

# Level 1

## Programming and Iterative Processes Course Notes

### Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Variables</b>	<b>4</b>
2.1	Variables – datatypes . . . . .	5
<b>3</b>	<b>Inputs and Formatting</b>	<b>6</b>
<b>4</b>	<b>Commenting</b>	<b>7</b>
<b>5</b>	<b>Casting</b>	<b>8</b>
<b>6</b>	<b>Dinner Order - Basic Program - Design and Build</b>	<b>9</b>
6.0.1	Planning and Testing . . . . .	9
6.1	Strawberry picking . . . . .	14
<b>7</b>	<b>Conditions</b>	<b>15</b>
7.1	Activities . . . . .	17
<b>8</b>	<b>Random numbers</b>	<b>18</b>
<b>9</b>	<b>Program using conditions and random numbers</b>	<b>19</b>
9.1	Guess greater , less or equal. . . . .	19
<b>10</b>	<b>Loops</b>	<b>21</b>
10.1	Activities - improving previously made programs . . . . .	23
10.2	Higher Lower Game . . . . .	25
10.2.1	Program . . . . .	25
10.2.2	Planning, Developing, Testing . . . . .	26
<b>11</b>	<b>Validation</b>	<b>29</b>
11.1	Integer Validation . . . . .	30
11.2	String Validation . . . . .	33
<b>12</b>	<b>Lists</b>	<b>35</b>
<b>13</b>	<b>Lists and Loops</b>	<b>36</b>
13.1	Creating and modifying a class group . . . . .	36
13.2	Lucky Unicorn Game . . . . .	37

13.2.1 Program . . . . .	37
13.2.2 Planning: Iterative Processes . . . . .	38
<b>14 Functions</b>	<b>45</b>
<b>15 Code Examples</b>	<b>47</b>
15.1 Dinner Order . . . . .	47
15.2 Strawberry Picking . . . . .	48
15.3 Guess greater, less or equal . . . . .	48
15.4 Higher Lower Game . . . . .	49
<b>16 Testing</b>	<b>50</b>
<b>17 Dictionaries</b>	<b>51</b>
<b>18 Coffee</b>	<b>53</b>
<b>19 Tests</b>	<b>65</b>
<b>20 Strings</b>	<b>68</b>
<b>21 Objects</b>	<b>68</b>

# 1 Introduction

A Programming Language is a language, so it has:

- **Vocabulary**

Words that we (our culture) agree on the meaning of. If we don't know what the word means, we can use a dictionary. Dictionary definitions contain a definition (explanation of the meaning of the word) and often will include an example of how the word is used in a real situation.

Programming languages have documentation. This documentation is often quite hard to understand without experience, but many websites serve a similar purpose and give more accessible explanations.

- **Syntax**

A way of organising words so that a sentence makes sense. "The jump moon cow over" does not make sense but "The cow jumped over the moon" does.

We need to learn about how a programming language organises words so we can make it do what we want it to do.

- **Organisation systems**

A way of putting blocks of writing together. In writing we use sentences and paragraphs, in programming we have systems to keep sections of code together or apart.

Python uses tab stops as special part of its organisation system and more generally, programs will use things like functions and objects

Every program language will have an official documentation <https://docs.python.org/3/>

Understanding the documentation can be a challenge, but there are many other references and tutorial sites, such as <https://www.w3schools.com/python/>

## 2 Variables

Variables are “holders” of some data and are given a name, so that we can identify them and use them.

Quite often we use “my” in the name to make it clear that it is a variable.

We can output the value of the variables using a print statement.

```
1 my_number = 10
2 print(my_number)
3 my_next_number = 10.0
4 print(my_next_number)
5 my_new_number = "10"
6 print(my_new_number)
7
8 my_number = "ten" + "hello"
9 print(my_number)
```

Listing 1: Python example

```
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/first_class-1.py
10
10.0
10
tenhello
>>>
```

### Output

If the variables are numbers (integers or floats), we can use basic mathematical operations with them (+ , - , \* , /).

```
1 first_number = 12
2 second_number = 15
3 print(first_number + second_number)
4 print(first_number - second_number)
5 print(first_number * second_number)
6 print(first_number / second_number)
```

Listing 2: Python example

```
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/math_operations.py
27
-3
180
0.8
>>> |
```

### Output

## 2.1 Variables – datatypes

If we have a number such as 41834567, this could mean the value of 41 million .. or it just be a string of letters, for example, a phone number.

A program must know what type the variable is.

In the case above (and for lots of other situation), Python makes a “best guess”. In stricter languages you have define what type the variable is.

```
1 my_number = 10
2 print(type(my_number))
3 my_next_number = 10.0
4 print(type(my_next_number))
5 my_new_number = "10"
6 print(type(my_new_number))
7
8 my_number = "ten" + "hello"
9 print(my_number)
```

Listing 3: Python example

```
Python 3.7.2 (v3.7.2:9a3ffc0492, Dec 24 2018, 02:44:43)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/Paul/Desktop/code_activities/first_class.py ======
<class 'int'>
<class 'float'>
<class 'str'>
tenhello
>>> |
```

Output

### 3 Inputs and Formatting

The user input is held by a variable (which will be a string)

To get user input, use the `input` command and the message that will be printed in the console goes , as a string, in the brackets.

We can combine the information in the `my_string` variable with another string , by using `.format`.

The variable referred to at the end is placed in the empty bracket.

Combining strings is called **concatenation**.

```
1 my_string = input("Please enter your name -> ")
2 my_output = "The name you have entered is {}".format(my_string)
3 print(my_output)
```

Listing 4: Python example

```
>>>
=RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/input_format_output.py
Please enter your name -> Harry
The name you have entered is Harry
>>> |
```

Output

It is a good idea to be able to control output formatting very confidently:

See [https://www.w3schools.com/python/ref\\_string\\_format.asp](https://www.w3schools.com/python/ref_string_format.asp)

Experiment with the following to get an idea about what the `format` method can do

```
1 my_name = "Harry"
2 my_age = 13
3 my_output = "My name is {} and I am {} years old".format(my_name,my_age)
4 print(my_output)
5 my_output = "My name is {1} and I am {0} years old".format(my_name,my_age)
6 print(my_output)
7 my_output = "My name is {:^10} and I am {:>10} years old".format(my_name,my_age)
8 print(my_output)
9 my_number = float(2.671998761726172)
10 print(my_number)
11 print("{:.4f}".format(my_number))
12 print("{:010.2f}".format(my_number))
```

Listing 5: Python example

## 4 Commenting

When programs become more complex, it is helpful to write comments in the code.

Comments do not run in any way, they are just there for the programmer to read.

Comments are created by starting a line with a “#”.

Comments are **essential**

```
1 #initialise variables
2 my_name = "Harry"
3 my_age = 13
4 # print out variables in a sentence
5 my_output = "My name is {} and I am {} years old".format(my_name,my_age)
6 print(my_output)
7 # swap the order
8 my_output = "My name is {1} and I am {0} years old".format(my_name,my_age)
9 print(my_output)
10 # print the first one with 10 character spaces centre aligned
11 # and the second one 10 character spaces right aligned
12 my_output = "My name is {:^10} and I am {:>10} years old".format(my_name,my_age)
13 print(my_output)
14 my_number = float(2.671998761726172)
15 print(my_number)
16 # print decimal to 4 decimal places (not rounded)
17 print("{:.4f}".format(my_number))
18 # print in 10 character spaces with trailing zeroes and with 2 decimal places
19 print("{:010.2f}".format(my_number))
```

Listing 6: Python example

## 5 Casting

At times we need to change a data type and the most common situation is to turn a string into a number (float or integer) or to go the other way.

To do this we need to **cast** the variable to a new data type.

```
1 # declare two string variables
2 num_a = "123"
3 num_b = "456"
4
5 print(num_a + num_b)
6
7 # cast them as integers
8 num_a = int(num_a)
9 num_b = int(num_b)
10
11 print(num_a + num_b)
12
13 # this cast will produce a program error
14 # because the value cannot be turned into number
15
16 my_string = "Twelve"
17 my_string = int(my_string)
```

Listing 7: Python example

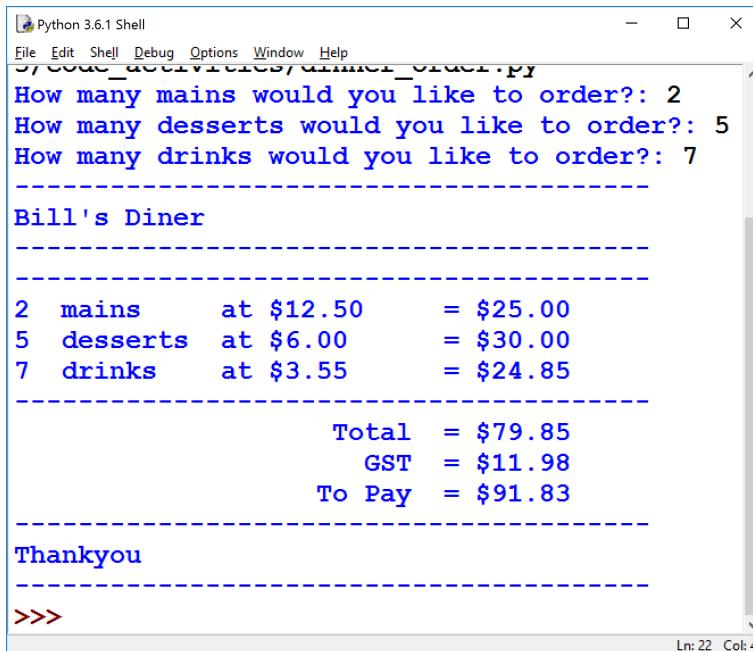
```
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/casting.py
123456
579
Traceback (most recent call last):
  File "/Users/khouripa/Documents/DTech/year11_2020/code_activities/casting.py", line 17, in <module>
    my_string = int(my_string)
ValueError: invalid literal for int() with base 10: 'Twelve'
>>>
```

## 6 Dinner Order - Basic Program - Design and Build

You need to create a program that allows a waiter to enter in the number of main meals ( at \$12.50 each ) , the number of desserts ( at \$6.00 each ) and the number of drinks ( at \$3.55 ) ordered at a table.

The program then prints out a summary of the order, a total and then adds on GST (and gives the new total ).

