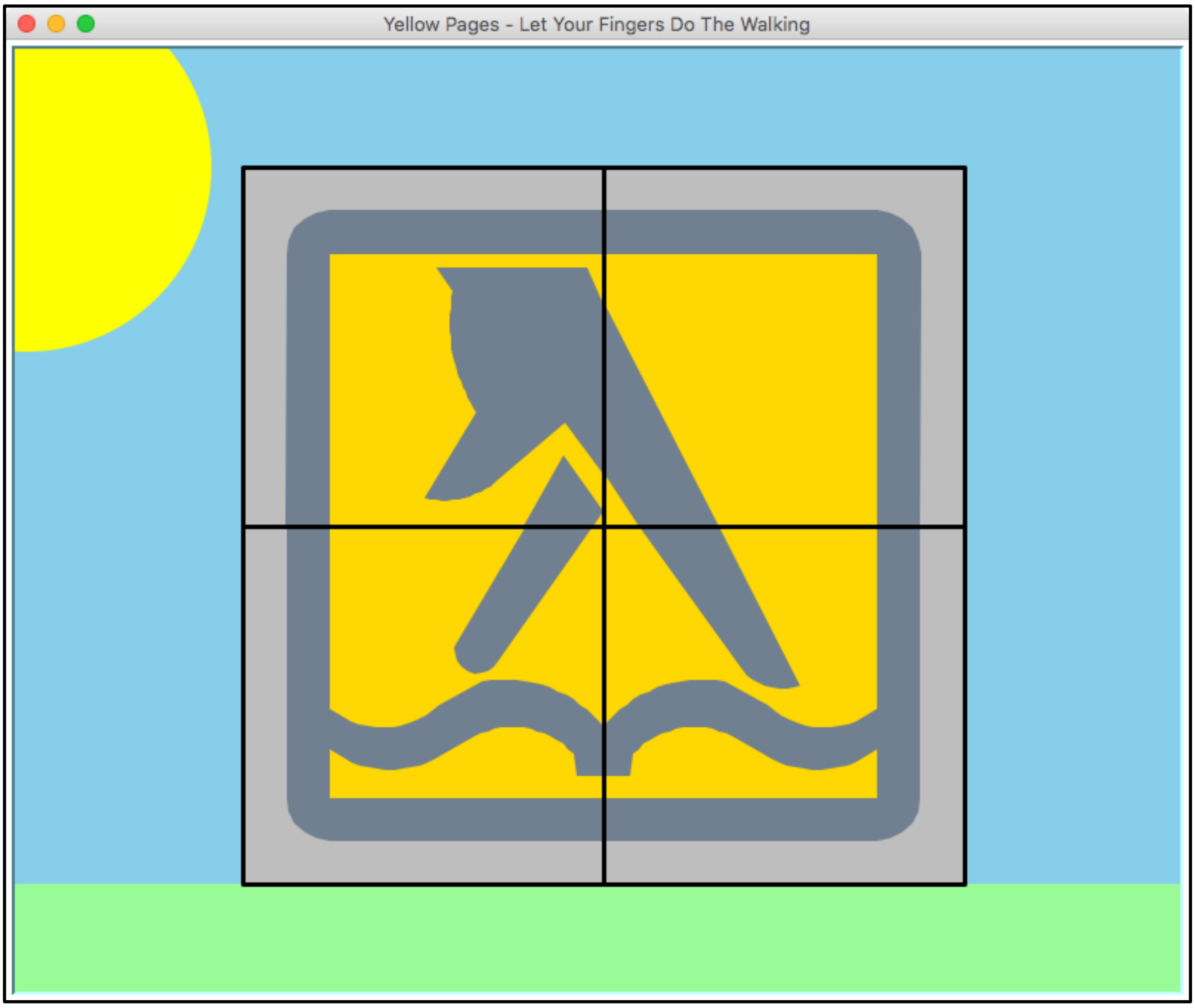
The resulting image in this case is as shown below, for our particular collection of blocks.



