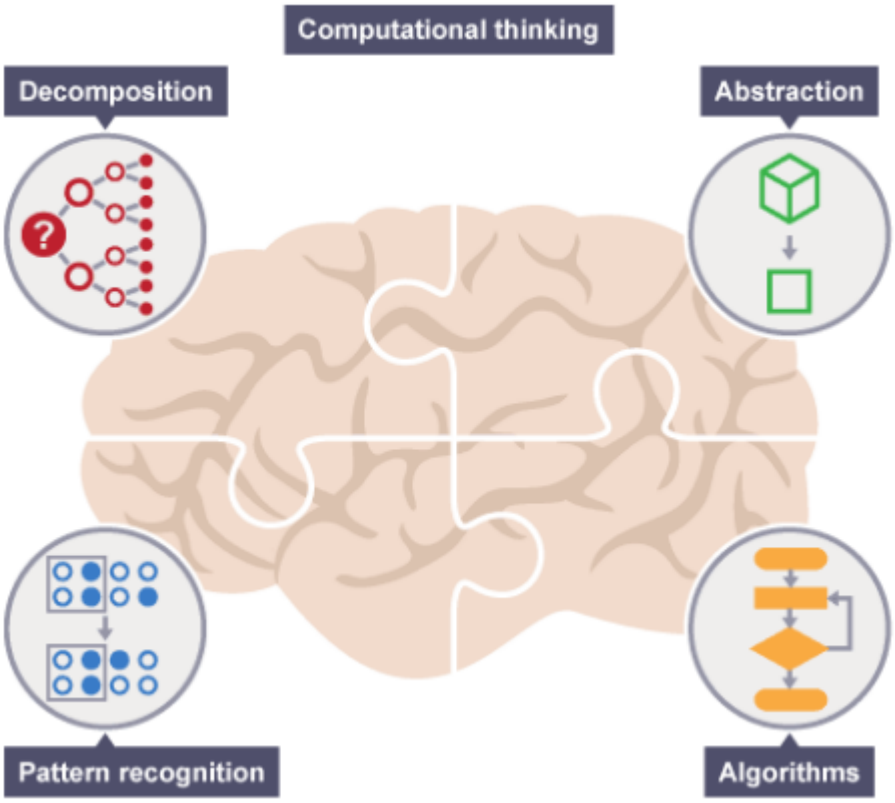
Topic 3 - Pseudocode, Flowcharting, and Computational Thinking

Computational Thinking

This involves taking complex problems, determining what the problem(s) are, and develop possible solutions



There are <u>four</u> parts to CT:

1. Decomposition

Breaking down a complex problem or system into smaller, more manageable parts

2. Pattern Recognition

Looking for similarities among and within problems

3. Abstraction

Focusing on the important information only, ignoring irrelevant details

4. Algorithms

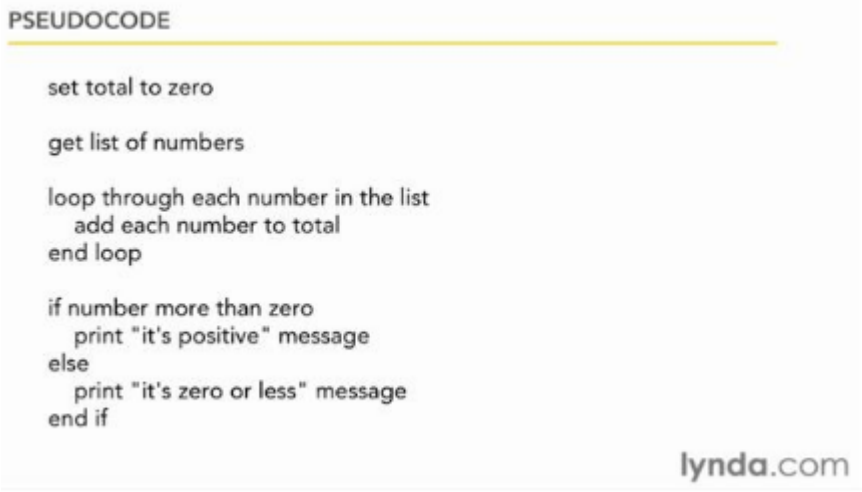
1 --> 2 --> 3

Developing a step-by-step solution to the problem, or the rules to follow to solve the problem

Pseudocoding -- What is it?

First off, it is simplified programming language

It is used for program design



Why use it?