(Note: the program should be easy to alter so the price values should be declared as constant variables)



A screenshot of a Python 3.6.1 Shell window. The title bar says "Python 3.6.1 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, Help. The file path in the title bar is "/Users/username/Desktop/dinner\_order.py". The code in the shell window is:

```
How many mains would you like to order?: 2
How many desserts would you like to order?: 5
How many drinks would you like to order?: 7
-----
Bill's Diner
-----
2 mains      at $12.50      = $25.00
5 desserts    at $6.00       = $30.00
7 drinks      at $3.55       = $24.85
-----
      Total   = $79.85
      GST    = $11.98
      To Pay = $91.83
-----
Thankyou
-----
>>>
```

The status bar at the bottom right shows "Ln: 22 Col: 4".

An example of Dinner Order program run

### 6.0.1 Planning and Testing

Much of the planning work for a program is done on paper (and will need to be photographed and placed in a document)

The testing is done on the computer and is documented.

A basic idea of the process is given below (but we will expand on this as things go)

- **Breaking the problem down into components (or sub- problems):**

This means identifying the key things the program needs to do.

- **Writing Pseudo code**

This means sketching out in writing or diagrams how the code is going to be put together. (the writing should communicate how the code is going to work, but does not need to be the exact language)

- **Building and Trialling**

Build the program in stages and trial each stage as you go.

- **Assembled Outcome Testing**

Once you have got the program made, there needs to be some formal testing of it.

Errors need to be identified

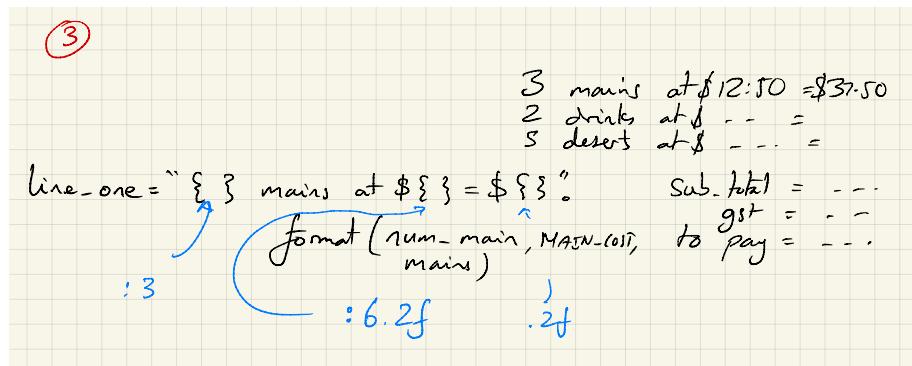
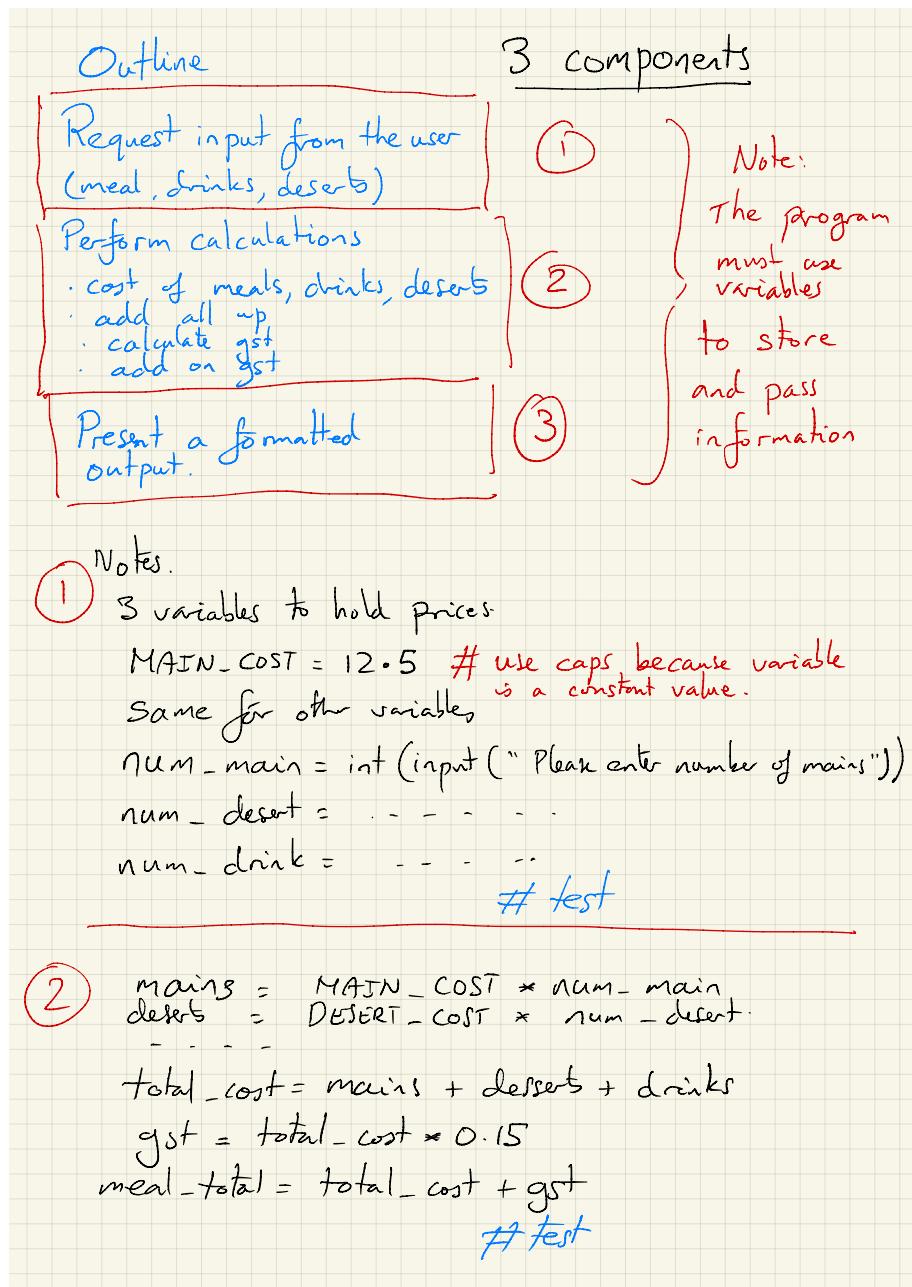
- **Usability Testing**

Ideally the program should be tested on some other user other than yourself

- **General Evaluation:**

A basic reflection

## Examples of Component Planning



## Basic Idea of Testing

When testing we consider

- Expected inputs (these are inputs we would normally expect the user to input).  
Does the program give the right outputs with the given expected inputs.
- Boundary inputs (are there maximum or minimum value that the user can enter)  
What happens at these boundaries (and if we go over them)
- Unexpected Inputs (these are character entries that we are not expecting)  
These could be just pressing enter or having spaces , using letters instead of numbers or characters like \* , & , etc.  
These could occur if the user makes a mistake or misunderstands what is expected.

Testing should have some kind of plan and documentation of the results.

Testing dinner order. (Done once the program is made, or a version of it)		
Expected Input	Boundary Input (upper, lower limit)	Unexpected Input
How many mains ? 4	How many mains ? 30	How many mains ? Three
How many deserts ? 3	How many deserts ? 0	How many deserts ? *?
How many drinks ? 2	How many drinks ? -1	How many drinks ? ↳ space.
Expected output ?	Expected output ?	Expected Output ?
Actual output ?	Actual Output ?	Actual output ?
↳ is this okay ?	↳ is this okay ?	↳ is this okay ?

## Result from expected

```
>>> 
= RESTART: /Users/Paul/Desktop/code_activities/tr
How many mains would you like to order?: 4
How many desserts would you like to order?: 3
How many drinks would you like to order?: 2
-----
Bill's Diner
-----
4 mains at $12.50 = $50.00
3 desserts at $6.00 = $18.00
2 drinks at $3.55 = $7.10
-----
Total = $75.1
GST = $11.26
To Pay = $86.36
-----
Thankyou
-----
>>> |
```

## Result from boundary

```
>>> 
= RESTART: /Users/Paul/Desktop/code_activities/tr
How many mains would you like to order?: 30
How many desserts would you like to order?: 0
How many drinks would you like to order?: -1
-----
Bill's Diner
-----
30 mains at $12.50 = $375.00
0 desserts at $6.00 = $0.00
-1 drinks at $3.55 = $-3.55
-----
Total = $371.45
GST = $55.72
To Pay = $427.17
-----
Thankyou
-----
>>> |
```

## Result from unexpected

```
type 'help', 'copyright', 'credits' or 'license()' for more information.
>>>
= RESTART: /Users/Paul/Desktop/code_activities/trinket/dinner_order_basic.py =
How many mains would you like to order?: Three
Traceback (most recent call last):
  File "/Users/Paul/Desktop/code_activities/trinket/dinner_order_basic.py", line 1
  3, in <module>
    mains = int(input("How many mains would you like to order?: "))
ValueError: invalid literal for int() with base 10: 'Three'
>>>
```

## Evaluation

- What works
  - Works properly on expected inputs
  - Presentation of receipt clearly laid out on console.
- What could be improved
  - Should have an upper limit on how many meals can be entered
  - Should not allow entries below zero
  - Unexpected inputs cause a program crash
  - At the moment the program only runs once (so need a way to enter a new order)

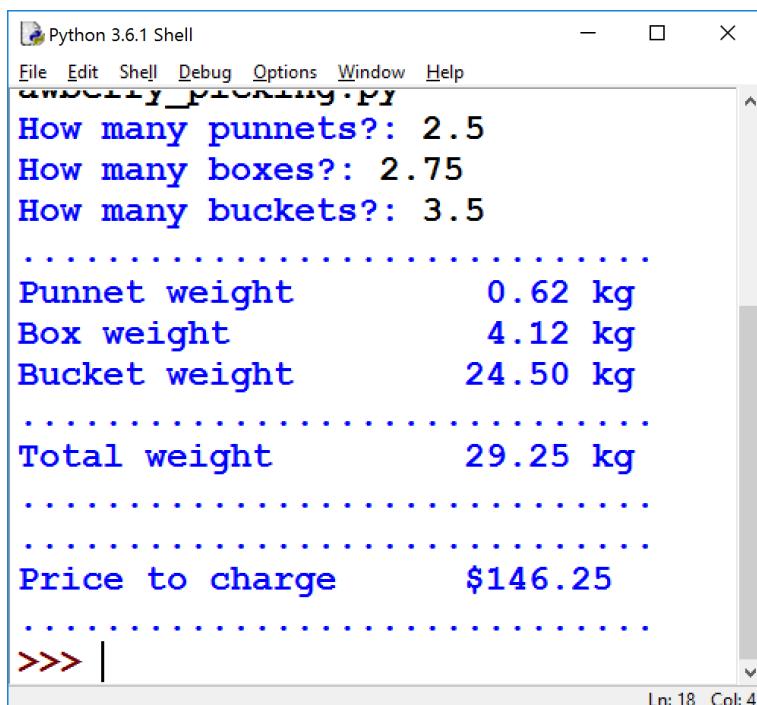
## 6.1 Strawberry picking

A strawberry farm lets families come and pick strawberries.