The final data sets require all four blocks to be drawn, in various locations and orientations. The very last data set provided is as follows.

```
arrangement_99 = [['Block C', 'Bottom left', 'Up', 'O'],
                  ['Block D', 'Bottom right', 'Up', 'O'],
                  ['Block A', 'Top left', 'Up', 'O'],
                  ['Block B', 'Top right', 'Up', 'O']]
```

Most importantly, this arrangement stacks all four blocks upright and in an order that completes the image, as shown overleaf. In this case it at last reveals that our chosen overall picture is the well-known logo for the *Yellow Pages* telephone directory. So that this final image looks nice we have turned off the coordinate markings below. We have also given the window a title, describing the image.



Requirements and marking guide

To complete this task you are required to extend the provided `building_blocks.py` Python file by completing function `stack_blocks` so that it can draw blocks at the places and orientations specified by a data set provided as its single parameter. Your code must work for all the supplied "arrangement" data sets and any other data set in the same format.

Your submitted solution will consist of a single Python file, and must satisfy the following criteria. Percentage marks available are as shown.

1. **Drawing four distinct building blocks (4%).** Your program must be able to draw four distinct blocks, each containing part of a single overall picture. Each block must be a 250 × 250 pixel square and must be of a reasonable degree of complexity, involving multiple shapes and colours. The entire square must be filled in with colour. Each block must be clearly different from all the others, regardless of their orientation, and the image on each individual block must be asymmetric so that it is easy to tell which way it is oriented.
2. **Creating a single picture in four parts (2%).** When arranged correctly, as per the final data set provided, the separate parts of the image must align perfectly to produce a single, clearly recognisable picture. The title of the drawing canvas must describe the

picture drawn. (NB: If your solution is incapable of drawing all the blocks in their final arrangement we cannot assess this criterion and must award zero for it.)

3. **Relocating blocks (3%)**. Your code must be capable of drawing each of the four blocks at any of the four marked places, as per the coordinates (optionally) drawn on the screen by the provided template. The blocks must be sufficiently different that it is easy to distinguish which one is being drawn in each location. The images on the blocks must preserve their integrity no matter where they are drawn and must fit perfectly into the marked places. No extraneous lines or shapes should be drawn outside the boundaries of the blocks. Your solution for relocating the blocks must work for all of the provided data sets and any other data sets in the same format. (You cannot "hardwire" your code for specific data sets and you may not change the data sets provided.)
4. **Rotating blocks (6%)**. Your code must be capable of drawing each of the four blocks in any of the four possible orientations, upright, upside down, on its left side or on its right side. The images on each block must be suffiently asymmetric that it is easy to tell one orientation from another. The images of the blocks must preserve their integrity no matter how they are oriented and must fit perfectly into a 250 x 250 pixel space regardless of their orientation. No extraneous lines or shapes may be drawn outside the boundaries of the blocks. Your solution for reorienting the blocks must work for all of the provided data sets and any other data sets in the same format. (You cannot "hardwire" your code for specific data sets and you may not change the data sets provided.)
5. **Code quality and presentation (4%)**. Your program code, for both Parts A and B of the assignment, must be presented in a professional manner. See the coding guidelines in the *IFB104 Code Presentation Guide* (on Blackboard under *Assessment*) for suggestions on how to achieve this. In particular, given the obscure and repetitive nature of the code needed to draw complex images using Turtle graphics, each significant code segment must be clearly commented to say what it does, e.g., "Draw index finger", "Draw left side of book", etc.
6. ***Extra features (6%)***. *Part B of this assignment will require you to make a 'last-minute extension' to your solution. The instructions for Part B will not be released until just before the final deadline for Assignment 1.*

You must complete the task using basic Turtle graphics and maths functions only. You may not import any additional modules or files into your program other than those already included in the given `building_blocks.py` template. In particular, you may not import any image files for use in creating your blocks.

Finally, you are *not* required to copy the example shown in this document. Instead you are strongly encouraged to be creative and to choose your own picture that interests you personally.

Artistic merit

You will not be assessed on the artistic merit of your solution, only the ability to create a recognisable picture in four parts. However, a "Hall of Fame" containing the solutions considered the most artistic or ambitious by the assignment markers will be created on Blackboard. (Sadly, additional marks will not be awarded to the winners, only kudos.)

