



Computer Science – Year 9

KNOWLEDGE ORGANISER

To support your revision for the End of Year Assessment

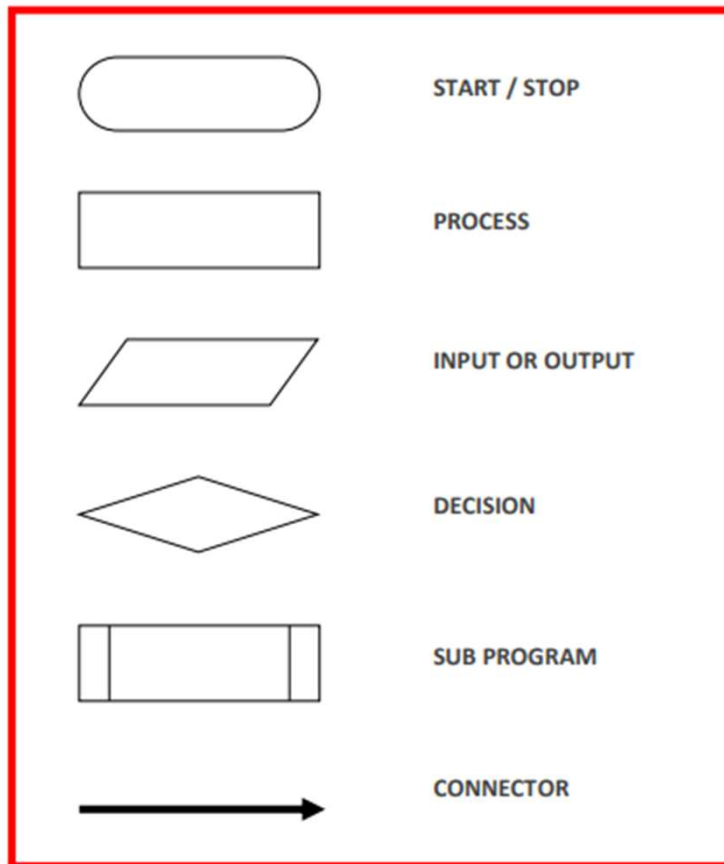
<http://teach-ict.com/>

Kindness Curiosity Creativity Community Respect Perseverance

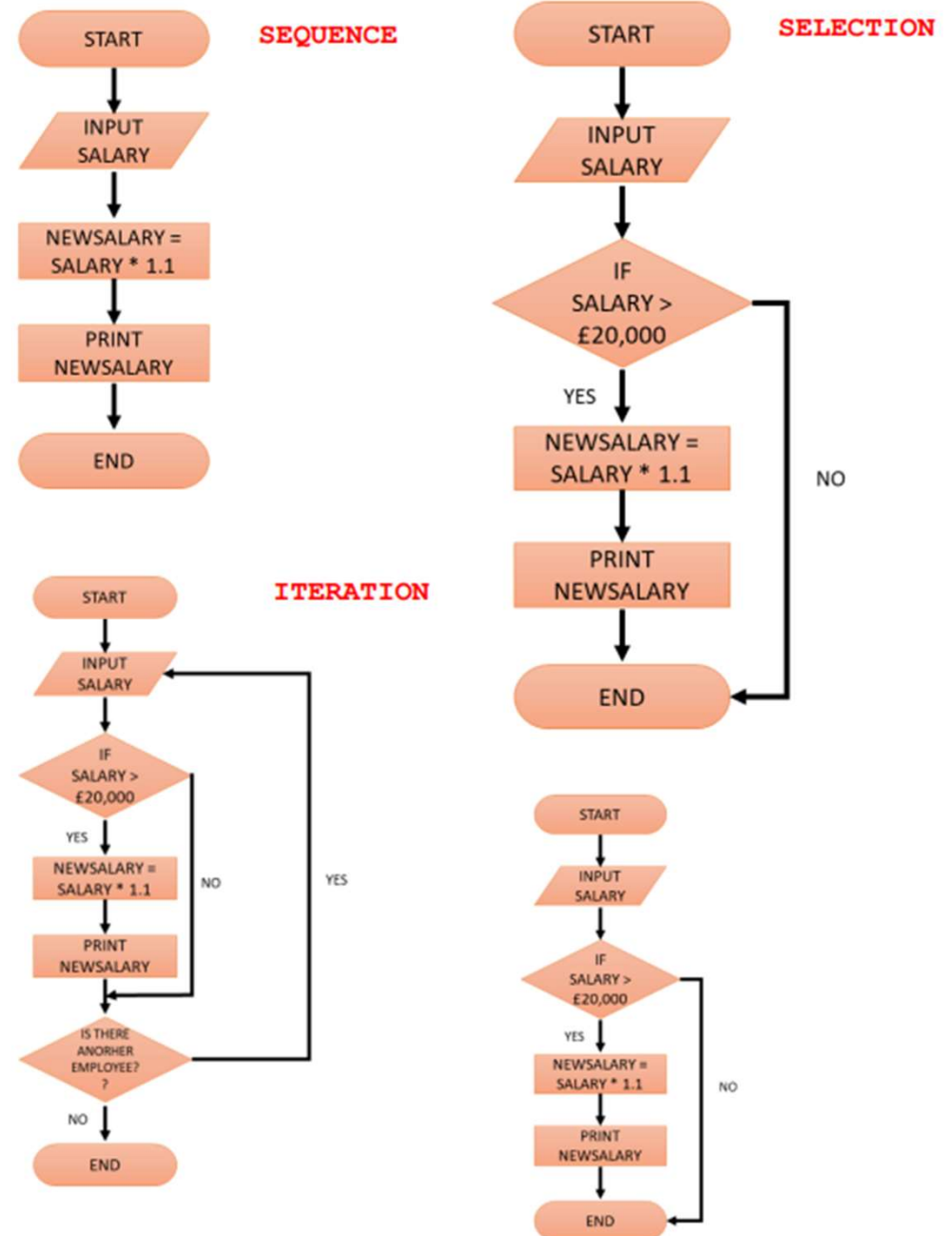


Unit 1 :: Flowcharts

Flow chart is a method of representing the sequences of steps in an algorithm in the form of a diagram.



Examples



Unit 2 :: Boolean logic and truth tables

Simple logic diagrams using the operators “AND”, “OR” AND “NOT”

Truth tables

Combining Boolean operators using “AND”, “OR” and “NOT”

Applying logical operators in truth tables to solve problems

There are a number of different logic gates which produce different results when they receive inputs (1's and 0's.)

Computers are made up of circuits containing millions of switches. As electrical switches have two possible values (ON or OFF), these values can be represented using binary values 1 or 0. Each circuit contains logic gates and **BOOLEAN LOGIC** is used to evaluate the results of different combinations of 1's and 0's.