They can fill boxes, buckets or punnets (any combination).

- They charge a rate of 5 \$/kg.
- Full of strawberries, a punnet weighs 250 g, a box weighs 1.5 kg and a bucket weighs 7 kg.

Design a program that calculates the amount to be charged given a certain number of boxes, punnets and buckets have been picked.



The screenshot shows a Python 3.6.1 Shell window. The user has run a script named 'strawberry\_picking.py'. The program prompts for the number of punnets, boxes, and buckets. It then calculates the weight of each type of container and the total weight. Finally, it calculates the price based on the total weight and the rate of 5 dollars per kilogram.

```
Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
strawberry_picking.py
How many punnets?: 2.5
How many boxes?: 2.75
How many buckets?: 3.5
.
.
.
Punnet weight      0.62 kg
Box weight         4.12 kg
Bucket weight     24.50 kg
.
.
.
Total weight       29.25 kg
.
.
.
Price to charge   $146.25
.
.
.
>>> |
```

Ln: 18 Col: 4

An example of Strawberries program run

## 7 Conditions

The solutions to many programming problems require an action to occur only if a particular condition is true.

A condition can only result in true or false.

(These values are called Booleans. True and False are reserved words in the Python language. True and False are values / objects of type bool)

A condition often involves an expression which compares one value with another.

Comparisons between numerical values are made using the same comparison operators that are used in mathematics:

>	greater than
$\geq$	greater than or equal to
<	less than
$\leq$	less than or equal to

and the equality / inequality operators:

$= =$	equal to ( the proper equals operator)
$! =$	not equal to

### Boundary cases

Human languages are imprecise.

Does "every child over 5" mean "aged 5 and over" or "aged 6 and over"? The difference between  $\geq$  and  $>$  is very important.

If you mean "every child aged 5 and over" you can express this as age  $> 4$  or age  $\geq 5$ .

Any lack of clarity needs to be removed at the design phase. Always clarify before coding. The boundary cases are where errors in programming often occur.

Boundary cases are the values at and just outside the specified limits.

In the case "aged 5 and over" the specified limit is 5 and the age values of 4 and 5 are the boundary cases.

Only one condition can execute

```

my_variable = 89

# example 1
if my_variable < 90:
    print("{} is less than 90".format(my_variable))

#example 2
if my_variable < 89:
    print("{} is less than 89 ".format(my_variable))

#example 3
if my_variable < 89:
    print("{} is less than 89".format(my_variable))
else:
    print("{} is not less than 89".format(my_variable))

#example 4
if my_variable < 89:
    print("{} is less than 89".format(my_variable))
elif my_variable == 89:
    print("{} is exactly 89".format(my_variable))
else:
    print("{} is more than 89".format(my_variable))

```

Only one condition can execute

The code after the colon is “tabbed in” by 4 spaces.  
The tabbed in code is what will run if the condition is met.

Running the code on pytutor : <http://www.pythontutor.com/visualize.html>

Python 3.6

```

1 my_variable = 89
2
3 # example 1
4 if my_variable < 90:
5     print("{} is less than 90".format(my_variable))
6
7 #example 2
8 if my_variable < 89:
9     print("{} is less than 89 ".format(my_variable))
10
11 #example 3
12 if my_variable < 89:
13     print("{} is less than 89".format(my_variable))
14 else:
15     print("{} is not less than 89".format(my_variable))
16
17 #example 4
18 if my_variable < 89:
19     print("{} is less than 89".format(my_variable))
20 elif my_variable == 89:
21     print("{} is exactly 89".format(my_variable))
22 else:
23     print("{} is more than 89".format(my_variable))

```

Print output (drag lower right corner to resize)

```

89 is less than 90
89 is not less than 89
89 is exactly 89

```

Frames Objects

Global frame  
my\_variable | 89

line that has just executed  
next line to execute  
Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.

<< First | < Back | Program terminated | Forward > | Last >>

## 7.1 Activities

1. Consider a program that asks for a user's age and then:
  - if their age 12 or less, tells them that they should be at primary school,
  - if their age is more than 12 but less than 18, they need to go to secondary school
  - if they are 18 years old, it tells them that they have become an adult
  - and if they are older than 18 tells them that they must have left school.
2. To go to university a person needs to be:
  - over the age of 20
  - **or** be 16 or older and have level 3 NCEA
  - **and** (for both conditions) the person need to be competent with English.

Create a program that given the following variables:

age= 25

ncea = False

english = True

Will give a correct response about whether a person can go to university or not.

Test with other values.

## 8 Random numbers

Random numbers are particularly useful for games and quizzes as they mean that the program will not always do things in the same order.

To get random numbers we need to import a **library** that handle random numbers.

Python has a large number of libraries that extend the functionality of the programming language.

```
1 # import the random number library
2 import random
3
4 #random number between 0 and 1
5 print(random.random())
6
7 # random number between two integers (inclusive)
8 print(random.randint(2,5))
9
10 # random number between two integers (inclusive)
11 # but with a step value (in this case will only get multiples
12 #of 5
13 print (random.randrange(0, 101, 5))
14
15 # create a list and randomly select a name from it
16 myList = ["Anna", "Michael", "Sarah", "Jessie"]
17 print(random.choice(myList))
```

Listing 8: Python example

# 9 Program using conditions and random numbers

## 9.1 Guess greater , less or equal.

- Create a program that :

Generates a random number between two integers.

- It calculates the middle value of the two integers.
- It then asks the user to choose whether the random number is Less than the middle number or More than or Equal to the middle number.
- It then gives feedback, saying whether they have won or lost

- Planning and Testing

- Break the problem into components (sub problems)
- Write a Psuedo code outline
- When the program is made, create and complete a full test table
- Introduce a small modification to the program to help you test it.

The screenshot shows a Mac OS X window titled "Python 3.8.3 Shell". The console output is as follows:

```
Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
[Clang 6.0 (clang-600.0.57) on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/trin
ket/random_choose_L_E_H.py
A random number has been chosen between 4 and 15
-----
Enter L if you think the number is less than 10
Enter G if you think the number is more than 10
Enter E if you think the number is equal to 10 : G
you lose
The number was 10
>>> |
```

Ln: 13 Col: 4

An example of greater , less , equal program run

Pseudo code plan: for Guess greater, less, equal.

### Pseudo Code

import random module

declare 2 variables to hold minimum & maximum number.

e.g. lower = 10  
higher = 20

generate a random integer and hold in my-random variable.

calculate middle number.  
e.g. mid-num =  $\frac{\text{lower} + \text{higher}}{2}$   
round.

print instructions to user.

get user input for their guess.

guess = input(message).

guess will be "L" or "E"  
or "G"

print error message if something else entered.

| Test if guess is correct  
| or not.

| if guess is "L" and my-random < mid-num  
| print ("win")

| elif guess is "G" and my-random > mid-num  
| print ("win")

| elif guess is "E" and my-random == mid-num  
| print ("win")

| else:  
| print (lose).

| End program.

Test Table: lower = 10 , higher = 20  $\Rightarrow$  middle number = 15

### Test Grid

my-random	user input	expect	actual
12	L	win	
12	G	lose	
12	E	lose	
12	H	error message	
12	T	error message	
12	l	error message	
12	g	error message	
15	L	lose	
15	E	win	
20	G	win	
20	L	lose	
20	E	lose	

Pseudo code and test grid

# 10 Loops

Often we want the program to repeat some kind of operation or to repeatedly run a section of code.

To do this we use loops.

There are **for** loops and **while** loops and we will focus on the while loop.

This while loop uses a counter to know how many times to run.

```
1 # loop with a counting condition
2 counter = 0
3 while counter < 30:
4     print("inside counter loop and counter value is {}".format(counter) )
5     # add one to the counter variable
6     counter += 1
7
8 # once the counter gets to 30 the loop stops and the resto of the program runs
9 print("Have now finished the loop")
```

Listing 9: Python example

```
inside counter loop and counter value is 0
inside counter loop and counter value is 6
inside counter loop and counter value is 7
inside counter loop and counter value is 8
inside counter loop and counter value is 9
inside counter loop and counter value is 10
inside counter loop and counter value is 11
inside counter loop and counter value is 12
inside counter loop and counter value is 13
inside counter loop and counter value is 14
inside counter loop and counter value is 15
inside counter loop and counter value is 16
inside counter loop and counter value is 17
inside counter loop and counter value is 18
```

Section of the output

```
1 # the condition
2 name = "Jessie"
3 while name == "Jessie":
4     print("inside the loop")
5     name = input("Please enter name: ")
6
7 print("have now left the loop")
```

Listing 10: Python example

```

>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/loop_condition.py
inside the loop
Please enter name: Jessie
inside the loop
Please enter name: Jessie
inside the loop
Please enter name: Bob
have now left the loop
>>> |

```

```

1 continue_program = True
2 while continue_program == True:
3     print("This where the program runs")
4     # as if user wants to run the program (or part of it) again
5     user_response = input("Would you like to play again? (Y/N) - > ")
6     if user_response == "Y":
7         print("Here we go again")
8     elif user_response == "N":
9         print("Thankyou for playing")
10    continue_program = False
11 else:
12     print("Your input was not recognised (and we should do something about
13     this)")
14 print("The program has ended")

```

Listing 11: Python example

```

= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/loop_continue.py
This where the program runs
Would you like to play again? (Y/N) -> Y
Here we go again
This where the program runs
Would you like to play again? (Y/N) -> Y
Here we go again
This where the program runs
Would you like to play again? (Y/N) -> Y
Here we go again
This where the program runs
Would you like to play again? (Y/N) -> j
Your input was not recognised (and we should do something about this)
This where the program runs
Would you like to play again? (Y/N) -> k
Your input was not recognised (and we should do something about this)
This where the program runs
Would you like to play again? (Y/N) -> N
Thankyou for playing
The program has ended