Development hints

- This can be a time-consuming task, so you are strongly encouraged to design your blocks carefully before developing any program code.
- The hardest part of this assignment is the need to allow the pieces to be drawn in different locations and orientations. You therefore need to devise a way of drawing each piece so that you can either (a) make all drawing moves *relative* to the starting position and orientation (e.g., using Turtle's `forward`, `left` and `right` commands) or (b) by calculating *absolute* positions for each drawing move (e.g., using Turtle's `goto` command) in terms of a given position and orientation, using trigonometric functions.
- If you are unable to complete the whole task, just submit whatever parts you can get working. You will receive partial marks for incomplete solutions.
- To help you debug your code we have provided several data sets, numbered 1 to 4 and 10 to 21, which draw just one piece at a time. You can use these to help create the code for each block separately.
- Part B of this assignment will require you to change your solution slightly in a short space of time. You are therefore encouraged to keep code maintainability in mind while developing your solution to Part A. Make sure your code is neat and well-commented so that you will find it easy to modify when the instructions for Part B are released.

Deliverable

You must develop your solution by completing and submitting the provided Python file `building_block.py` as follows.

1. Complete the "statement" at the beginning of the Python file to confirm that this is your own individual work by inserting your name and student number in the places indicated. *We will assume that submissions without a completed statement are not your own work!*
2. Complete your solution by developing Python code to replace the dummy `stack_blocks` function. You must complete your solution using only the modules already imported by the provided template. You may *not* use or import any other modules to complete this program. In particular, you may *not* import any image files into your solution.
3. Submit *a single Python file* containing your solution for marking. Do *not* submit an archive (e.g., in 'zip' or 'rar' formats) containing several files. Only a single file will be accepted, so you cannot accompany your solution with other files or pre-defined images. **Do not submit any other files!** **Submit only a single file!**

Apart from working correctly your program code must be well-presented and easy to understand, thanks to (sparse) commenting that explains the *purpose* of significant code segments and *helpful* choices of variable and function names. *Professional presentation* of your code will be taken into account when marking this assignment.

If you are unable to solve the whole problem, submit whatever parts you can get working. You will receive *partial marks for incomplete solutions*.