It helps you remember / plan your program out

Easier to remember your intended outcome

Keeps you focused on your task

Exercise 1 - Interpret a Program into a Pseudocode

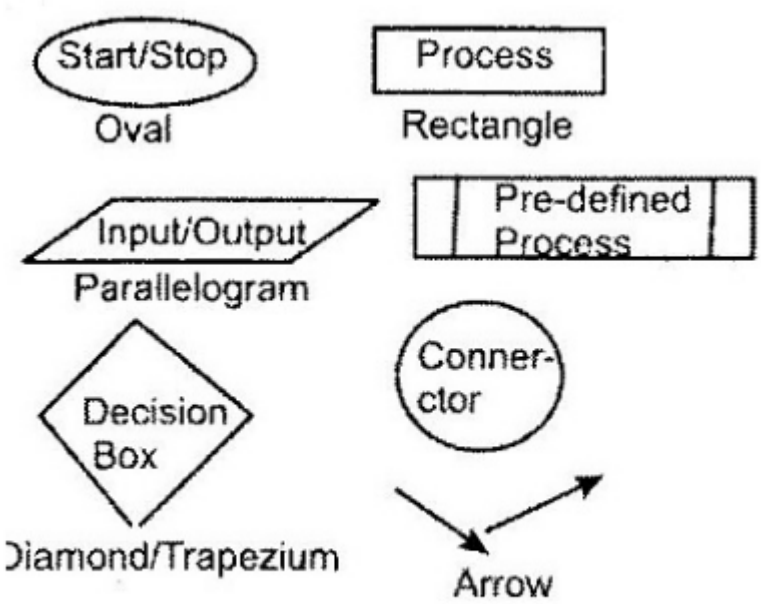
In [ ]:
```
while True:
    print("1. Addition")
    print("2. Subtraction")
    print("3. Multiplication")
    print("4. Division")
    print("5. Exit")

    choice = int(input("Enter your choice: "))
    if (choice >= 1 and choice <= 4):
        print("Enter two numbers: ")
        num1 = int(input())
        num2 = int(input())

        if choice == 1:
            res = num1 + num2
            print("Result = ", res)
        elif choice == 2:
            res = num1 - num2
            print("Result = ", res)
        elif choice == 3:
            res = num1 * num2
            print("Result = ", res)
        else:
            res = num1 / num2
            print("Result = ", res)
    elif choice == 5:
        break
    else:
        print("Wrong choice...")
```

Flowcharting

Flowcharting is a tool used to create a map of a computer program

It helps you debug your computer programs

- Where are you in the program where something is going wrong?
- What is keeping it from moving down the chart



Program:Flowchart

Exercise 2 - Write a Flowchart for this Program

In [ ]:
```
while True:
    print("1. Addition")
    print("2. Subtraction")
    print("3. Multiplication")
    print("4. Division")
    print("5. Exit")

    choice = int(input("Enter your choice: "))
    if (choice >= 1 and choice <= 4):
        print("Enter two numbers: ")
        num1 = int(input())
        num2 = int(input())

        if choice == 1:
            res = num1 + num2
            print("Result = ", res)
        elif choice == 2:
            res = num1 - num2
            print("Result = ", res)
        elif choice == 3:
            res = num1 * num2
            print("Result = ", res)
        else:
            res = num1 / num2
            print("Result = ", res)
    elif choice == 5:
        break
    else:
        print("Wrong choice...")
```

Topic 4 - Scope of Variables

A variable can have something called a "scope". This refers to its accessibility.

Scope of Variable *"Mr. Clarke"* -- Leduc Composite High School

Scope of Variable *"Mr. Wain"* -- Lab 209

Scope of Variable *"Justin Trudeau"* -- House of Commons!

A variable can be declared in one of two ways:

Look at the following program

In [2]:
```
name = "Luke Skywalker"
print(name)

name = "Jar Jar Binks"
print(name)
```

```
Luke Skywalker
Jar Jar Binks
```

So far, we have been working with variables in this manner.

Note, I can call *name* from **ANYWHERE** in the program

These declarations have usually been done at the start of our programs, and have been declared **OUTSIDE** of any **functions**

These are referred tot as **GLOBAL** variables

In [7]:
```
def Greeting():
    name = "Luke Skywalker"
    age = 60
    address = "Far far away"
    print(name)
    print(age)
    print(address)

Greeting()
```

```
Luke Skywalker
60
Far far away
```

Notice that these variables have been **declared inside** a function.

This means that the variables are only accessible from within the function

Hence, we have declared **LOCAL** variables here

If I typed **print(name)** outside of this function, it would result in a **compile error**!

So far we have learned some important facts about GOOD programming:

1. Use meaningful identifiers for variable names
   - Name not x
2. Use commenting to aid understanding
   - #this function checks to see if a letter is in the phrase

Best programming practice avoids use of global variables and favours local variables

1. Avoid the use of global variables in favour of **local variables**

Global and Local Variables within a Function

In this example, although both variables are called fun, one is **inside the function** and is a local variable and the other is **outside the function** and is a global variable.

They are completely independent of each other.

Example 1: What is our expected result?

In [ ]:
```
fun = 6 # global variable

def multi():
    fun = 6 # local variable
    fun = fun * 3
    print("I am a local variable called fun inside multi(). I equal: ", fun) #
print local variable

multi()
print("I am a global variable called fun outside multi(). I equal: ", fun) # p
rint global variable
```

Example 2: What will be printed out here?

In [ ]:
```
magic_number = 6  # Global Variable

def multi():
    fixed_number = 6 # Local Variable
    return fixed_number * magic_number

print(multi())
```

Example 3: What will happen here?

In [ ]:
```
magic_number = 6 # Global Variable

def multi():
    fixed_number = 6 # Local Variable

print(magic_number * fixed_number)
```

Look at the following code

This simply asks the user for a number and then calls a function to increment each time the CheckGuess function is called

Type this into Python and see what happens!

In [ ]:
```
guesses = 0

def CheckGuess (aGuess):
    guesses = guesses + 1 # can also type guesses += 1

aGuess = int(input("Please attempt a guess: "))
CheckGuess(aGuess)
print(guesses)
```

In [ ]:
```
guesses = 0

def CheckGuess (aGuess):
    global guesses
    guesses = guesses + 1 # can also type guesses += 1

aGuess = int(input("Please attempt a guess: "))
CheckGuess(aGuess)
print(guesses)
```

TASK #2 - Speed Task

Part A Write a program that will work out the distance travelled if the user enters in the speed and time

Part B Get the program to tell you the speed you would have to travel at in order to go a distance within a certain time entered by the user.