```

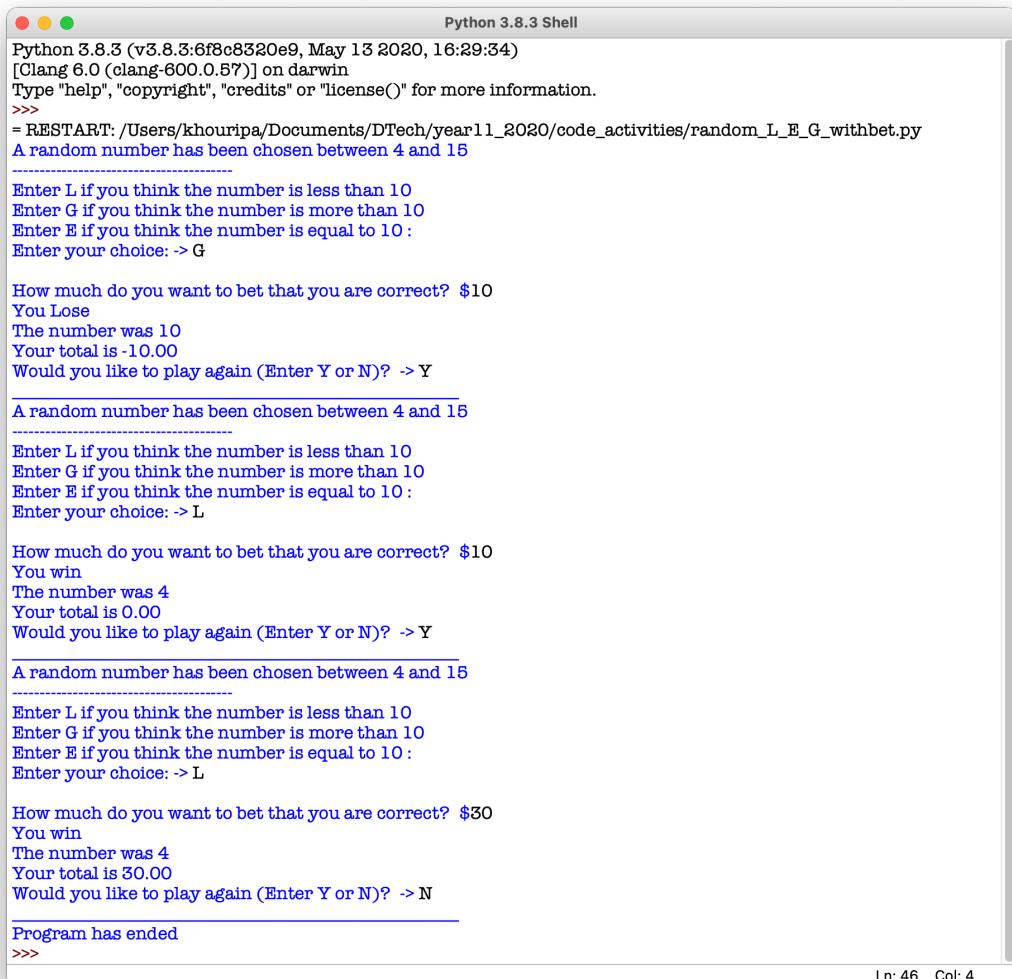
## 10.1 Activities - improving previously made programs

Take the Dinner Order and the Guess greater less or equal programs and make a new version so that they can run more than once.

See example: [https://sites.google.com/marsden.school.nz/yr11digitaltech2020/computer-program-loops?#h.p\\_XCtf-2lKZ-CK](https://sites.google.com/marsden.school.nz/yr11digitaltech2020/computer-program-loops?#h.p_XCtf-2lKZ-CK)

There is considerable scope now for extension: Consider:

- Dinner order prints out total cost of all orders when the program finishes (and/or total mains, deserts, drinks ordered)
- Guess greater less or equal prints out total score of wins at the end of the program
- Guess greater less or equal allows the user to bet each time and feeds back on winnings/losses



The screenshot shows a terminal window titled "Python 3.8.3 Shell". It displays three separate runs of a script named "random\_L\_E\_G\_withbet.py". Each run involves a random number being chosen between 4 and 15, and the user is prompted to guess if the number is less (L), greater (G), or equal (E) to 10. The user's choice and a bet amount are entered. The program then reveals the actual number and calculates the user's total score, which starts at \$10 and decreases with each loss. The user is asked if they want to play again (Y or N). The process repeats for three rounds, with the final message indicating the program has ended.

```
Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
[Clang 6.0 (clang-600.0.57) on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/random_L_E_G_withbet.py
A random number has been chosen between 4 and 15
-----
Enter L if you think the number is less than 10
Enter G if you think the number is more than 10
Enter E if you think the number is equal to 10:
Enter your choice: -> G

How much do you want to bet that you are correct? $10
You Lose
The number was 10
Your total is -10.00
Would you like to play again (Enter Y or N)? -> Y

A random number has been chosen between 4 and 15
-----
Enter L if you think the number is less than 10
Enter G if you think the number is more than 10
Enter E if you think the number is equal to 10:
Enter your choice: -> L

How much do you want to bet that you are correct? $10
You win
The number was 4
Your total is 0.00
Would you like to play again (Enter Y or N)? -> Y

A random number has been chosen between 4 and 15
-----
Enter L if you think the number is less than 10
Enter G if you think the number is more than 10
Enter E if you think the number is equal to 10:
Enter your choice: -> L

How much do you want to bet that you are correct? $30
You win
The number was 4
Your total is 30.00
Would you like to play again (Enter Y or N)? -> N

Program has ended
>>>
```

Greater less equal, with loop and betting

Pseudo code for G.E.L with loop and bet.

input random module

set lower = # limits for range  
higher = of numbers.

set total-money = 0

# hold the total result of the bets.

start game loop

generate random number  
calculate middle number

print instructions.

request user guess.

request user bet.

set win = False.

evaluate user input

if correct set win=True

if invalid input set win = ""

check win True or False or ""

if True

print "win"

add bet value to total money

total-money + bet

elif False

print lose

subtract bet value from total money

else:

print "No result"

Print what the number was  
Print total money in account

ask user if they want to play again

if ~~NO~~

set game loop to false.

print thank you for playing.

Pseudo code: Greater less equal, with loop and betting

## 10.2 Higher Lower Game

```
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/higher_lower.py
Please enter your name Paul
Hello Paul, you are about to play a guessing game
where you need to guess the secret number between 1 and 50
Testing help: the secret number is 37
Press enter to start
Please guess the secret number between 1 and 50 25
Your guess is too small
Please guess the secret number between 1 and 50 37
You have found it!!
You took 2 tries to find it
You are really fast
Do you want to play again? y/n  n
The program has ended
>>>
```

### 10.2.1 Program

#### Scenario:

You have a younger family member who loves the ‘higher / lower’ game so you decide to make a game that they can play whenever they want.

---

#### Your game should:

- Ask for the user’s name (and address them by their name)
  - Generate a ‘secret’ number between 1 and 100 and then ask the user to guess the number.
  - Tell the user if their guess is ‘too high’ or ‘too low’.
  - Then asks them to guess again (until they find the number)
  - If the user correctly guesses the number, the game should congratulate them.
  - Tell the user how many goes it took them to guess the number.
  - If they don’t guess the number in 10 goes, they should be told that they have lost the game and the mystery number should be revealed.
- 

#### Improvements:

(do not worry about these until you have completed the main requirements):

Create small test programs to work out how to solve the problem before you include them in your next program version.

- Allow the user to choose the lowest and highest number that will be used to generate the secret number between.

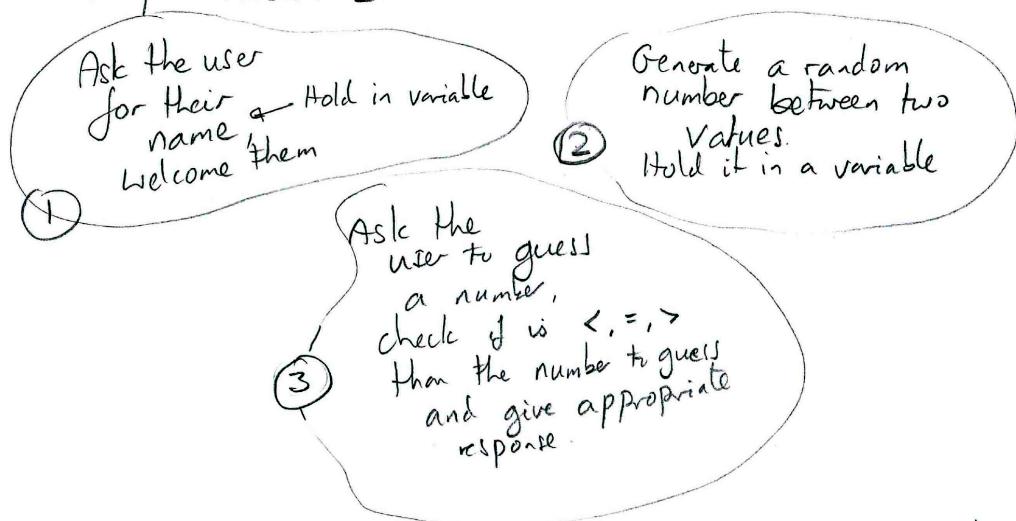
- Ideally the game should be set up so that users can't guess the same "wrong" number twice  
()This will require collecting a list of the guessed numbers).
  - Either allow the user to choose the number of guesses or set a sensible limit based on the difference between the highest and lowest number.
  - Keep track of the number of rounds played and the user's scores.
  - Improve the **usability** of the game by **validating** the user entry to make sure that it is appropriate.
- 

### 10.2.2 Planning, Developing, Testing

- **On paper** you need to break the big problem of making the program into a set of smaller sub-problems
  - If a problem looks tricky or there may be more than one way to solve it, you should make small test programs
- Sketch out psuedo code for the main things you are going to make  
( ideally before you make it : ) )
- Don't create the program all in one go, build the simplest **version** you can.
  - Then test it and then plan and build the next version.
  - After each version you should make some simple notes about what works, what could be improved , what the aim for the next stage is.
  - Then plan, build and test the next **version**
  - Repeat this process to **iteratively** develop the program.