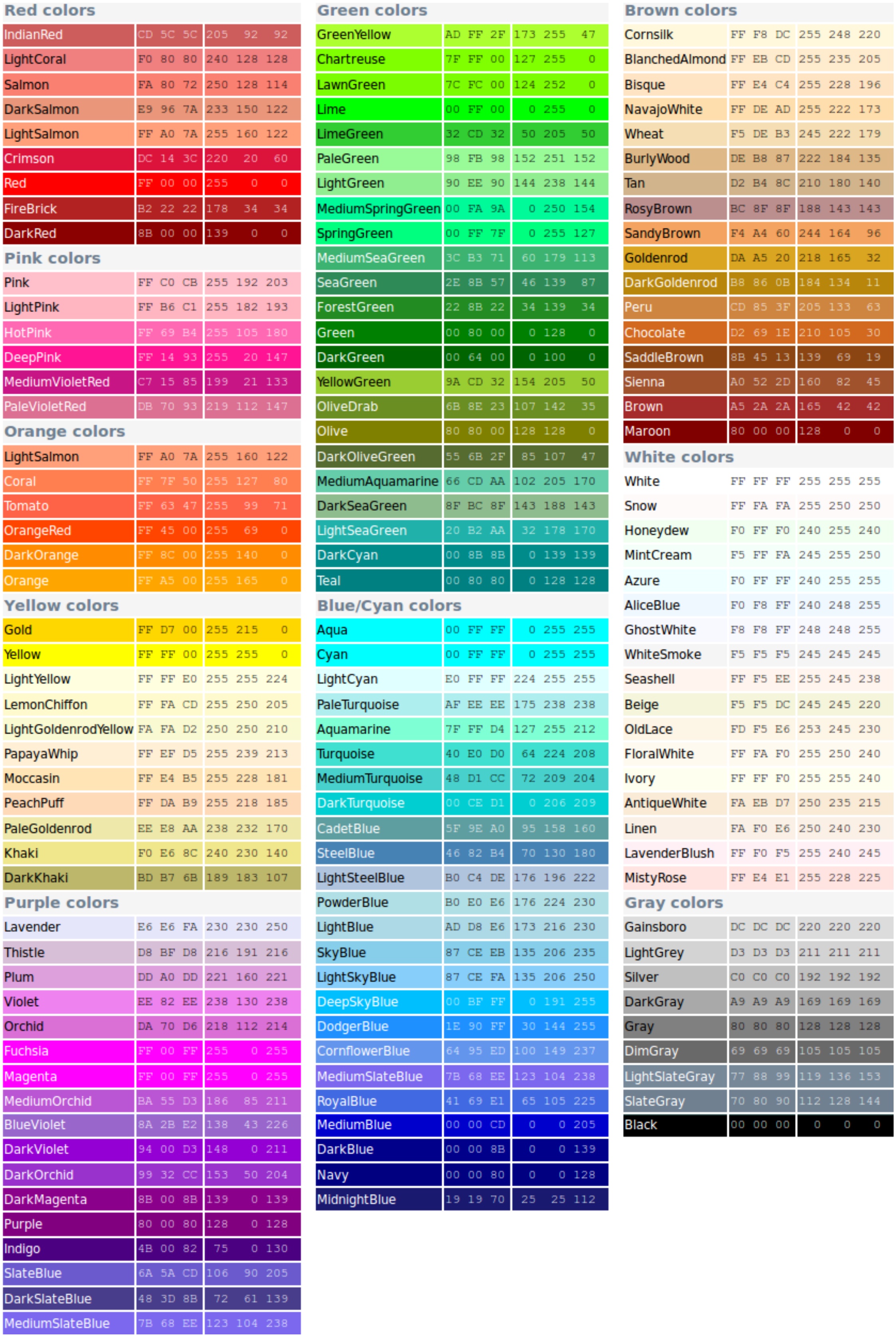
Plagiarism

This is an individual assessment item. All files submitted will be subjected to software plagiarism analysis using the MoSS system (http://theory.stanford.edu/~aiken/moss/). Serious violations of the university's policies regarding plagiarism will be forwarded to the Science and Engineering Faculty's Academic Misconduct Committee for trial.

How to submit your solution

A link will be available on Blackboard under *Assessment* for uploading your solution file before the deadline (11:59pm Sunday, April 16th). You can *submit as many drafts of your solution as you like*. You are strongly encouraged to *submit draft solutions* before the deadline. Students who encounter problems uploading their Python files to Blackboard should contact the *IT Helpdesk* (ithelpdesk@qut.edu.au; 3138 4000) for assistance and advice. **Teaching staff will *not* be available to help you after work hours, on weekends or on public holidays, so make sure you have submitted a draft version before close-of-business on the last working day before the deadline.**

Appendix: Some standard Turtle graphics colours



Red colors

IndianRed	CD 5C 5C	205 92 92
LightCoral	F0 80 80	240 128 128
Salmon	FA 80 72	250 128 114
DarkSalmon	E9 96 7A	233 150 122
LightSalmon	FF A0 7A	255 160 122
Crimson	DC 14 3C	220 20 60
Red	FF 00 00	255 0 0
FireBrick	B2 22 22	178 34 34
DarkRed	8B 00 00	139 0 0

Pink colors

Pink	FF C0 CB	255 192 203
LightPink	FF B6 C1	255 182 193
HotPink	FF 69 B4	255 105 180
DeepPink	FF 14 93	255 20 147
MediumVioletRed	C7 15 85	199 21 133
PaleVioletRed	DB 70 93	219 112 147

Orange colors

LightSalmon	FF A0 7A	255 160 122
Coral	FF 7F 50	255 127 80
Tomato	FF 63 47	255 99 71
OrangeRed	FF 45 00	255 69 0
DarkOrange	FF 8C 00	255 140 0
Orange	FF A5 00	255 165 0

Yellow colors

Gold	FF D7 00	255 215 0
Yellow	FF FF 00	255 255 0
LightYellow	FF FF E0	255 255 224
LemonChiffon	FF FA CD	255 250 205
LightGoldenrodYellow	FA FA D2	250 250 210
PapayaWhip	FF EF D5	255 239 213
Moccasin	FF E4 B5	255 228 181
PeachPuff	FF DA B9	255 218 185
PaleGoldenrod	EE E8 AA	238 232 170
Khaki	F0 E6 8C	240 230 140
DarkKhaki	BD B7 6B	189 183 107