You are expected to use FUNCTIONS in your solution

Topic 5 - Control Structures

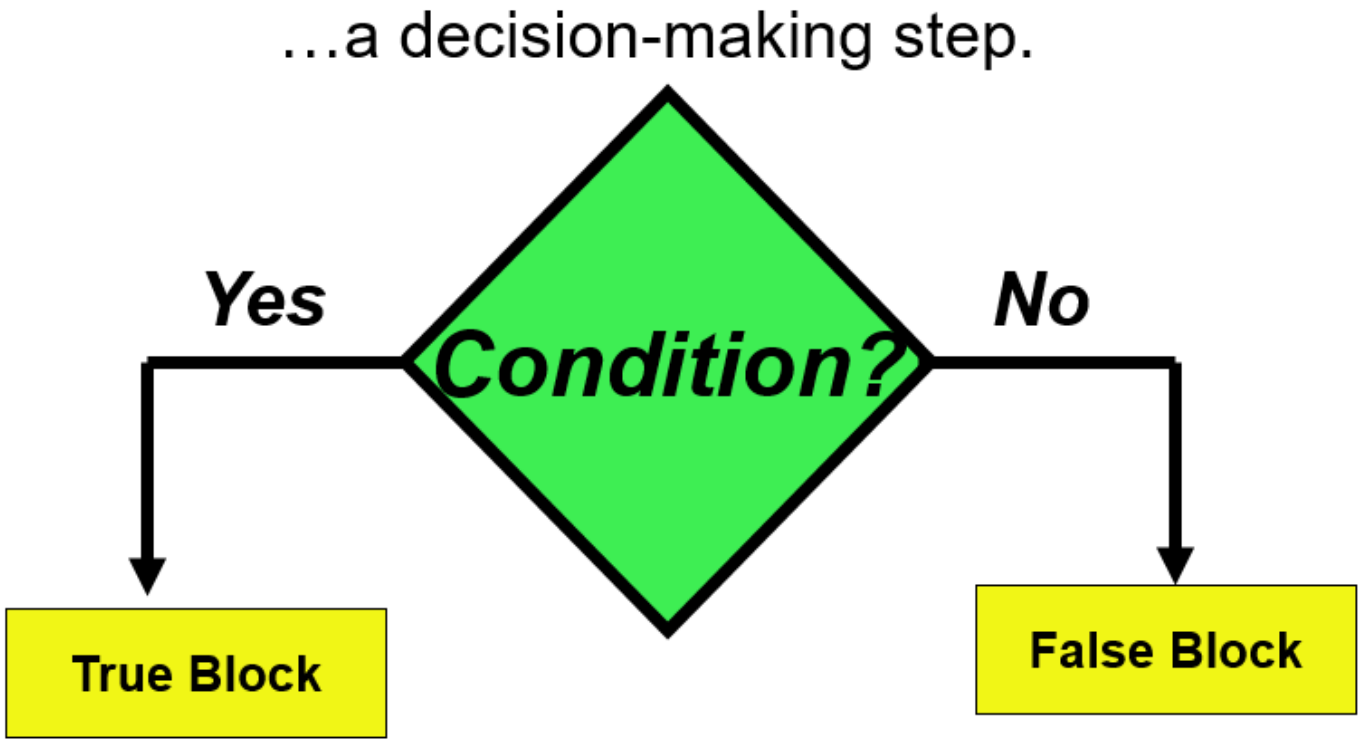
If-Else Statement

Programs run in consecutive steps (or groups of steps) processed one after another in the order that they arise.

Sometimes we want to test if one condition is true or false.

For example:

```
if Sarah >= 14:
    then she can learn to drive
else:
    she is too young to drive
```



In [4]:
```
name = "Sydney"

if len(name) > 15:
    print("You have a long name! ") # true block
else:
    print("You have a short name! ") # false block
```

```
You have a short name!
```

Task 3 - Password Checker

This task has **3 parts**. Each section adds onto the previous part, you will hand in **THREE SEPARATE FILES**

Task 1

1) Create a program that prompts the user for their age

2) If their age is less than 14 it should produce a simple funny statement

3) If their age is greater than or equal to 14, it should produce a different statement