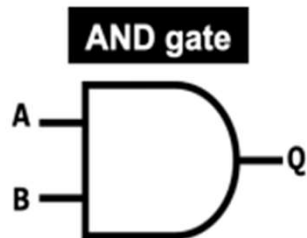
REVISION NOTE

You need to be able draw a truth table for a given circuit. You also need to be able to represent a circuit as a Boolean expression

The possible values for each gate can be represented using a **TRUTH TABLE**.

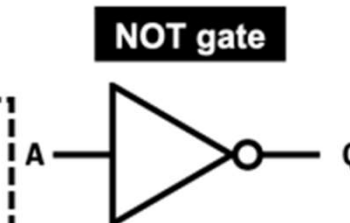
An **AND** gate has two possible inputs - ‘A’ and ‘B’

‘Q’ are the possible outputs



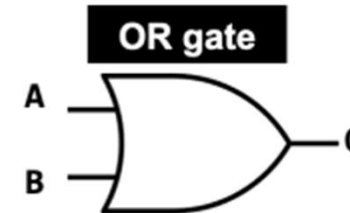
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

A **NOT** gate has a single input – ‘A’



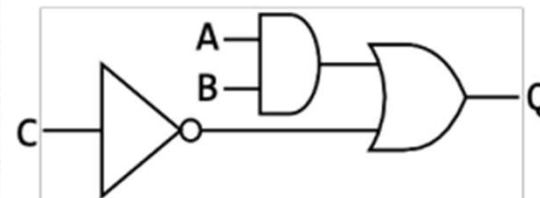
A	Q
0	1
1	0

An **OR** gate has two possible inputs – ‘A’ and ‘B’



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

Logic gates can be combined to create complete circuits. These can also be represented using truth tables. The circuit below is made up of three gates:



This can also be represented as a Boolean expression:

(A AND B) OR (NOT C)