Purple colors

Lavender	E6 E6 FA	230 230 250
Thistle	D8 BF D8	216 191 216
Plum	DD A0 DD	221 160 221
Violet	EE 82 EE	238 130 238
Orchid	DA 70 D6	218 112 214
Fuchsia	FF 00 FF	255 0 255
Magenta	FF 00 FF	255 0 255
MediumOrchid	BA 55 D3	186 85 211
BlueViolet	8A 2B E2	138 43 226
DarkViolet	94 00 D3	148 0 211
DarkOrchid	99 32 CC	153 50 204
DarkMagenta	8B 00 8B	139 0 139
Purple	80 00 80	128 0 128
Indigo	4B 00 82	75 0 130
SlateBlue	6A 5A CD	106 90 205
DarkSlateBlue	48 3D 8B	72 61 139
MediumSlateBlue	7B 68 EE	123 104 238

Green colors

GreenYellow	AD FF 2F	173 255 47
Chartreuse	7F FF 00	127 255 0
LawnGreen	7C FC 00	124 252 0
Lime	00 FF 00	0 255 0
LimeGreen	32 CD 32	50 205 50
PaleGreen	98 FB 98	152 251 152
LightGreen	90 EE 90	144 238 144
MediumSpringGreen	00 FA 9A	0 250 154
SpringGreen	00 FF 7F	0 255 127
MediumSeaGreen	3C B3 71	60 179 113
SeaGreen	2E 8B 57	46 139 87
ForestGreen	22 8B 22	34 139 34
Green	00 80 00	0 128 0
DarkGreen	00 64 00	0 100 0
YellowGreen	9A CD 32	154 205 50
OliveDrab	6B 8E 23	107 142 35
Olive	80 80 00	128 128 0
DarkOliveGreen	55 6B 2F	85 107 47
MediumAquamarine	66 CD AA	102 205 170
DarkSeaGreen	8F BC 8F	143 188 143
LightSeaGreen	20 B2 AA	32 178 170
DarkCyan	00 8B 8B	0 139 139
Teal	00 80 80	0 128 128

Blue/Cyan colors

Aqua	00 FF FF	0 255 255
Cyan	00 FF FF	0 255 255
LightCyan	E0 FF FF	224 255 255
PaleTurquoise	AF EE EE	175 238 238
Aquamarine	7F FF D4	127 255 212
Turquoise	40 E0 D0	64 224 208
MediumTurquoise	48 D1 CC	72 209 204
DarkTurquoise	00 CE D1	0 206 209
CadetBlue	5F 9E A0	95 158 160
SteelBlue	46 82 B4	70 130 180
LightSteelBlue	B0 C4 DE	176 196 222
PowderBlue	B0 E0 E6	176 224 230
LightBlue	AD D8 E6	173 216 230
SkyBlue	87 CE EB	135 206 235
LightSkyBlue	87 CE FA	135 206 250
DeepSkyBlue	00 BF FF	0 191 255
DodgerBlue	1E 90 FF	30 144 255
CornflowerBlue	64 95 ED	100 149 237
MediumSlateBlue	7B 68 EE	123 104 238
RoyalBlue	41 69 E1	65 105 225
MediumBlue	00 00 CD	0 0 205
DarkBlue	00 00 8B	0 0 139
Navy	00 00 80	0 0 128
MidnightBlue	19 19 70	25 25 112

Brown colors

Cornsilk	FF F8 DC	255 248 220
BlanchedAlmond	FF EB CD	255 235 205
Bisque	FF E4 C4	255 228 196
NavajoWhite	FF DE AD	255 222 173
Wheat	F5 DE B3	245 222 179
BurlyWood	DE B8 87	222 184 135
Tan	D2 B4 8C	210 180 140
RosyBrown	BC 8F 8F	188 143 143
SandyBrown	F4 A4 60	244 164 96
Goldenrod	DA A5 20	218 165 32
DarkGoldenrod	B8 86 0B	184 134 11
Peru	CD 85 3F	205 133 63
Chocolate	D2 69 1E	210 105 30
SaddleBrown	8B 45 13	139 69 19
Sienna	A0 52 2D	160 82 45
Brown	A5 2A 2A	165 42 42
Maroon	80 00 00	128 0 0