4) Create your program in SCRIPT mode and save it as ***Simple If Statement***

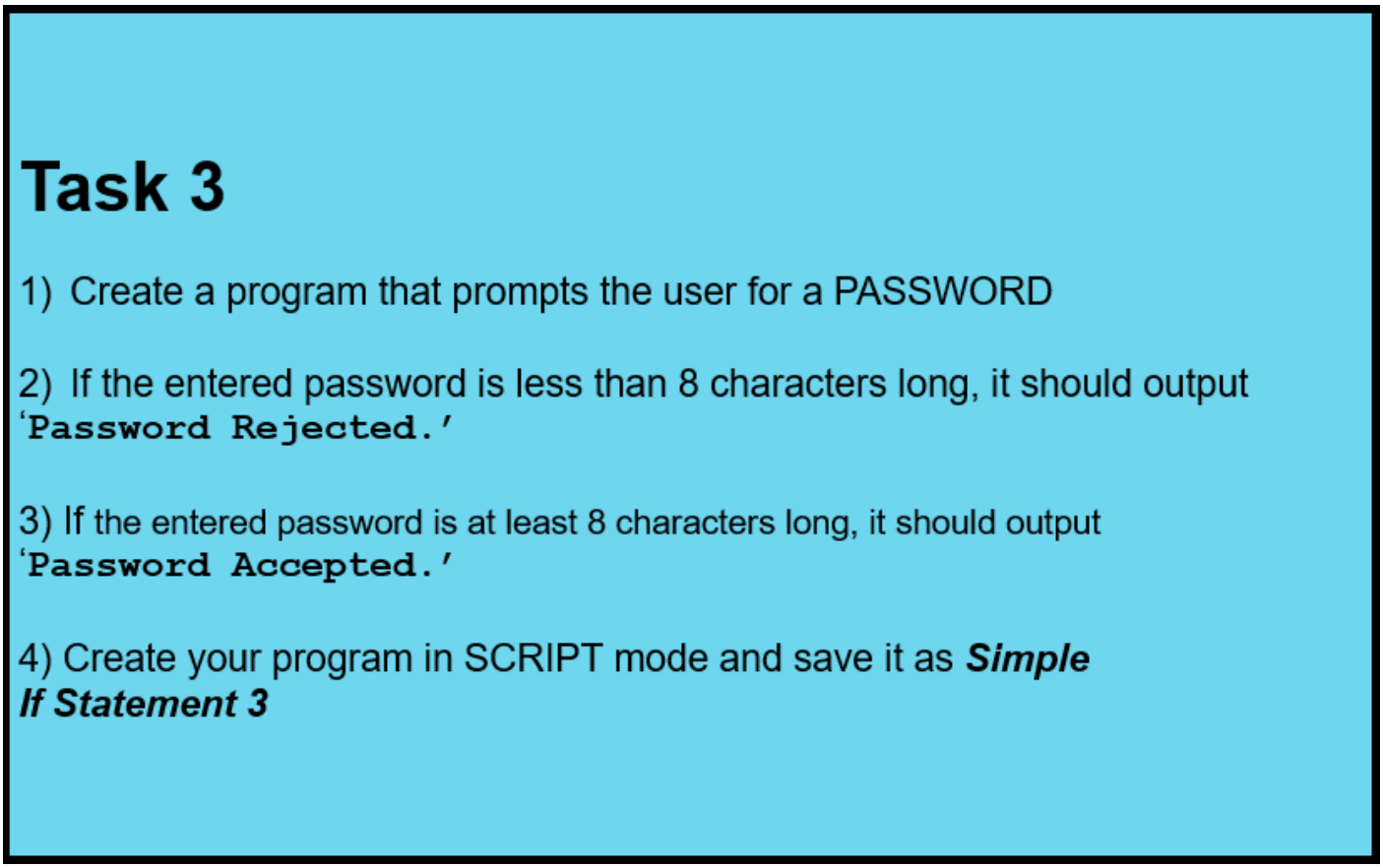
Task 2

1) Create a program that prompts the user for their **NAME** and the **YEAR** they were born.

1) If their year is after 1999, the program should output
'**Welcome NAME, you are a 21st Century Child!**'

3) If their year is not after 1999, the program should output
'**Welcome NAME, you are a 20th Century child!**'

4) Create your program in SCRIPT mode and save it as ***Simple If Statement 2***



For Loops

Repetition or Iteration: a step or sequence of steps that are repeated until some condition is satisfied.

Sometimes we want to repeat a bit of code many times. We use **loops** to achieve this.

If we wanted to output the **same** statement 3 times, there are two ways we could do this:

Method 1:

In [5]:
```
print("Hello World")
print("Hello World")
print("Hello World")
```

```
Hello World
Hello World
Hello World
```

Method 2:

In [7]:
```
for x in range(0,3):
    print("Hello World")
```

```
Hello World
Hello World
Hello World
```

For example, if you wanted to output the **same** statement **3 times**, we would use a **FOR loop**

- This is because we know the number of iterations needed

RULE: Use a FOR loop if the number of iterations is known

The **RANGE** function is useful here, as it enables us to specify a **start value** and an **end value**

range (start , stop)

Pay attention to the **SYNTAX**. Remember the amount of compile issues you had with the colon in our IF-ELSE statement? Not that the FOR loop uses one too.

How many times will "Hello World" be output below?

In [8]:
```
for x in range(1,5):
    print("Hello World")
```

```
Hello World
Hello World
Hello World
Hello World
```

Everytime a loop is made, we call this **iteration**

There were 4 iterations in the command above.

Try entering the following into your Python Interpreter:

In [ ]:
```
for x in range(5000):
    print(x)
```

Let's look at the following examples together:

In [9]:
```
# Example 1

for x in range(0,10):
    print("Hello World")
```

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

In [10]:
```
# Example 2

for x in range(0,10):
    print("Hello World")
    print("I love Python!")
```

```
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
```

In [11]:
```
# Example 3

for x in range(0,10):
    print("Hello World")
    print("I love Python!")
print("I need sleep... zzzz")
```

```
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
Hello World
I love Python!
I need sleep... zzzz
```

When learning looping, it is useful to count the number of iterations (loops).

As this for loop iterates, the variable "x" increments by 1.

We can simply output the value of "x" for each iteration!