A	B	C	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Unit 3 :: Sorting algorithms

Standard searching algorithms:

- ☐ Binary search
- ☐ Linear search

A **BINARY SEARCH** requires data to be sorted in order before it can be searched. A **LINEAR SEARCH** does not—the algorithm will look at every item in list until it either locates the data or reaches the end of the list. The binary search is the more efficient of the two

```
INPUT item to be searched for
found = False
numbers = [4,2,6,1,5,3]
REPEAT
  Compare item with current item in list
  IF current item is the item searched for then
    found = True
  UNTIL end of list OR found = True
IF found = True
  PRINT ("Item found")
ELSE
  PRINT ("Item not found")
```

LINEAR SEARCH

We are searching for 6 in a sorted list: 1 2 3 4 5 6 7

List is split in two at the mid point: 1 2 3 4 5 6 7 $6 > 4$ so discard items less than 4

List is split in two at the mid point: 4 5 6 7 $6 > 5$ so discard items less than 5

List is split in two at the mid point: 6 7 Item has been found

BINARY SEARCH

Standard sorting algorithms:

- ☐ Bubble sort
- ☐ Merge sort
- ☐ Insertion sort

REVISION NOTE

You need to be familiar with searching sorting algorithms but there is no need for you to be able to code them

-A **BUBBLE SORT** is an algorithm for sorting data.
-The algorithm works by going through a list of unordered data and evaluating the data in pairs.
-If two data items are in the wrong order they are exchanged.
-The algorithm then moves to the next pair.
-When the algorithm reaches the end of the data, the process will be repeated until all data has been sorted correctly. This might take **SEVERAL PASSES** through the data.

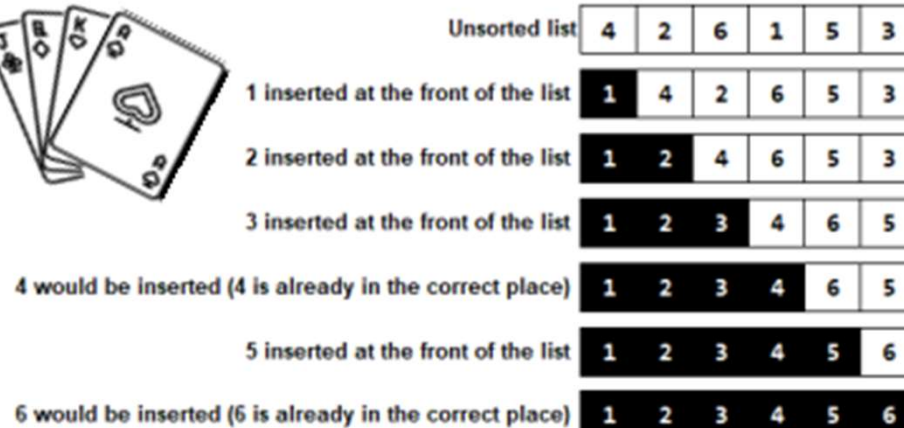
STARTING DATA	4	2	6	1	5	3	
Items 1 & 2	2	4	6	1	5	3	$2 > 4$ so SWAP
Items 2 & 3	2	4	6	1	5	3	$4 < 6$ NO SWAP
Items 3 & 4	2	4	1	6	5	3	$1 < 6$ so SWAP
Items 4 & 5	2	4	1	5	6	3	$5 < 6$ so SWAP
Items 2 & 3	2	4	1	5	3	6	$3 < 6$ so SWAP

Unit 3 :: Sorting algorithms

- A **MERGE SORT** is a **DIVIDE AND CONQUER** algorithm;
- First of all, the items of data in a list are divided in half until each item is in a **SUBLIST** of one item.(This is the **DIVIDE** stage)
- The algorithm will then merge each sublist, after comparing and sorting them as appropriate.
- When all of the data has been merged back into a single list it will be in the correct order. (This is the **CONQUER** stage)
- Merge sorts are more efficient than bubble or insertion sorts.



- An **INSERTION SORT** is more efficient than a bubble sort.
- The insertion sort works in a similar way to sorting a hand of cards.
- The algorithm works by comparing the current data item with the other items in the list
- If the data item is in the wrong place, it is shifted to left until it is in the correct place.
- This continues until all the items of data are in the correct place.



