1. **Start up *idle3***

Open a terminal and enter the command ‘idle3’ and press return

1. **Import the turtle library**

Start a new file and enter the following line:

import turtle

Save the file as ‘inf1-wc1-[yourname].py’. This is your program for today. Every line that is not code (like answers to questions) should begin with a hash (#). Also, code that should not be active shall be started with a hash.

1. **Make a turtle**

Add the following line to your program:

don = turtle.Turtle()

Save the program and run it. From this point on, ***at the end of each step run the program.***

*Assignment: Write a comment in the program explaining what happens in that line of code. Write answers to questions as comments in the program.*

1. **Draw a line: forward and turn**

To make your turtle *don* go forward 100 pixels, call his *forward* function with the value 100. To make him turn left 72 degrees (one fifth of a circle), call his *left* function with the value 72:

don.forward(100)

don.left(72)

1. **Draw a pentagon**

Just draw a pentagon. You already started.

1. *How many lines of code do you need for drawing a pentagon?*
2. *How many lines of code do you need for drawing an octagon?*
3. *How many lines of code do you need to draw a circle?*
4. **Avoid repeating yourself (tell the computer to do it for you)**

A pentagon can be drawn efficiently using a for-loop. Replace the lines you have for drawing the pentagon with the following code (mind the indentation):

for step in range(5):

don.forward(100)

don.left(72)

1. *How many lines of code do you now need for an octagon?*
2. *How many lines do you now need for a circle?*
3. *What does range(5) do?*
4. *How does the for-loop work?*
5. **Draw a flower of hexagons**

For each shape in a range of 90, draw a hexagon and turn left 4 degrees.

1. *How many lines of code do you need?*
2. *How many levels of indentation do you need?*
3. *How does the double for-loop work?*
4. **Make a function for drawing a polygon**

Make the double for-loop inactive by adding hashes (#) in front of each line.

Now write the following code to do the same flower using a function:

def polygon(sides):

for step in range(sides):

don.forward(100)

don.left(360 / sides)

for shape in range(90):

polygon(6)

don.left(4)

A function ***defined***, using the keyword *def*. You give it a name and indicate the parameters as names within the parentheses. Here, the only parameter (for now) is *sides*. The lines following the line starting with *def* form the *body* of the function: the lines of code that are executed if the function is called.

1. *How does this code work to give the same result as before?*
2. *What happens when the function is called as* polygon(6)*?*
3. *Why/when is a function useful?*
4. *What kind of data is* sides *and why (whole number, decimal number, text, or something else)?*
5. *How can you draw a flower of large squares and a flower of small hexagons?*
6. **Improve the function**

The function is now restricted to a size of 100 *and* it is restricted to a turtle named *don*. Change the function to:

def polygon(turt, sides, size=100):

for step in range(sides):

turt.forward(size)

turt.left(360 / sides)

Now update the call to the polygon function in the flower-drawing for-loop, so that it works again.

NOTE: the parameter names can be chosen at will, but you should avoid confusion. The name ‘turtle’ refers to the library, so here the shorthand ‘turt’ is used instead.

1. *What happens if you call the function with a turtle (e.g. don), the number 10, and the number 50, as: polygon(don, 10, 50)?*
2. *What happens if you call the function with a turtle, and the number 13, but without a value for the size, as: polygon(don, 13)?*
3. *What happens if you call the function with only a turtle, as: polygon(don)?*
4. *What happens if you call the function with a turtle, the number 1, the number 2 and the number 3, as: polygon(don, 1, 2, 3)?*
5. *What happens if you call the function with the text ‘don’, instead of the turtle don, as: polygon(‘don’, 5, 50)?*
6. **Try this**

for size in range(90):

polygon(don, 7, size)

don.left(4)

1. *What does this do?*
2. *How does this work?*
3. **Try this**

for sides in range(3, 9):

polygon(don, sides)

don.left(60)

1. *What does this do?*
2. *How does this work?*
3. *What does* range(3, 9) *do?*
4. **Try this**

for sides in range(3, 12, 2):

polygon(don, sides)

don.left(72)

1. *What does this do?*
2. *What does* range(3, 12, 2) *do? (You can type* help(range) *in the IDLE console to read more)*
3. **Alternating colors**

for step in range(90):

if (step % 2) == 1:

don.color(“red”)

else:

don.color(“blue”)

polygon(don, 5)

don.left(360 / 90)

1. *What does this do?*
2. *How does this work?*
3. *What sort of value is “red” (see question 7d)?*
4. **More alternating colors**

for step in range(90):

if (step % 3) == 1:

don.color(“red”)

elif (step % 3) == 2:

don.color(“yellow”)

else:

don.color(“blue”)

polygon(don, 5)

don.turn(360 / 90)

1. *What does the %* operator *do?*
2. *How does the if/elif/else work?*
3. **More alternating colors**

The colors are set by *name*, given as a piece of text. This is different from names of values and functions, which are stored in memory (like *turtle*, *don*, and *polygon*). The latter are written without quotes, while pieces of text are written with quotes.

Now we’ll make a flower of shapes with many alternating colours in a rainbow scheme. For that we first define the *list of color names*:

colors = [“red”, “orange”, “yellow”, “green”, “cyan”, “blue”, “magenta”]

for step in range(70):

index = step % len(colors)

color = colors[ index ]

don.color(color)

polygon(don, 6)

don.turn(360 / 70)

1. *What does* len(colors) *do?*
2. *What does* step % len(colors) *do (think what happens if* step *has the value of 0, 1, 2, …, 7, 8, …, etc)?*
3. *What does* colors[ index ] *do (think what happens if* index *has the value if 0, 1, … 6)?*
4. *What would happen if index would have a value of 7, or higher?*
5. *What would happen if index would have a value of -1, or lower?*
6. **A better rainbow**

We now have alternating colors, but it would be a better rainbow if the first 1/7th part of shapes were one color, then the second part had the second color, etc. That can be done with a subtle change to the code:

colors = [“red”, “orange”, “yellow”, “green”, “cyan”, “blue”, “magenta”]

for step in range(70):

index = step // len(colors)

color = colors[ index ]

don.color(color)

polygon(don, 6)

don.turn(360 / 70)

1. *What does the //* operator *do?*
2. *How is the //* operator *different from the /* operator *(you can try this in the IDLE console: 4 / 3 versus 4 // 3, 9 / 4 versus 9 // 4, etc)?*
3. *Why is it kind of important to use 70 or another*