In [12]:
```
for x in range(5):
    print("This is iteration number: ", x)
```

```
This is iteration number:  0
This is iteration number:  1
This is iteration number:  2
This is iteration number:  3
This is iteration number:  4
```

Sometimes, we can put a loop inside another loop

This is known as **NESTED Loops**

In [13]:
```
for x in range(1,3):
    print("This is the OUTER loop statement -- ", x)
    for y in range(1,4):
        print("This is the INNER loop statement -- ", y)
```

```
This is the OUTER loop statement --  1
This is the INNER loop statement --  1
This is the INNER loop statement --  2
This is the INNER loop statement --  3
This is the OUTER loop statement --  2
This is the INNER loop statement --  1
This is the INNER loop statement --  2
This is the INNER loop statement --  3
```

Things seem to be getting a little complicated, but the mechanism is exactly the same!

Look at the following source codes:

What is the expected output from the following source code? Be as exact as you can!

In [14]:
```
# For Loop Example 1

for x in range(1,4):
    print("This is the OUTER loop number: ", x)
    for y in range(1,3):
        print("This is the INNER loop number: ", x, ", ", y)
```

```
This is the OUTER loop number:  1
This is the INNER loop number:  1 ,  1
This is the INNER loop number:  1 ,  2
This is the OUTER loop number:  2
This is the INNER loop number:  2 ,  1
This is the INNER loop number:  2 ,  2
This is the OUTER loop number:  3
This is the INNER loop number:  3 ,  1
This is the INNER loop number:  3 ,  2
```