Unit 4 :: Turtle

Turtle motion

```
turtle.forward(distance)
```

```
turtle.fd(distance)
```

Parameters: distance – a number (integer or float)

Move the turtle forward by the specified distance, in the direction the turtle is headed.

```
turtle.right(angle)
```

```
turtle.rt(angle)
```

Parameters: angle – a number (integer or float)

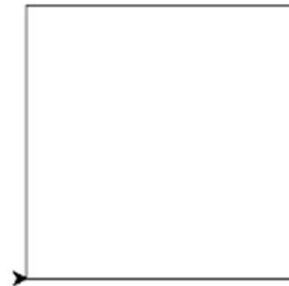
Turn turtle right by angle units. (Units are by default degrees).

```
turtle.left(angle)
```

```
turtle.lt(angle)
```

Parameters: angle – a number (integer or float)

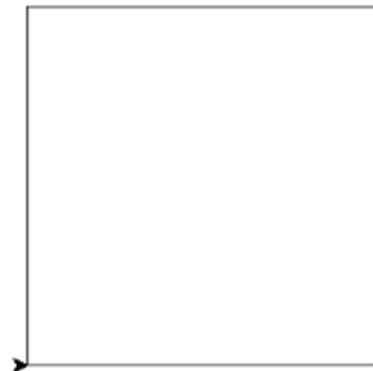
Turn turtle left by angle units. (Units are by default degrees, but can be set via the `degrees()` and `radians()` functions.)



```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
```

```
>>> import turtle
>>> win=turtle.Screen()
>>> turtle.forward(200)
>>> turtle.left(90)
>>> turtle.forward(200)
>>> turtle.left(90)
>>> turtle.forward(200)
>>> turtle.left(90)
>>> turtle.forward(200)
>>> turtle.left(90)
>>>
```

Doing angles a different way



```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
```

```
>>> import turtle
>>> wn=turtle.Screen()
>>> dan=turtle.Turtle()
>>> for i in range(4):
>>>     turtle.forward(250)
>>>     turtle.left(360/4)
```

```
>>>
```

Unit 4 :: Turtle

```
turtle.undo()
```

Undo (repeatedly) the last turtle action(s).

```
turtle.reset()
```

Delete the turtle's drawings from the screen, re-center the turtle and set variables to the default values.

```
turtle.back(distance)
```

```
turtle.bk(distance)
```

```
turtle.backward(distance)
```

Parameters: distance – a number

Move the turtle backward by distance, opposite to the direction the turtle is headed.

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 2
2:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more in
formation.
```

```
>>> import turtle
>>> wn=turtle.Screen()
>>> turtle.forward(200)
>>> turtle.left(90)
>>> turtle.forward(300)
>>> turtle.undo()
>>> turtle.reset() I
>>> |
```

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 2
2:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more in
formation.
```

```
>>> import turtle
>>> wn=turtle.Screen()
>>> turtle.forward(200)
>>> turtle.left(90)
>>> turtle.forward(300)
>>> turtle.undo()
>>> turtle.reset()
>>> turtle.backward(100)
>>> turtle.bk(100) I
>>> turtle.reset()
>>> |
```

Unit 4 :: Turtle

Shape / size of turtle

`turtle.shape(name)`

Parameters : name – a string which is a valid shape name

Set turtle shape to shape with given name or, if name is not given, return name of current shape. Shape with name must exist in the TurtleScreen's shape dictionary. Initially there are the following polygon shapes: "arrow", "turtle", "circle", "square", "triangle", "classic".



`turtle.turtlesize(size)`



Naming turtle

```
>>> dan=turtle.Turtle()
>>> dan.forward(200)
>>> |
```

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 2
2:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more in
formation.
```

```
>>> import turtle
>>> wn=turtle.Screen()
>>> turtle.forward(200)
>>> turtle.left(90)
>>> turtle.forward(300)
>>> turtle.undo()
>>> turtle.reset()
>>> turtle.backward(100)
>>> turtle.bk(100)
>>> turtle.reset()
>>> turtle.shape("turtle")
>>>
```

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 2
2:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more in
formation.
```

```
>>> import turtle
>>> wn=turtle.Screen()
>>> turtle.forward(200)
>>> turtle.left(90)
>>> turtle.forward(300)
>>> turtle.undo()
>>> turtle.reset()
>>> turtle.backward(100)
>>> turtle.bk(100)
>>> turtle.reset()
>>> turtle.shape("turtle")
>>> turtle.turtlesize(2)
>>> turtle.turtlesize(3)
>>> |
```