## Paper Planning for Higher Lower Game.

- Main subproblems/components of a simple version 1.



Build in steps to get



Test

Gives version 1.

- Next level

Subproblem.

Be able to  
Repeat the guessing process.

keeping doing the guessing requires a loop

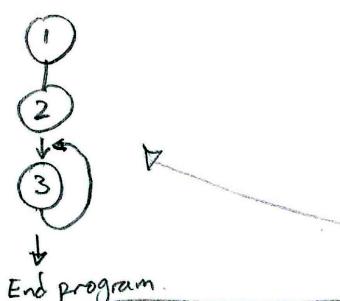
Loop conditions

what is needed for the loop

to run → The user has not guessed correctly

→ The user has not guessed 10 times.

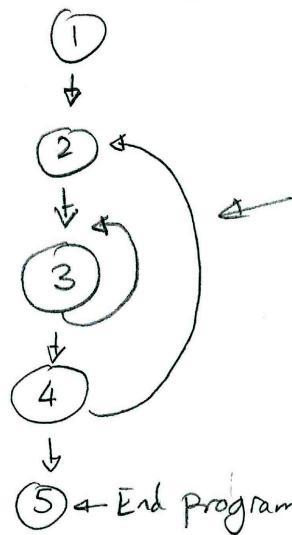
Version 2



Test

- Next level Subproblem.

Be able to repeat the whole game



outer loop that holds  
whole main part of  
the program.

Ask the user if they want  
to play again  
Ask for yes or no input  
and evaluate this whether  
to stop the program or  
not.

### Version 3

Test

Next level subproblem  
improve the user experience

- Make sure the user has entered an appropriate name. If they make a mistake (e.g. press <enter>) ask them to enter again.
- Make sure the user enters an appropriate number (not below the smallest number, not above the highest number).

Make sure the number is a number (5 rather than "Five").

### Version 4

Test

# 11 Functions

Functions allow us to package up blocks of code in a separate place and re-use them whenever we need them.

- A function is made using the “def” statement
- It is then given any logical name (no spaces, no numbers in the name)
- A function has values that can be given to it. These are placed as variable names in the round brackets after the name.
- we can then use these values “inside” the function
- the function can be “called” (made to do its thing) by writing the function name in the main body of code (with values)
- functions can send values back to the main part of the program using the “return” statement.

```
1 def adding(X,Y):  
2     A = X  
3     B = Y  
4     C = X + Y  
5     output = "{} + {} = ".format(A,B)  
6     answer = int(input(output))  
7     if answer == C:  
8         print("answer is correct")  
9     else:  
10        print("Sorry, the answer was {}".format(C))  
11  
12 def subtracting(X,Y):  
13     A = X  
14     B = Y  
15     C = X - Y  
16     output = "{} - {} = ".format(A,B)  
17     answer = int(input(output))  
18     if answer == C:  
19         print("answer is correct")  
20     else:  
21        print("Sorry, the answer was {}".format(C))  
22  
23 def multiplying(X,Y):  
24     A = X  
25     B = Y  
26     C = X * Y  
27     output = "{} {} {} = ".format(A,chr(215),B)
```

```

28     answer = int(input(output))
29
30     if answer == C:
31         print("answer is correct")
32     else:
33         print("Sorry, the answer was {}".format(C))
34
35
36
37
38
39
40
41
42
43
44
45
46 # calling the functions
47 # we can do this as often as we like with lots of different values
48 adding(70, 30)
49 subtracting(70, 30)
50 multiplying(80, 40)
51 dividing(80,40)

```

Listing 12: Functions

**70 + 30 = 80**  
**Sorry, the answer was 100**  
**70 - 30 = 40**  
**answer is correct**  
**80 × 40 = 3200**  
**answer is correct**  
**80 ÷ 40 = 3**  
**Sorry, the answer was 2.0**  
**>>>**

Functions output

## 12 Validation

When a user is making an input they may make an error and enter an **invalid** input.

This is an input that the program is not expecting:

Examples include:

- Entering a string instead of a number (such as “Two” or “Tracy” when the program is expecting an integer or float).
- Entering nothing or a single letter , when the program wants a full name.
- Entering a letter “outside of bounds” (For example: the program expects one of A,B,C,D and the user enters F)
- Entering a number outside of bounds (For example, the program expects a number between 1 and 10 and the user enters 11).
- The program expects a single letter (“y” or “n”), but is given a word (“Yes”, “No”).
- The program expects a capital letter (“A”) and is given a lower case letter (“a”).

Some of these entries can cause a program crash or just lead to poor “performance” of the program (for example: rejecting a user entry when what the user intended was valid).

The aim is always to make the program perform as well as possible and to be **User Friendly**.

To manage this **Usability** we need to **Validate** the user input.

This means one or both of two things:

- If the user input is invalid, communicate the problem to the user and ask the user again for the entry
- Evaluate and possibly alter the user entry, so that its meaning is not changed but that it is formatted in a way that program expects. (An example of this is changing “A” to “a”. In this case the user does not need to be informed)

## 12.1 Integer Validation

Consider an input request “How many hamburgers would you like to order?”

- We would expect this to be an integer with a value of 0 or more.
- Most likely we would like to “operate” on this later in the program (for example: multiply it by the cost of the hamburger to get a total price)

The input is always read as a string, so we need to cast the input to an integer.

```
1 hamburger_price = 7.5
2 #note the casting int(      ----      )
3 #this converts the string input to an integer
4 num_burgers = int(input("How many hamburgers would you like? -> "))
5
6 total_cost = num_burgers*hamburger_price
7 print("Please pay ${:.2f}".format(total_cost))
```

Listing 13: Python example

This works properly assuming the user enters a valid input.

This assumption is not good enough, and we need to deal with situations like the user entering <Enter> or 3.5 or “six”.

If they do, we get a program crash: To manage integer casting, we need a “try except” statement.

```
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/validation_int_2.py
How many hamburgers would you like? -> 3.5
Traceback (most recent call last):
  File "/Users/khouripa/Documents/DTech/year11_2020/code_activities/validation_int_2.py", line 4, in <module>
    num_burgers = int(input("How many hamburgers would you like? -> "))
ValueError: invalid literal for int() with base 10: '3.5'
>>>
```

This gets the program to do a check before accepting the cast.

The “except ValueError” will run if the input is not an integer.

```
1 hamburger_price = 7.5
2
3 try:
4     num_burgers = int(input("How many hamburgers would you like? -> "))
5 except ValueError:
6     print("The entry is not a valid integer")
7
8 #total_cost = num_burgers*hamburger_price
9 #print("Please pay ${:.2f}".format(total_cost))
```

Listing 14: Python example

```

>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/validation_int_2.py
How many hamburgers would you like? -> 3.5
The entry is not a valid integer
>>> |

```

This partly fixes the problem, but we still don't have an acceptable value for the number of hamburgers.

So we need to ask the user again for an input and if we want to ask again (and maybe again ....) we need a loop:

```

1 hamburger_price = 7.5
2 #set a condition for the loop
3 have_value = False
4 while have_value == False:
5     try:
6         num_burgers = int(input("How many hamburgers would you like? -> "))
7     except ValueError:
8         # a pleasant, helpful error message.
9         print("Unfortunately you have not given a whole number for the hamburgers
. Please try again")
10        # introduce a 'continue', this means the program will jump straight back
11        # to the top of the loop.
12        continue
13    #if we get to this point then we have a valid integer, so end the loop
14    have_value = True
15
16 # we now know that we have what we need, so can proceed
17 total_cost = num_burgers*hamburger_price
18 print("Please pay ${:.2f}".format(total_cost))

```

Listing 15: Python example

```

= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/validation_int_3.py
How many hamburgers would you like? -> 3.5
Unfortunately you have not given a whole number for the hamburgers. Please try again
How many hamburgers would you like? ->
Unfortunately you have not given a whole number for the hamburgers. Please try again
How many hamburgers would you like? -> Three hundred
Unfortunately you have not given a whole number for the hamburgers. Please try again
How many hamburgers would you like? -> 4
Please pay $30.00
>>> |

```

We still have a problem that the program accepts negative numbers:

```

>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/validation_int_3.py
How many hamburgers would you like? -> -30
Please pay $-225.00
>>> |

```

So we can extend the validation loop

```
1 hamburger_price = 7.5
2 #set a condition for the loop
3 have_value = False
4 while have_value == False:
5     try:
6         num_burgers = int(input("How many hamburgers would you like? -> "))
7     except ValueError:
8         # a pleasant, helpful error message.
9         print("Unfortunately you have not given a whole number for the hamburgers
. Please try again")
10        # introduce a 'continue', this means the program will jump straight back
11        to the top of the loop.
12        continue
13        # this will now request re-entry if the number of burgers is outside an
14        acceptable range
15        if num_burgers < 1 or num_burgers > 10:
16            print("Please enter at least one burger and no more than 10")
17            continue
18        #if we get to this point then we have a valid integer, so end the loop
19        have_value = True
20
21 # we now know that we have what we need, so can proceed
22 total_cost = num_burgers*hamburger_price
23 print("Please pay ${:.2f}".format(total_cost))
```

Listing 16: Python example

```
>>>
=RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/validation_int_3.py
How many hamburgers would you like? ->
Unfortunately you have not given a whole number for the hamburgers. Please try again
How many hamburgers would you like? -> 3.5
Unfortunately you have not given a whole number for the hamburgers. Please try again
How many hamburgers would you like? -> 0
Please enter at least one burger and no more than 10
How many hamburgers would you like? -> 11
Please enter at least one burger and no more than 10
How many hamburgers would you like? -> 9
Please pay $67.50
>>> |
```

## 12.2 String Validation

If we require a name, we still need to check that we are getting something acceptable. A simple validation can be to check that the entry has at least 2 characters and no more than (maybe) 20 (what do you think is the longest number of characters that can be in a name?).

```
1 # set a loop variable
2 have_name = False
3
4 while have_name == False:
5     # request name
6     name = input("Please enter your name: -> ")
7
8     # check the number of characters in the name and
9     # print error message if it is outside of an acceptable range.
10    # and continue to top of loop if it is
11    if len(name)< 2 or len(name) > 20:
12        print("This doesn't look like a valid name, please try again")
13        continue
14
15    # if have got here, then all okay so end loop
16    have_name = True
17
18 print("Welome {}".format(name))
```

Listing 17: Python example

```
>>>
=RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/validation_name.py
Please enter your name: -> p
This doesn't look like a valid name, please try again
Please enter your name: -> ksdhksfjhkdsfhkdhfkdfhkdfhkhkdsfh
This doesn't look like a valid name, please try again
Please enter your name: -> Paul
Welome Paul
>>> |
```

This validation checks that the characters in the name are all acceptable (i.e no numbers or special characters).