In [15]:
```
# For Loop Example 2

for y in range(5):
    print(y * '*')
```

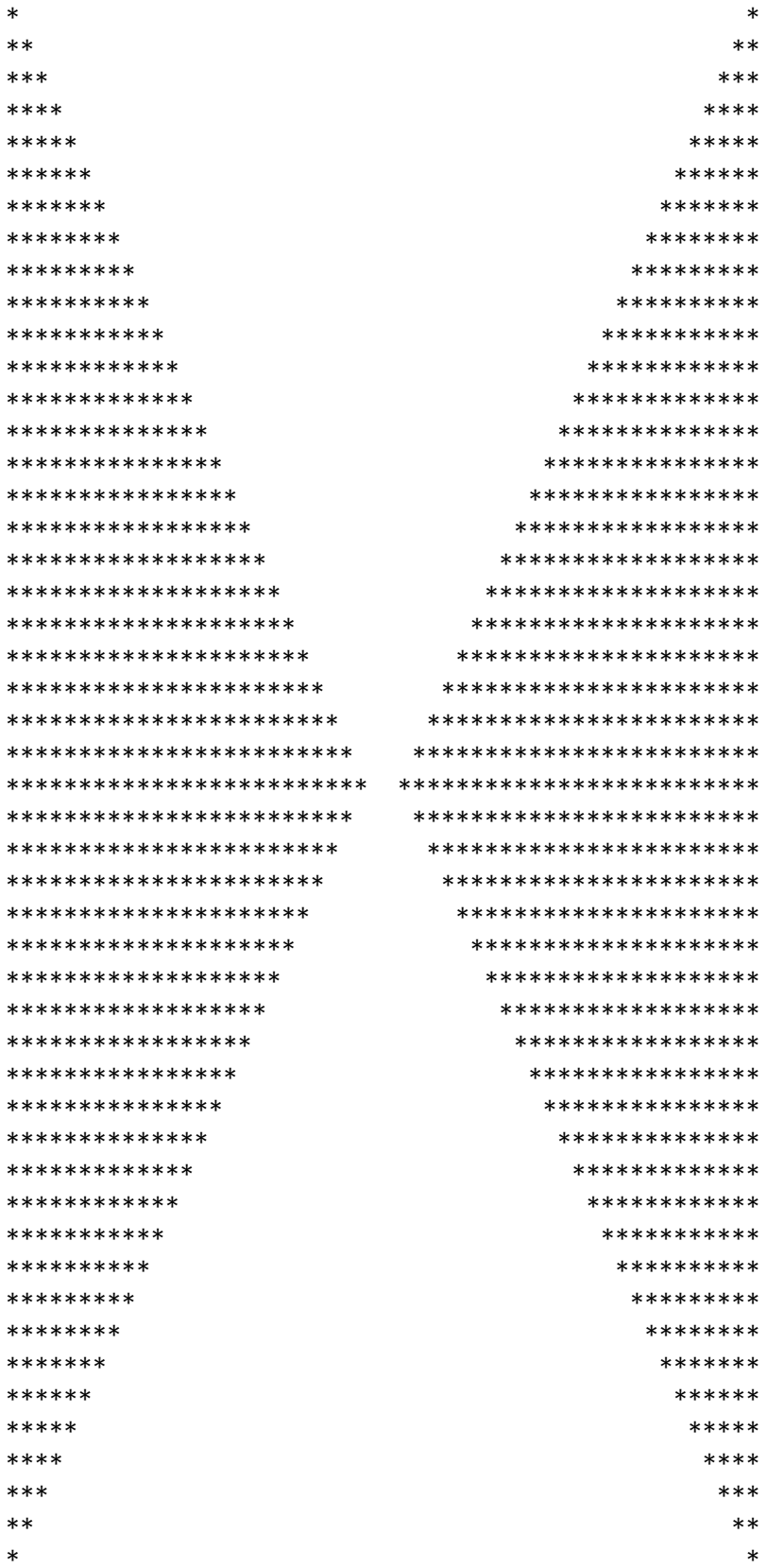
```
*
**
***
****
```

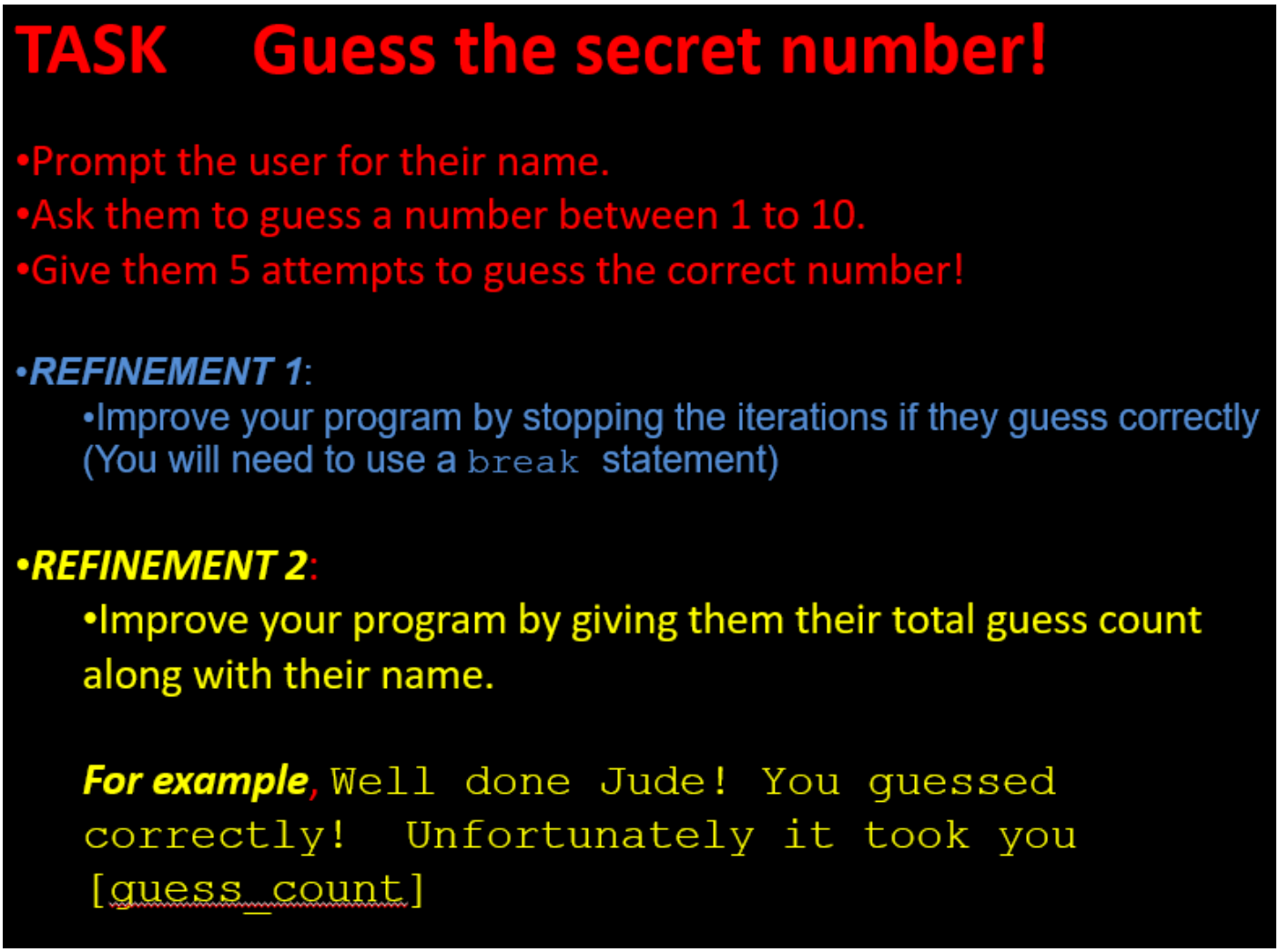
In [16]:
```
# For Loop Example 3

for y in range(5):
    print(y * '*')
    # (start, stop, step)
for y in range(5,0,-1): # this means decrement (goes down) by one after each i
teration
    print(y * '*')
```

```
*
**
***
****
*****
****
***
**
*
```

Try this out for fun

In [17]:
```
iterations = 25

for x in range(iterations):
    print('*' * x, '  ' * (iterations-x), '*' * x)

for y in range(iterations,0,-1):
    print('*' * y, '  ' * (iterations-y), '*' * y)
```





In [22]:
```
my_number = 7
their_guess = 8
guess_count = 1
name = input("What is your name? ")

for x in range(1,5):
    their_guess = int(input("Please enter a guess between 1 and 10: "))
    if their_guess == my_number:
        print("Well done " + name + ", it took you " + str(guess_count) + " gu
esses")
        break
    else:
        print("Hard luck...")
        guess_count += 1
```

```
What is your name? Mr. Wain
Please enter a guess between 1 and 10: 7
Well done Mr. Wain, it took you 1 guesses
```

Using the Random Number Generator

We can "import" very useful **modules** which can give extra functionality to our programs. One such module is the '**random** module. This is how we import it:

In [ ]:
```
from random import randint
```

In [ ]:
```
from random import randint
randint(1,100)
```

Extra Task

Using the random number generator

Copy your "Guess Secret Number" game so that it uses the randomiser to pick a random number between 1 and 10, rather than you setting it at design time.