Unit 4 :: Turtle

Drawing circle

`turtle.circle(radius)`
Parameters: radius – a number
Draw a circle with given radius

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more
information.
>>> import turtle
>>> wn=turtle.Screen()
>>> turtle.shape("turtle")
>>> dan=turtle.Turtle()
>>> dan.circle(50)
>>> dan.reset()
>>>
```

Change pen colour

`turtle.pencolor(colorstring)`
Set pencolor to colorstring, which is a Tk color specification string, such as "red", "yellow", or "#33cc8c".

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more
information.
>>> import turtle
>>> wn=turtle.Screen()
>>> dan=turtle.Turtle()
>>> wn.bgcolor("lightgray")
>>> dan.color("red")
>>> dan.forward(200)
>>> |
```

Change the background colour

`turtle.bgcolor(*args)`
Parameters : args – a color string or three numbers in the range 0..colormode or a 3-tuple of such numbers

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more
information.
>>> import turtle
>>> wn=turtle.Screen()
>>> turtle.shape("turtle")
>>> dan=turtle.Turtle()
>>> wn.bgcolor("lightblue")
>>> wn.bgcolor("lightgreen")
>>> |
```

Unit 4 :: Turtle

Pen up / Pen down, Pen size

```
turtle.pendown()  
turtle.pd()  
turtle.down()
```

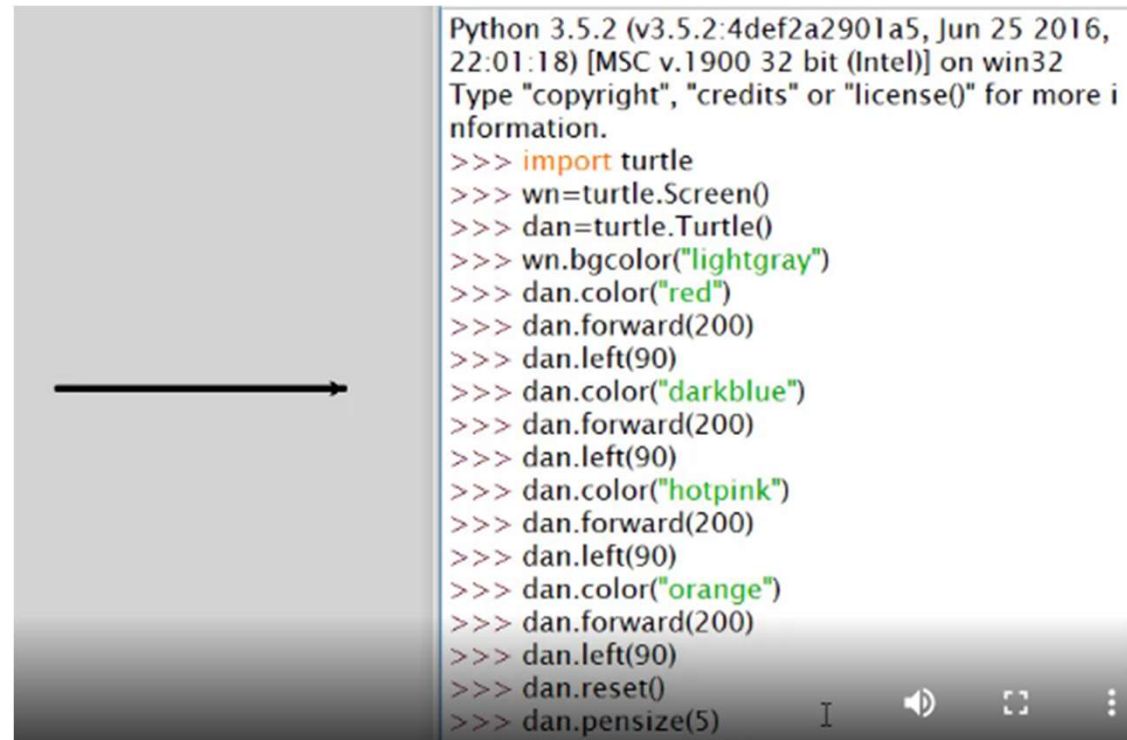
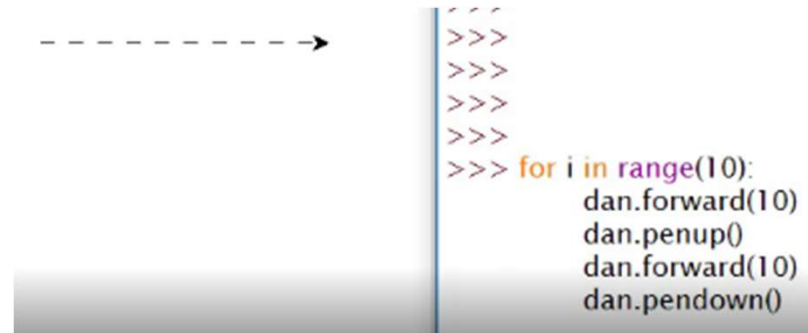
Pull the pen down – drawing when moving.