It does this by checking each character in the name against a list of characters than are acceptable.

```
1 chars = ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r",
2     "s", "t", "u", "v", "w", "x", "y", "z", " ", "-"]
3 have_name = False
4
5     name = input("Please enter your name: -> ")
6     # check name lenght and continue to the top
7     if len(name)< 2 or len(name) > 20:
8         print("This doesn't look like a valid name, please try again")
```

```

9     continue
10    # assume all the characters are acceptable
11    chars_okay = True
12    # loop through the letters in the name (that have been set to lower case)
13    for x in name.lower():
14        # if an unacceptable character is found
15        # print a useful message
16        # set chars_okay to be false
17        if x not in chars:
18            print("{} is not an acceptable character in a name.".format(x))
19            chars_okay = False
20    # once the letter loop is finished
21    # check if chars_okay is false and if it is
22    # continue to top of loop
23    if chars_okay == False:
24        continue
25
26    # if no problems
27    have_name = True
28
29 print("Welcome {}".format(name.title()))

```

Listing 18: Python example

```

>>>
=RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/validation_name_1.py
Please enter your name: -> mary5
5 is not an acceptable character in a name.
Please enter your name: -> may@logs.com
@ is not an acceptable character in a name.
. is not an acceptable character in a name.
Please enter your name: -> mary anne
Welcome Mary Anne
>>> |

```

- Checking that strings follow an acceptable “pattern” is called ”regular expressions” and this is outside of the scope of Level 1 (but we can explore options like in the code example).

[https://www.w3schools.com/python/python\\_regex.asp](https://www.w3schools.com/python/python_regex.asp)

- It is also a good idea to be familiar with “string methods” as they can be very useful.

[https://www.w3schools.com/python/python\\_strings\\_methods.asp](https://www.w3schools.com/python/python_strings_methods.asp)

## 13 Lists

Quite often we want to store a whole set of items (for example, names). Having a variable for each one is not very helpful (particularly if we want to add more names to the lists or remove some names). Suppose we have a small class of 6 students and we want to store all of their names.

Their names are: Agatha , Melinda, Harriet, Hine, Dandelion and Te Aroha

To create a list we use square brackets and commas.

```
1 # each item in the list has an index number 0,1,2,3,4,5
2 # the first item is at index 0 (Agatha)
3 # the length of the list is 6 (because it has 6 items in it)
4 # but the last index number is 6-1 = 5 ( because the counting starts from 0 )
5
6 my_class = ["Agatha" , " Melinda" , "Harriet" , "Hine" , "Dandelion" , "Te Aroha"]
```

Listing 19: Python example

To manage lists, there are a large range of methods that can be used.

See the examples below.

```
1 my_class = ["Agatha" , "Melinda" , "Harriet" , "Hine" , "Dandelion" , "Te Aroha"]
2
3 #print out a name at a specific index (use the square bracket notation)
4 # this prints out the fourth person in the last
5 print(my_class[3])
6 print()
7 # print the list out "informally"
8 print(my_class)
9 print()
10 # print using a loop (this is the usual way to do it)
11 # this a shorthand way of doing it,
12 # where the "x" is, we can use anything we like and it
13 # will hold the value on each iteration of the loop
14 for x in my_class:
15     print(x)
16 print()
17 # find out how long the list is (you often need to do this)
18 class_length = len(my_class)
19 output = "There are {} students in the class\n".format(class_length)
20 print(output)
21 # sort the list
22 my_class.sort()
23 print(my_class)
24 print()
25 # shuffle the list
26 # for this we need the random module(should be up the top)
27 import random
28 random.shuffle(my_class)
```

```

29 print(my_class)
30 print()
31 # add a new name to a list
32 my_class.append("Phoebe")
33 print(my_class)
34 print()
35 # check if someone is in the list
36 check_name = "Jane"
37 if check_name in my_class:
38     print("{} is in the class".format(check_name))
39 else:
40     print("{} is not in the class".format(check_name))
41 print()
42 # find the index number of a particular name (where is Harriet??)
43 index_num = my_class.index("Harriet")
44 print("{} is at index number {}".format("Harriet" , index_num))
45 print()
46 # remove someone from the list
47 my_class.remove("Harriet")
48 print(my_class)
49 print("{} is no longer in the list".format("Harriet"))

```

Listing 20: Python example

See also:

- Python Lists: [https://www.w3schools.com/python/python\\_lists.asp](https://www.w3schools.com/python/python_lists.asp)
- Python List Methods: [https://www.w3schools.com/python/python\\_lists\\_methods.asp](https://www.w3schools.com/python/python_lists_methods.asp)

## 14 Lists and Loops

### 14.1 Creating and modifying a class group

#### Scenario:

Create a simple program that asks the user to enter the names for students in a class.

Once the names have been entered it prints out the names as a numbered list.

#### Extension:

Once the names have been entered and listed, ask the user if they would like to modify or remove any of the names. (and if they do allow them to do it)

## 14.2 Lucky Unicorn Game

### 14.2.1 Program

#### Scenario:

You have decided to create a fun game to raise money for the charity Doctors without Borders.  
You will set up your computer at lunchtime and players will pay to play.

#### Here are the rules:

Users pay an initial amount at the start of the game.

- The cost should be \$1 per round and users should press <enter> to play.
  - The computer should then generate a token that is either a zebra, horse, donkey or unicorn.
  - This should be displayed to the user (as text).
  - If the token is a unicorn, the user wins \$5, if it is a zebra or horse, they win 50c and if it is a donkey then they don't win anything.
- 

The maximum amount of money that students can spend on the game is \$10 per session.

- The game should allow players continue / quit provided they have not lost all of their money.
  - It should supply appropriate feedback so that the user knows how much money they have won / lost each round and how much money they have left.
- 

Once students have no more money, the game should end (although players do have the option of quitting while they are ahead).

---

Variations (optional) Once you have a working game, you are welcome to develop the outcome further.

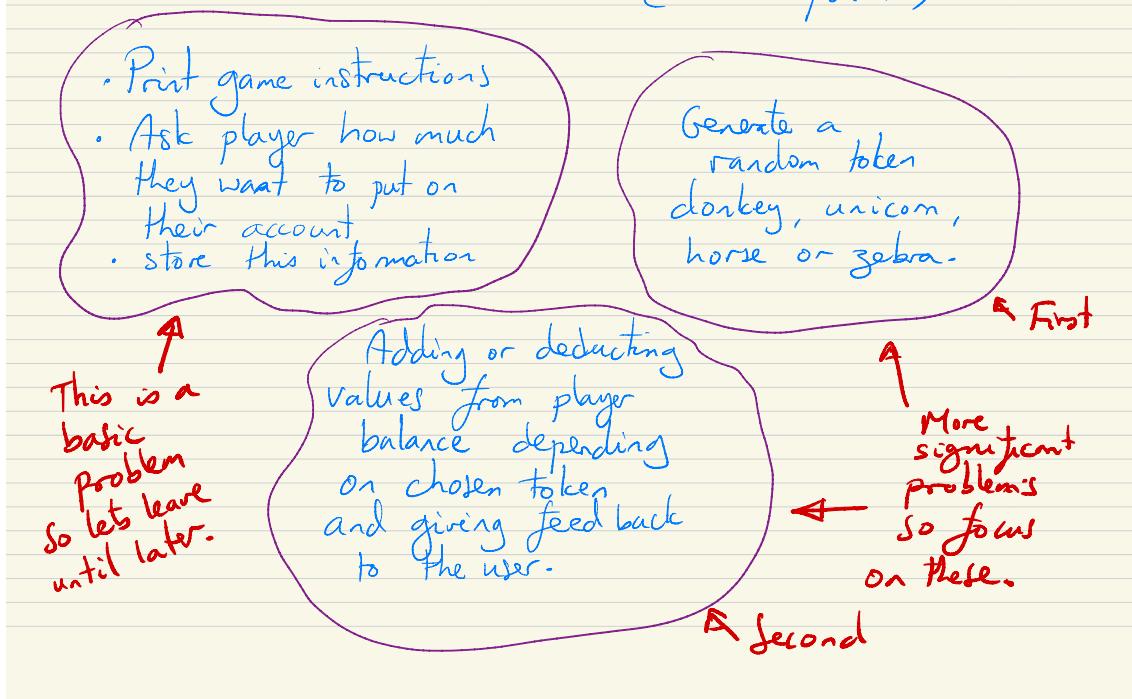
Here are some variations you could consider...

- Change the tokens / context ( so rather than having zebra, horses, donkeys and unicorns the game could involve other items )
- Allow users to bet more than \$1 per round and adjust the pay-outs proportionally  
(be careful to set up your game so that the house has a long term advantage)

#### **14.2.2 Planning: Iterative Processes**

- Break the problem down into parts (Draw some kind of diagram)
- Each of the parts can be called a “component” of the program or a “sub-problem”.
- You need plan and test code for each of these “sub-problems”.
  - Ideally (and this must happen somewhere), you need to look at “tralling” different ways to solve the sub-problem), with the idea being that you can trial the different ways and choose the best (and also look to ways of improving).  
This needs to be captured in the planning documentation for the iterative processes assessment.
- Combine your code into a fully working program (your first version)
  - Test the program and document the test (gridded test plan)
  - Write a short reflection on what works and what could be improved upon.
- Plan the next version
  - Test the program and document the test (gridded test plan)
  - Write a short reflection on what works and what could be improved upon.
- Repeat this planning and testing process.
- If new sub-problems appear , then work on them in the same way as referred to above

## Lucky Unicorn: Basic sub-problems (aka components)



Breaking the problem down into components

**Lucky Unicorn Subproblem:**

**First**

- How to randomly select a token: "Unicorn", "Horse", "Zebra", "Donkey".