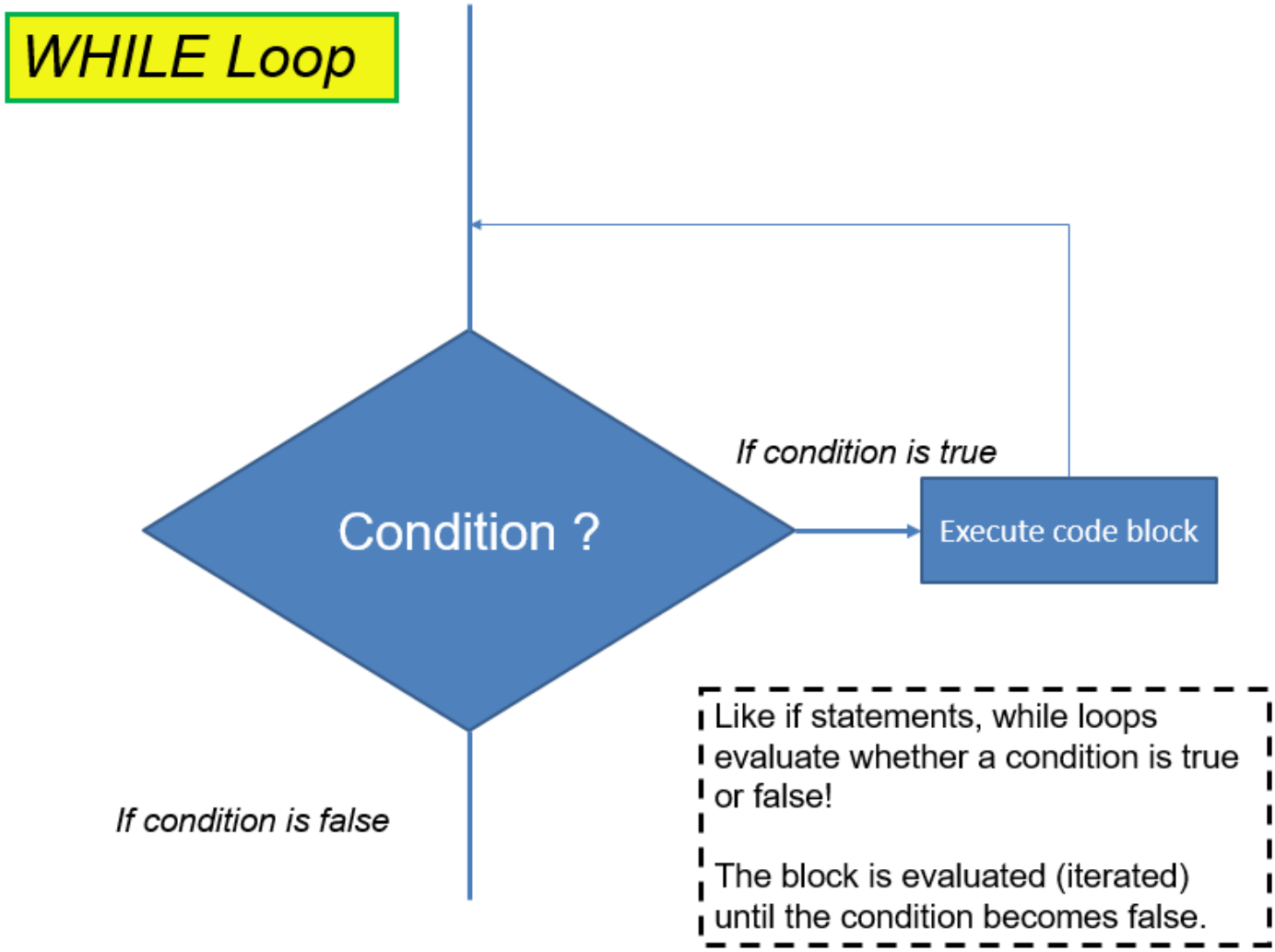
Save this as "Guess Random Secret Number". Hand in these with your original Guess Number program.