```
turtle.penup()  
turtle.pu()  
turtle.up()
```

Pull the pen up – no drawing when moving.

```
turtle.pensize(width)
```

Parameters : width – a positive number



Unit 4 :: Turtle

Using variables for side lengths and angles

Using a FOR Loop

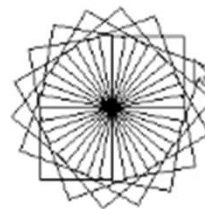


```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import turtle
>>> wn=turtle.Screen()
>>> dan=turtle.Turtle()
>>> x=200
>>> y=100
>>> z=90
>>> for i in range(2):
    dan.forward(x)
    dan.left(z)
    dan.forward(y)
    dan.left(z)
```

Using Functions

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import turtle
>>> wn=turtle.Screen()
>>> dan=turtle.Turtle()
>>> def square():
    for i in range(4):
        dan.forward(100)
        dan.left(90)
>>> square()
>>> dan.reset()
>>> square()
>>> |
```

Tilting shapes



```
>>> dan.left(20)
>>> for i in range(4):
    dan.forward(50)
    dan.left(90)
```

```
>>> dan.left(20)
>>> for i in range(4):
    dan.forward(50)
    dan.left(90)
```

```
>>> dan.left(20)
>>> for i in range(4):
    dan.forward(50)
    dan.left(90)
```

Unit 4 :: Turtle

```
turtle.write(arg, font=('Arial', 8, 'normal'))
```

Parameters : arg – object to be written to the TurtleScreen

font – a triple (fontname, fontsize, fonttype)

Write text - the string representation of arg - at the current turtle position according to align ("left", "center" or "right") and with the given font. If move is true, the pen is moved to the bottom-right corner of the text. By default, move is False.

Hello my name is Dan

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import turtle
>>> wn=turtle.Screen()
>>> dan=turtle.Turtle()
>>> dan.write('Hello my name is Dan',font=('Tahoma',24,'bold'))
>>> 
>>>
```

```
turtle.goto(x, y)
```

Move turtle to an absolute position. If the pen is down, draw line. Do not change the turtle's orientation.

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import turtle
>>> wn=turtle.Screen()
>>> dan=turtle.Turtle()
>>> dan.penup()
>>> dan.goto(100,200)
>>> dan.reset()
>>> dan.goto(-100,-200)
>>> |
```

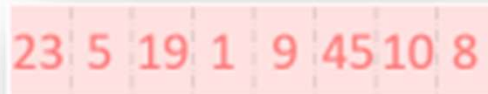


Unit 4 :: Lists

Understanding index positions

Python Lists

A list is a *collection of data that only needs to be assigned to just one variable*. Look at the example below. What would you have to do without a list to store each of these numbers? You would need 8 different variables! Each 'slot' in the list has an index so that we can find an item in the list. These always start at zero. Have a look below:



23 5 19 1 9 45 10 8

```
my_list = [23, 5, 19, 1, 9, 45, 10, 8]
```

Python lists have just one variable name. The list has square brackets at either end, and items of data are separated by commas.