White colors

White	FF FF FF	255 255 255
Snow	FF FA FA	255 250 250
Honeydew	F0 FF F0	240 255 240
MintCream	F5 FF FA	245 255 250
Azure	F0 FF FF	240 255 255
AliceBlue	F0 F8 FF	240 248 255
GhostWhite	F8 F8 FF	248 248 255
WhiteSmoke	F5 F5 F5	245 245 245
Seashell	FF F5 EE	255 245 238
Beige	F5 F5 DC	245 245 220
OldLace	FD F5 E6	253 245 230
FloralWhite	FF FA F0	255 250 240
Ivory	FF FF F0	255 255 240
AntiqueWhite	FA EB D7	250 235 215
Linen	FA F0 E6	250 240 230
LavenderBlush	FF F0 F5	255 240 245
MistyRose	FF E4 E1	255 228 225

Gray colors

Gainsboro	DC DC DC	220 220 220
LightGrey	D3 D3 D3	211 211 211
Silver	C0 C0 C0	192 192 192
DarkGray	A9 A9 A9	169 169 169
Gray	80 80 80	128 128 128
DimGray	69 69 69	105 105 105
LightSlateGray	77 88 99	119 136 153
SlateGray	70 80 90	112 128 144
Black	00 00 00	0 0 0

Assignment 1, Part B: Lost Blocks

(6%, due 11:59pm Monday, April 17th)

Overview

This is the second part of a two-part assignment. This part is worth 6% of your final grade for IFB104. Part A which preceded it was worth 19%. This part is intended as a last-minute extension to the assignment, thereby testing the maintainability of your code from Part A and your ability to work under time presssure. If you have a neat, clear solution to Part A you will find completing Part B much easier. For the whole assignment you will submit only one file, containing your combined solution to both Parts A and B, and you will receive one grade for the whole 25% assignment.

Motivation

One of the most common tasks in "Building IT Systems" is modifying some existing code. In practice, computer programs are written only once but are subsequently modified and extended *many* times during their operational lifetime. Code changes may be required in response to internal factors, such as the need to correct design flaws or coding errors, or external factors, such as changes in consumer requirements.

This task requires you to extend your solution to Part A of the assignment by adding an additional feature. It tests:

- Your ability to work under time pressure; and
- The quality and clarity of your code for Part A, because a well-written solution to Part A will make completing this part of the assignment much easier.

Note that Part B of the assignment involves modifying your complete solution to Part A. You cannot receive marks for Part B unless you have a working solution to Part A.

Goal

In Part A of this assignment you were required to create code that simulated different arrangements of children's building blocks. In real life, however, children are notoriously careless with their toys and are likely to have lost some of the blocks! Therefore, in Part B of this assignment you will simulate this real-life scenario.

To complete this task you must modify your solution to Part A so that it does *not* draw blocks we assume are lost. If you study the data sets in the provided Python file, `building_blocks.py`, you will see that some "arrangements" have an '`X`' associated with certain blocks. This value marks blocks that have been lost. Your task for Part B of the assignment is to modify your Part A solution so that it does *not* draw missing blocks.

However, what do we do if the block intended to be underneath another one is lost? Obviously blocks cannot float in mid air, so in this situation you must draw the uppermost block on the ground.

You will complete this part of the assignment by extending your code for Part A. No additional Python template file is supplied for this part. Your program must work for all of the data sets in the supplied Python file, and any other similar data sets in the same format.