(TRIAL DIFFERENT WAYS TO SOLVE THE SAME SUB PROBLEM)

- ① Put the tokens in a list ["unicorn", ...]
- ② Use a list and use randint
- ③ Use random. random(). This will give a number (decimal) between 0 and one.
- ④ Use 2 and have

```

use the random module to select the token.
token_list = ["Unicorn", "Zebra", "Horse", "Donkey"]
chosen_token = random.choice(token_list)
print(chosen_token)

random_index = random.randint(0,3)
chosen_token = token_list[random_index]
print(chosen_token)

if random_index == 0:
    chosen_token = "unicorn"
elif random_index == 1:
    chosen_token = "horse"
elif random_index == 2:
    chosen_token = "zebra"
else:
    chosen_token = "donkey"
  
```

Planning options for the first sub-problem

## Component (aka sub-problem) trials for randomly selecting a token. With outputs to test that they are working.

```
# Trial idea 1
import random
token_list = ["unicorn", "horse", "zebra", "donkey"]
chosen_token = random.choice(token_list)
print(chosen_token)

#loop test
# to make sure all the tokens are getting chosen ,
#|roughly equally
count = 0
while count < 100:
    chosen_token = random.choice(token_list)
    print(chosen_token)
    count += 1

Ln: 9 Col: 2
```

```
zebra
unicorn
donkey
donkey
donkey
unicorn
zebra
donkey
donkey
donkey
donkey
unicorn
```

```
#Trial idea 2
import random
token_list = ["unicorn", "horse", "zebra", "donkey"]
random_index = random.randint(0,3)
chosen_token = token_list[random_index]
print(chosen_token)

#loop test
# to make sure all the tokens are getting chosen ,
#|roughly equally
count = 0
while count < 100:
    random_index = random.randint(0,3)
    chosen_token = token_list[random_index]
    print(chosen_token)
    count += 1

Ln: 7 Col: 0
```

```
donkey
zebra
unicorn
donkey
donkey
horse
donkey
donkey
horse
horse
horse
horse
```

```
#Trial idea 3
import random
chosen_token = ""
random_num = random.random()
if random_num < 0.25:
    chosen_token = "Unicorn"
elif random_num < 0.5:
    chosen_token = "Horse"
elif random_num < 0.75:
    chosen_token = "Zebra"
else:
    chosen_token = "Donkey"

print(chosen_token)

Ln: 17 Col: 0
```

```
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_act
ests/get_token_3.py
Unicorn
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_act
ests/get_token_3.py
Horse
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_act
ests/get_token_3.py
Horse
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_act
ests/get_token_3.py
Donkey
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_act
ests/get_token_3.py
Donkey
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_act
ests/get_token_3.py
Donkey
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_act
ests/get_token_3.py
Zebra
>>>
```

```
#Trial idea 4
import random
chosen_token = ""
random_num = random.randint(0,3)
if random_num == 0:
    chosen_token = "Unicorn"
elif random_num == 1:
    chosen_token = "Horse"
elif random_num == 2:
    chosen_token = "Zebra"
else:
    chosen_token = "Donkey"

print(chosen_token)

Ln: 15 Col: 4
```

```
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_act
ests/get_token_4.py
Donkey
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_act
ests/get_token_4.py
Zebra
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_act
ests/get_token_4.py
Zebra
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_act
ests/get_token_4.py
Donkey
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_act
ests/get_token_4.py
Unicorn
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_act
ests/get_token_4.py
Horse
>>>
```

Trialling different ways to solve the same sub-problem

## Reflection on the trials.

- ① + The shortest code. - Don't know which index gets chosen each time.  
(but could possibly sort out)
- ② + clearly "see" which index number is selected and then can access the list using this - A bit longer than ①
- ③ + Although it is much longer I can adjust the intervals so the probabilities don't have to be equal for each token - Much longer. Difficult to add or remove tokens.
- ④ ? - Longer, difficult to add or remove tokens.

An going to choose ②.

## A reflection on the trials and a choice

Subproblem.

Second

Check what the token is and add or subtract money from user.

Program needs:

total-money of user.  
to deduct \$1 for each play.  
to check the result and then add/subtract money.

```
① total-money = 10
chosen-token = "Unicorn"
total-money -= 1
if chosen-token == "Unicorn" :
    total-money += 5
elif chosen-token == "Zebra":
    total-money += 0.5
elif chosen-token == "Horse":
    total-money += 0.8
else:
    total-money += 0
print("You have a {}".format(chosen-token))
print("You now have ${}: {:.2f}.".format(total-money))
```

② We could use a parallel list.

```
tokens = ["Unic.", "Zeb.", "Hor.", "Dont"]
values = [5, 0.5, 0.5, 0]
token_index = random.randint(0, 3)
```

```
total-money -= 1
total-money += values[token_index]
print("You have ...")
print("You now have ...").
```

## Planning options for the second sub-problem

## Component (aka sub-problem) trials for adding or deducting player money and feedback to player.

```
● ○ ● add_subtract_feed_back.py - /Users/khouripa/Documents/DTech/year11_2020/code_activities/luckyunicorn/sub...
total_money = 10
chosen_token = "Zebra"

total_money -= 1
if chosen_token == "Unicorn":
    total_money += 5
elif chosen_token == "Horse":
    total_money += 0.5
elif chosen_token == "Zebra":
    total_money -= 0.5
elif chosen_token == "Donkey":
    total_money += 0

print("You got a {}".format(chosen_token))
print("You now have ${:.2f}".format(total_money))

Ln: 2 Col: 21
```

```
● ○ ● add_subtract_feed_back_1.py - /Users/khouripa/Documents/DTech/year11_2020/code_activities/luckyunicorn/su...
import random
total_money = 10
token_list = ["Unicorn", "horse", "zebra", "donkey"]
values = [5, 0.5, 0.5, 0]
token_index = 3

total_money -= 1
total_money += values[token_index]

print("You got a {}".format(token_list[token_index]))
print("You now have ${:.2f}".format(total_money))

Ln: 5 Col: 15
```

```
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/luckyuni
ests/add_subtract_feed_back.py
You got a Unicorn
You now have $14.00
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/luckyuni
ests/add_subtract_feed_back.py
You got a Donkey
You now have $9.00
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/luckyuni
ests/add_subtract_feed_back.py
You got a Horse
You now have $9.50
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/luckyuni
ests/add_subtract_feed_back.py
You got a Zebra
You now have $9.50
>>>
```

↓ Improvement after the first trial

```
● ○ ● add_subtract_feed_back-a.py - /Users/khouripa/Documents/DTech/year11_2020/code_activities/luckyunicorn/sub...
# after the first trial of this idea
# I modified it to simplify the
# conditions so it is now much more efficient

total_money = 10
chosen_token = "Unicorn"

total_money -= 1
if chosen_token == "Unicorn":
    total_money += 5
elif chosen_token == "Horse" or chosen_token == "Zebra":
    total_money -= 0.5

print("You got a {}".format(chosen_token))
print("You now have ${:.2f}".format(total_money))

Ln: 6 Col: 23
```

```
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/luci
ests/add_subtract_feed_back-a.py
You got a Zebra
You now have $9.50
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/luci
ests/add_subtract_feed_back-a.py
You got a Horse
You now have $9.50
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/luci
ests/add_subtract_feed_back-a.py
You got a Donkey
You now have $9.00
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/luci
ests/add_subtract_feed_back-a.py
You got a Unicorn
You now have $14.00
>>>|
```

Trialling different ways to solve the same sub-problem (the second)

## Reflection on Trials for adding and subtracting player amount and feedback.

### ① Using of chf.

A standard solution  
easy to understand  
works fine.

Is still a bit "cumbersome".  
could be a bit laborious  
to add on new tokens or  
to update values.

→ The second trial was able to improve  
the efficiency of the code  
and make it clever. There was no need to  
include the donkey as not no money is added.

### ②

Very simple and  
is easy to add new tokens  
and change token values.

None at this stage  
(Possibly could be improved further,  
but okay for now).

A reflection on the trials and a choice

```

1 import random
2 # print output for start of Lucky Unicorn Game
3 print("----- Welcome to the Lucky Unicorn Game -----")
4 print("Enter between $1 and $10 to play")
5 print("It costs $1 per round")
6 print("If you get a unicorn you will get $5 back")
7 print("If it is a Zebra or Horse you get 50c back")
8 print("A donkey gives nothing back")
9 print("----- Start -----")
10 print()
11 # get amount of money to play with
12 tokens = ["horse", "zebra", "donkey", "unicorn"]
13 player_amount=int(input("Please enter a value between 1 and 10: -> "))
14 print("You have entered ${:.2f}".format(player_amount))
15 #game play
16 play=""
17 while play=="":
18     player_token = random.choice(tokens)
19     player_amount -= 1
20     if player_token == "unicorn":
21         player_amount += 5
22         print("Congratulations, you had a {}".format(player_token))
23     elif player_token == "horse" or player_token == "zebra":
24         player_amount +=0.5
25         print("You had a {}".format(player_token))
26     else:
27         print("Unfortunately you had a {}".format(player_token))
28     print("Your balance is now ${:.2f}".format(player_amount))
29     play=input("Press <Enter> to continue or quit(q)")
```

Listing 21: Python example

# 15 Code Examples

## 15.1 Dinner Order

```
1 #constants
2 MAIN_PRICE = 12.5
3 DESSERT_PRICE = 6.0
4 DRINKS_PRICE = 3.55
5
6 #totalling variables (initially declared as 0)
7 mains_total = 0.0
8 desserts_total = 0.0
9 drinks_total = 0.0
10 ext_gst = 0.0
11
12 # input variables
13 mains = int(input("How many mains would you like to order?: "))
14 desserts = int(input("How many desserts would you like to order?: "))
15 drinks = int(input("How many drinks would you like to order?: "))
16
17 #calulate costs
18 mains_total = mains * MAIN_PRICE
19 desserts_total = desserts * DESSERT_PRICE
20 drinks_total = drinks * DRINKS_PRICE
21
22 ext_gst = mains_total + desserts_total + drinks_total
23 my_gst = ext_gst*0.15
24 my_to_pay = ext_gst + my_gst
25
26 print("-"*40)
27 print("Bill's Diner")
28 print("-"*40)
29 print("{:<3}{:10}at ${:<10.2f}= ${:.2f}".format(mains,"mains", MAIN_PRICE,
   mains_total))
30 print("{:<3}{:10}at ${:<10.2f}= ${:.2f}".format(desserts,"desserts",
   DESSERT_PRICE, desserts_total))
31 print("{:<3}{:10}at ${:<10.2f}= ${:.2f}".format(drinks,"drinks", DRINKS_PRICE,
   drinks_total))
32 print("-"*40)
33 print("{:>27}= ${}{}".format("Total ", ext_gst))
34 print("{:>27}= ${:.2f}{}".format("GST ", my_gst))
35 print("{:>27}= ${:.2f}{}".format("To Pay ", my_to_pay))
36 print("-"*40)
37 print("Thankyou")
38 print("-"*40)
```