Lists are sometimes called 'arrays'



23 5 19 1 9 45 10 8
0 1 2 3 4 5 6 7

my_list[0] contains 23

my_list[1] contains 5

my_list[2] contains 19

my_list[3] contains 1

my_list[4] contains 9

my_list[5] contains 45

my_list[6] contains 10

my_list[7] contains 8

Unit 4 :: Lists

```
favFoods=['banana','ice cream','cheese','chocolate','pizza']
print(favFoods)
```

Adding an item to a list

```
favFoods=['banana','ice cream','cheese','chocolate','pizza']
print(favFoods)
favFoods.append('nachos')
print(favFoods)
```

Insert an item at an index position

```
favFoods=['banana','ice cream','cheese','chocolate','pizza','nachos']
favFoods.insert(1,'jam')
print(favFoods)
```

```
['banana', 'jam', 'ice cream', 'cheese', 'chocolate', 'pizza', 'nachos']
```

Remove an item from a list

```
favFoods=['banana','jam','apple','ice cream','cheese','chocolate']
favFoods.remove('cheese')
print(favFoods)
```

Removing an item using its index position

```
favFoods=['banana','jam','apple','ice cream','cheese','chocolate']
favFoods.remove('cheese')
#print(favFoods)
del favFoods[2]
print(favFoods)
```

Modify an item in a list

```
colours=['red','yellow','green','orange']
colours[3]='orange'
print(colours)
```

Unit 4 :: Lists

Adding brown to your list

```
rs=['red','yellow','green','orange']
rs[0]='purple'
print(colours)
rs[2]='blue'
print(colours)
rs.insert(2,'brown')
print(colours)
```

Finding the number of items in a list

```
dogs=['poodle','doberman','greyhound','collie','spaniel']
print(len(dogs))
```

Search a list

•Python is **case sensitive** and so
'nicholas' is not the same as
'Nicholas'

```
team=['Jacob','Aiden','Ethan','Nicholas','Daniel']
if 'Gareth' in team:
    print('Gareth is a member of the team')
else:
    print('Gareth is not a member of the team')
```

Using numerical data in lists

```
scores=[25, 19, 36, 12, 34, 28]
if 76 in scores:
    print('76 is a valid score')
else:
    print('76 was not one of the scores achieved')
```

Using the random module with lists

```
import random
names=['Bob','Fred','James','John','Sam','Ahmed']
pickNames=random.choice(names)
print(pickNames)
```

Concatenate text with a random item

```
import random
pets=['dog', 'cat', 'goldfish', 'hamster', 'tortoise', 'snake', 'rabbit']
pickPet=random.choice(pets)
print('I believe that the most loyal pet is a ' + pickPet) I
```

Unit 4 :: Lists

Using the 'range' function to Iterate

```
forenameList = ["Gomong", "Momik", "Throbag", "Rurib", "Shatog", "Kobik"]
surnameList = ["Bloodfang", "Irontooth", "Bonespear", "Skulley", "Redclub", "Ribaxe"]
for item in range(0,6):
    print(forenameList[item] + " " + surnameList[item])
```

Using the term 'i' with the range function

```
forenameList = ["Gomong", "Momik", "Throbag", "Rurib", "Shatog", "Kobik"]
surnameList = ["Bloodfang", "Irontooth", "Bonespear", "Skulley", "Redclub", "Ribaxe"]
for i in range(1,6):
    print(forenameList[i] + " " + surnameList[i])
```

Using the random.choice function

```
import random
forenameList = ["Gomong", "Momik", "Throbag", "Rurib", "Shatog", "Kaog", "Kobik"]
surnameList = ["Bloodfang", "Irontooth", "Bonespear", "Skulley", "Redclub", "Ribaxe"]
finalName = random.choice(forenameList) + " " + random.choice(surnameList)
print(finalName)
```

Using four lists to generate a random name

```
import random

forenameList1 = ["Go", "Mo", "Thro", "Ru", "Sha", "Ka", "Ko", "Ki", "Bar", "Sku", "Ha", "Por", "Ig", "Ung", "Ool"]
forenameList2 = ["mong", "mik", "bag", "rib", "tog", "og", "bik", "bash", "tor", "kun", "zar", "fash", "gark", "narl"]

surnameList1 = ["Blood", "Iron", "Bone", "Skull", "Fish", "Red", "Rib", "Fire", "Steel", "Bronze", "Black", "Pig"]
surnameList2 = ["fang", "tooth", "-eye", "spear", "club", "killer", "smasher", "eater", "hand", "tongue", "claw"]

forenamePart1 = random.choice(forenameList1)
forenamePart2 = random.choice(forenameList2)

surnamePart1 = random.choice(surnameList1)
surnamePart2 = random.choice(surnameList2)

finalName = forenamePart1 + forenamePart2 + " " + surnamePart1 + surnamePart2

print(finalName)
```