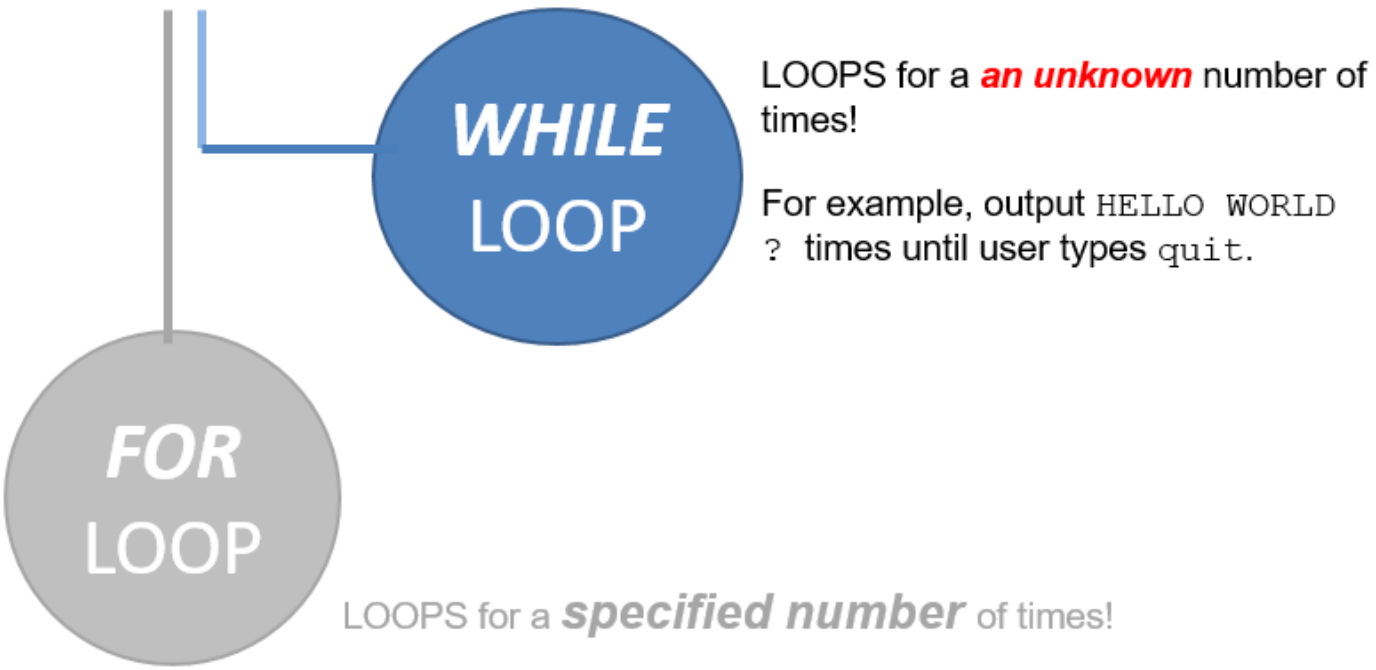
While Loops

As we know, if you know the number of iterations, you use a **FOR loop**

What if we don't know the number of iterations?

We use the WHILE Loop





While Loop Practice

What will be the output here?

In [25]:
```
name = "Ash Ketchum"

while len(name) < 20:
    name = name + "!"
    print(name)
```

```
Ash Ketchum!
Ash Ketchum!!
Ash Ketchum!!!
Ash Ketchum!!!!
Ash Ketchum!!!!!
Ash Ketchum!!!!!!
Ash Ketchum!!!!!!!
Ash Ketchum!!!!!!!!
Ash Ketchum!!!!!!!!!
```

Password Checker Example

In [26]:
```
password = "pokemon"

guess = input("Please enter your password: ")

while guess != password:
    print("Access denied")
    guess = input("Please try again: ")

print("Access granted")
```

```
Please enter your password: Gary
Access denied
Please try again: Professor Oak
Access denied
Please try again: Pokemon
Access denied
Please try again: pokemon
Access granted
```

What will the output be here?

In [27]:
```
number = 1

while number < 5:
    print(number*number)
    number += 1 #number = number + 1
```

```
1
4
9
16
```

Python Lessons

Lists and Tuples

Python Lists

A list is a container which holds comma separated values (items or elements) between square brackets where items or elements need not all have the same type.

In general, we can define a list as an object that contains multiple data items (elements). The contents of a list can be changed during program execution. The size of a list can also change during execution, as elements are added or removed from it.

In [28]:
```
list = [0,1.0,2,3,"hello", "world"]
print(list)
```

```
[0, 1.0, 2, 3, 'hello', 'world']
```

In [29]:
```
list = [0,1.0,2,3,"hello", "world"]
list.pop(4)
print(list)

list.insert(4,"goodbye")
print(list)
```

```
[0, 1.0, 2, 3, 'world']
[0, 1.0, 2, 3, 'goodbye', 'world']
```

In [30]:
```
list = [0,1.0,2,3,"hello", "world"]
list.insert(4,"goodbye")
print(list)
```

```
[0, 1.0, 2, 3, 'goodbye', 'hello', 'world']
```

In [33]:
```
top = []

for i in range(5):
    books = input("Enter your favourite book:")
    top.insert(i, books)

print(top)

pos = int(input("What book do you want to replace? "))
new = input("What book would you like to put there? ")

while pos <= 5:
    top.pop(pos)
    top.insert(pos, new)
    break

print(top)

top.insert(6,"Amulet")
print(top)
```

```
Enter your favourite book:The Lord of the Rings
Enter your favourite book:The Hobbit
Enter your favourite book:Harry Potter
Enter your favourite book:Hitchhiker's Guide to the Galaxy
Enter your favourite book:What the Dog Saw
['The Lord of the Rings', 'The Hobbit', 'Harry Potter', "Hitchhiker's Guide t
o the Galaxy", 'What the Dog Saw']
What book do you want to replace? 3
What book would you like to put there? Talking to Strangers
['The Lord of the Rings', 'The Hobbit', 'Harry Potter', 'Talking to Stranger
s', 'What the Dog Saw']
['The Lord of the Rings', 'The Hobbit', 'Harry Potter', 'Talking to Stranger
s', 'What the Dog Saw', 'Amulet']
```

Python Tuples

A tuple is a container which holds a series of comma-separated values (items or elements) between parentheses such as an (x, y) coordinate.

Tuples are like lists, except they are immutable (i.e. You cannot change its content once created), and can hold and mix data types.

In [34]:
```
tuple = (0,1,2,3)
print(tuple)
```

```
(0, 1, 2, 3)
```

Exercise 1

Write a Python program which accepts a sequence of comma-separated numbers from the user and generate a list and a tuple with those numbers.

Sample data : 3, 5, 7, 23

Output:

List: ['3', '5', '7', '23']

Tuple: ('3', '5', '7', '23')

Exercise 2

In geometry, the area enclosed by a circle of radius r is πr2. Here the Greek letter π represents a constant, approximately equal to 3.14159

Write a Python program which accepts a user input and outputs the radius and area of the circle.

Sample Output:

r = 1.1

Area = 3.8013271108436504