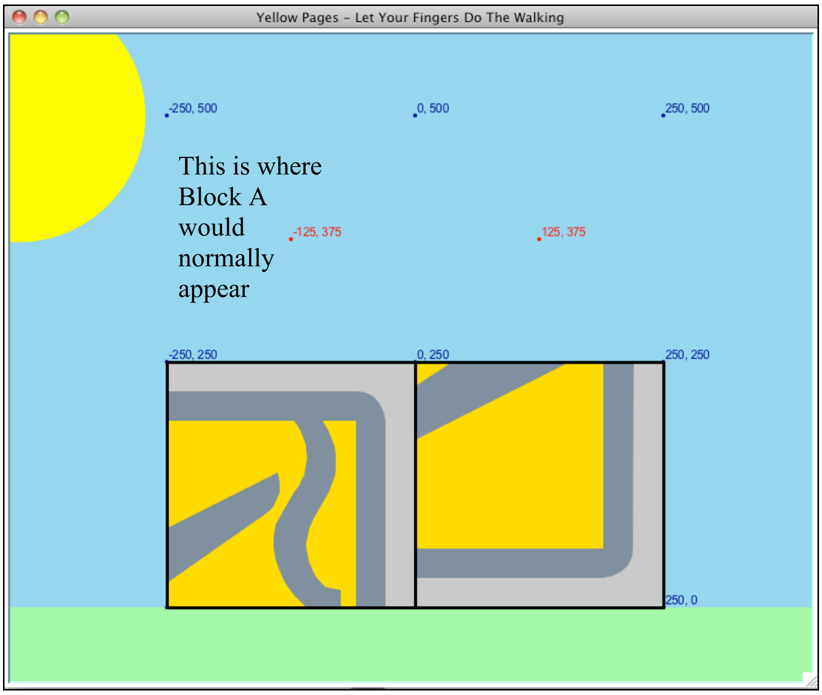
Illustrative example

To illustrate the requirements we'll continue our example from the Part A instructions. Recall that we created four blocks which, when arranged correctly, produced the Yellow Pages logo.

However, consider the following arrangement.

```
arrangement_51 = [['Block B', 'Bottom right', 'Right', 'O'],
                  ['Block D', 'Bottom left', 'Left', 'O'],
                  ['Block A', 'Top left', 'Right', 'X']]
```

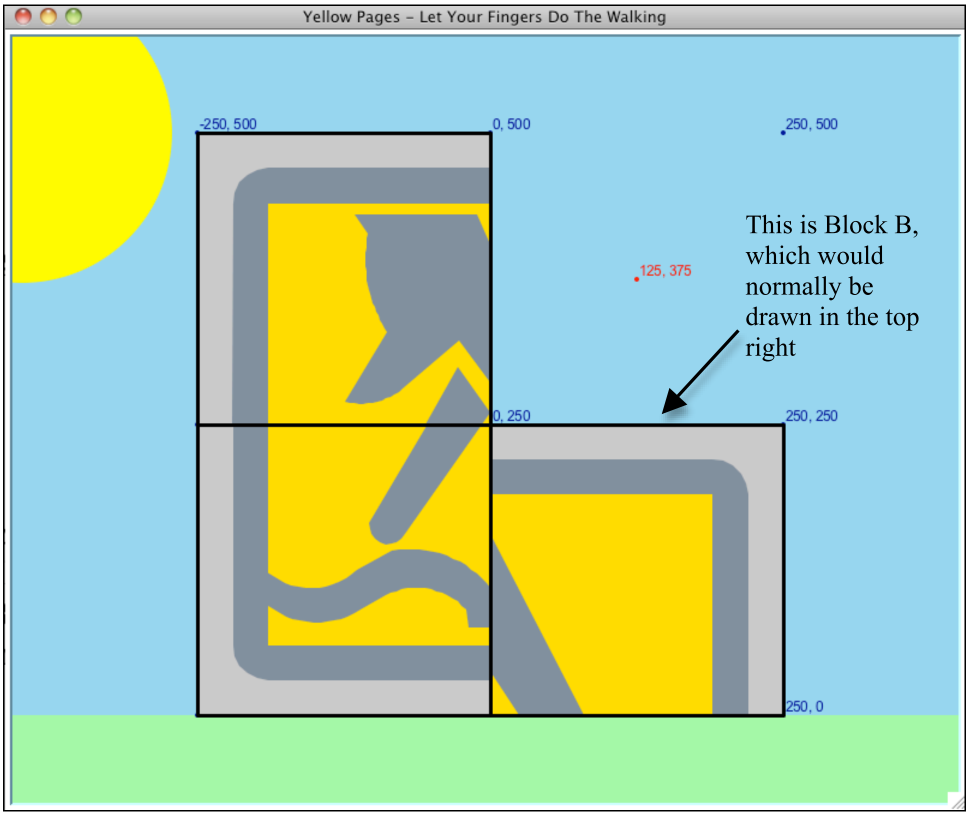
Normally this arrangement would oblige us to draw three blocks, but in this case the presence of an '`X`' indicates that Block A has been lost. Therefore we should draw only Blocks B and D as follows.



In this case the missing block was from the top row, so we merely needed to suppress drawing it. But now consider the following arrangement.

```
arrangement_92 = [['Block D', 'Bottom right', 'Up', 'X'],
                  ['Block B', 'Top right', 'Up', 'O'],
                  ['Block C', 'Bottom left', 'Up', 'O'],
                  ['Block A', 'Top left', 'Up', 'O']]
```

In this case we would normally draw four blocks, but Block D has been lost so should not be drawn. However, the lost block is from the bottom row in this case, which would leave Block B suspended in mid air. In this case, therefore, the unsupported block must be drawn at ground level, resulting in the following image.



This is Block B, which would normally be drawn in the top right

Requirements and marking guide

To complete this task you are required to extend your Part A `building_blocks.py` file by modifying the code that draws the blocks so that any block that has been lost, as indicated by an '`X`', is *not* drawn. Furthermore, blocks that would be unsupported because the block meant to be underneath them has been lost must be drawn on the ground, instead of floating in mid air.

Your submitted solution for both Parts A and B will consist of a *single Python file*. Your Part B extension must satisfy the following criteria. Marks available are as shown.

1. **Lost blocks are *not* drawn (2%)**. Of course, your solution must still draw all blocks which have not been lost. You cannot receive marks for this criterion if your code does not draw a block which isn't lost. A program that never draws *any* blocks will not be awarded any marks.
2. **Unsupported blocks are drawn on the ground (4%)**. If a missing block would normally support another in the arrangement, then the unsupported block must be drawn on the ground. Note that in all the arrangements the blocks on the bottom row appear in the arrangement before those on the bottom row.

You must complete this task using *basic Turtle graphics and maths functions only*. You may not import any additional modules or files into your program other than those already included in the original `building_blocks.py` template.

Development hints

- It should be possible to complete this task merely by *adding* code to your existing solution, with little or no change to the code you have already completed.
- To help decide where to draw "unsupported" blocks you can rely on the fact that the "arrangements" require drawing "supporting" blocks before "supported" ones.
- If you are unable to complete the whole task, just submit whatever part you can get working. You will receive *partial marks for incomplete solutions*.

Deliverable

You must develop your solution by completing and submitting the provided Python file `building_blocks.py` as follows. **Do not submit any other files! Do not submit a compressed archive ('zip' or 'rar') containing multiple files!**

1. Complete the "statement" at the beginning of the Python file to confirm that this is your own individual work by inserting your name and student number in the places indicated. *We will assume that submissions without a completed statement are not your own work and they will not be marked.*
2. Complete your solution by developing Python code to replace the dummy `stack_blocks` function. You must complete your solution using *only the modules already imported by the provided template*. You may *not* use or import any other modules to complete this program. In particular, you may *not* import any image files into your solution.
3. Submit *a single Python file* containing your solution for marking. Do *not* submit an archive (e.g., in 'zip' or 'rar' formats) containing several files. Only a single file will be accepted, so you cannot accompany your solution with other files or pre-defined images.

Apart from working correctly your program code must be well-presented and easy to understand, thanks to (sparse) commenting that explains the *purpose* of significant code segments and *helpful* choices of variable and function names. *Professional presentation* of your code will be taken into account when marking this assignment.

If you are unable to solve the whole problem, submit whatever parts you can get working. You will receive *partial marks for incomplete solutions*.

How to submit your solution

A link is available on Blackboard under *Assessment* for uploading your solution file before the deadline (11:59pm Monday, April 17th). You can submit as many drafts of your solution as you like. You are strongly encouraged to *submit draft solutions* before the deadline.