Listing 22: Dinner Order Program

## 15.2 Strawberry Picking

```
1 # get number of punnets boxes buckets
2 my_punnets = float(input("How many punnets?: "))
3 my_boxes = float(input("How many boxes?: "))
4 my_buckets = float(input("How many buckets?: "))
5
6 #testing
7 #my_punnets = 2.5
8 #my_boxes = 1.75
9 #my_buckets = 3.5
10
11 punnet_weight = my_punnets*0.25
12 box_weight = my_boxes*1.5
13 bucket_weight = my_buckets*7
14
15 total_weight = punnet_weight + box_weight + bucket_weight
16
17 total_price = total_weight * 5
18 print("."*30)
19 print("{:20}{:>6.2f} kg".format("Punnet weight" , punnet_weight))
20 print("{:20}{:>6.2f} kg".format("Box weight" , box_weight))
21 print("{:20}{:>6.2f} kg".format("Bucket weight", bucket_weight))
22 print("."*30)
23 print("{:20}{:>6.2f} kg".format("Total weight", total_weight))
24 print("."*30)
25 print("."*30)
26 print("{:20} ${:>6.2f}".format("Price to charge", total_price))
27 print("."*30)
```

Listing 23: Python example

## 15.3 Guess greater, less or equal

```
1 import random
2 lower = 4
3 higher = 15
4 my_random = random.randint(lower,higher)
5 middle_number = round((lower + higher)/2)
6 #print(middle_number)
7 #print(my_random)
8
9 print("A random number has been chosen between {} and {}".format(lower, higher))
10 print(40*"-")
11 guess = input("Enter L if you think the number is less than {0} \n"
12             "Enter G if you think the number is more than {0} \n"
13             "Enter E if you think the number is equal to {0} : ".format(
    middle_number))
```

```

14
15 if guess == "L" and my_random < middle_number:
16     print("you win")
17 elif guess == "G" and my_random > middle_number:
18     print("you win")
19 elif guess == "E" and my_random == middle_number:
20     print("you win")
21 elif guess != "G" and guess != "L" and guess != "E":
22     print("you have not entered an appropriate letter")
23 else:
24     print("you lose")
25 print("The number was {}".format(my_random))

```

Listing 24: Python example

## 15.4 Higher Lower Game

```

1 import random
2
3 mainLoop="y"
4 while mainLoop=="y":
5     #declare variables
6     name=""
7     secretNumber=0
8     guess=0
9     count=0
10    #get user input and validate
11    getUserNameLoop="y"
12    while getUserNameLoop=="y":
13        name=input("Please enter your name    ")
14        if len(name)<2 or len(name)>30 or name.isalpha()!=True:
15            print("Please enter a valid name")
16            continue
17        else:
18            #name has been captured correctly
19            getUserNameLoop="n"
20    # welcome
21    print("Hello {}, you are about to play a guessing game".format(name))
22    print("where you need to guess the secret number between 1 and 50")
23    secretNumber=random.randint(1,50)
24    #testing help
25    print("Testing help: the secret number is {}".format(secretNumber))
26    ready=input("Press enter to start    ")
27    #start game
28    gameLoop="y"
29    while gameLoop=="y":
30        # get guess and validate
31        getGuessInputLoop="y"

```

```

32     while getGuessInputLoop=="y":
33         try:
34             guess=int(input("Please guess the secret number between 1 and 50
"))
35         except ValueError:
36             print("Please enter a valid integer")
37             continue
38         if guess<1 or guess>50:
39             print("Please enter a number between 1 and 50")
40         else:
41             # integer correctly entered
42             getGuessInputLoop="n"
43 #guess count
44 count=count+1
45 #respond to input
46 if guess<secretNumber:
47     print("Your guess is too small")
48 elif guess>secretNumber:
49     print("Your guess is too large")
50 else:
51     #number is found, end loop and print responses
52     print("You have found it!!")
53     gameLoop="n"
54     print("You took {} tries to find it".format(count))
55     if count<5:
56         print("You are really fast")
57     elif count<7:
58         print("Pretty good")
59     else:
60         print("You can improve, I think")
61 #exited game loop
62 mainLoop=input("Do you want to play again? y/n      ")
63 #exited main loop
64 print("The program has ended")

```

Listing 25: Python example

## 16 Testing

## 17 Dictionaries

```
1 car = {  
2     "brand": "Ford",  
3     "model": "Mustang",  
4     "year": 1964  
5 }  
6  
7 x = car.setdefault("model", "Bronco")  
8  
9 print(x)  
10  
11 quiz_one = {  
12     "question": "What is the capital of France? ",  
13     "answer": "Paris"  
14 }  
15 quiz_two = {  
16     "question": "What is the capital of Spain? ",  
17     "answer": "Madrid"  
18 }  
19 quiz_three = {  
20     "question": "What is the capital of Germany? ",  
21     "answer": "Berlin"  
22 }  
23 print(quiz_one.get("question"))  
24  
25 user_answer = input(quiz_one.get("question"))  
26 if user_answer == quiz_one.get("answer"):  
27     print("Correct")  
28 user_answer = input(quiz_two.get("question"))  
29 if user_answer == quiz_two.get("answer"):  
30     print("Correct")  
31 user_answer = input(quiz_three.get("question"))  
32 if user_answer == quiz_three.get("answer"):  
33     print("Correct")  
34  
35 quiz_set = [  
36     {  
37         "question": "What is the capital of France? ",  
38         "answer": "Paris"  
39     },  
40     {  
41         "question": "What is the capital of Spain? ",  
42         "answer": "Madrid"  
43     },  
44     {  
45         "question": "What is the capital of Germany? ",
```

```

46     "answer" : "Berlin"
47 }
48 ]
49
50 for x in quiz_set:
51     user_answer = input( x.get("question") )
52     if user_answer == x.get("answer"):
53         print("Correct")

```

Listing 26: Python example

```

1 question_one = {
2     "question": "What is the capital of France?",
3     "A": "Paris",
4     "B": "Moscow",
5     "C": "Shanghai",
6     "answer": "C"
7 }
8 print(question_one["A"])
9 print(question_one["B"])
10 print("-"*30)
11
12 for x,y in question_one.items():
13     if x in ["A", "B", "C"]:
14         print("{} , {}".format(x,y))

```

Listing 27: Python example

## 18 Coffee

```
1 import random
2
3 char_list = ["C", "O", "F", "E"]
4
5
6
7 def run_trial():
8     counter = 1
9     C_count = 0
10    O_count = 0
11    F_count = 0
12    E_count = 0
13    char_string = ""
14    while True:
15        my_char = random.choice(char_list)
16        char_string += my_char
17        # count number of C, O,
18        C_count = char_string.count("C")
19        O_count = char_string.count("O")
20        F_count = char_string.count("F")
21        E_count = char_string.count("E")
22        if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
23            break
24        # increment count
25        counter += 1
26        #print(char_string)
27    return len(char_string)
28
29
30 total_trials = 100
31 trials = 1
32 total_chars = 0
33 while trials <= total_trials:
34     total_chars += run_trial()
35     trials += 1
36
37
38 print(total_chars)
39 print(total_trials)
40
41 my_average = total_chars/total_trials
42 print(my_average)
```

Listing 28: Python example

```

1 import random
2
3 my_list=["c", "o", "f", "e"]
4
5 my_string = ""
6 count = 1
7 while count<100:
8     my_letter = random.choice(my_list)
9     my_string += my_letter
10    #print(my_letter)
11    if my_string.count("c")>=1 and my_string.count("o")>=1 \
12        and my_string.count("f")>=2 and my_string.count("e")>=2:
13        break
14    count += 1
15 for x in my_list:
16     print("{} * {}".format(x,my_string.count(x)))
17 print(my_string)
18 print("count is {}".format(count))

```

Listing 29: Python example

```

1 import random
2
3 my_list=["c", "o", "f", "e"]
4
5 my_string = ""
6 count = 1
7 while count<100:
8     my_letter = random.choice(my_list)
9     my_string += my_letter
10    #print(my_letter)
11    if my_string.count("c")>=1 and my_string.count("o")>=1 \
12        and my_string.count("f")>=2 and my_string.count("e")>=2:
13        break
14    count += 1
15 for x in my_list:
16     print("{} * {}".format(x,my_string.count(x)))
17 print(my_string)
18 print("count is {}".format(count))

```

Listing 30: Python example

```

1 import random
2
3 my_list=["c", "o", "f", "e"]
4
5 string_list = []
6

```

```

7 counter = 0
8 while counter < 1000:
9     my_string = ""
10    count = 1
11    while True:
12        my_letter = random.choice(my_list)
13        my_string += my_letter
14        #print(my_letter)
15        if my_string.count("c")>=1 and my_string.count("o")>=1 \
16            and my_string.count("f")>=2 and my_string.count("e")>=2:
17            break
18        count += 1
19        string_list.append(my_string)
20        counter += 1
21        # print(my_string)
22 #print(string_list)
23 string_list.sort(key=len)
24 print(string_list[0])
25 print(string_list[- 1])
26
27 total_times = 0
28 number_trials = 0
29 for x in string_list:
30     total_times += len(x)
31     number_trials += 1
32 print(total_times)
33 print(number_trials)

```

Listing 31: Python example

```

1 import random
2
3 char_list = ["C", "O", "F", "E"]
4
5
6 chars_in_six = 0
7
8 C_count = 0
9 O_count = 0
10 F_count = 0
11 E_count = 0
12
13
14 trials = 0
15 trial_total = 100000
16 while trials < trial_total:
17     counter = 0
18     char_string = ""

```

```

19     while counter < 6:
20         my_char = random.choice(char_list)
21         char_string += my_char
22         # count number of C, O,
23         C_count = char_string.count("C")
24         O_count = char_string.count("O")
25         F_count = char_string.count("F")
26         E_count = char_string.count("E")
27         if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
28             chars_in_six +=1
29             # increment count
30             counter += 1
31         #print(char_string)
32         trials += 1
33
34 print(chars_in_six/trial_total)
35 #print(char_string.count("C"))

```

Listing 32: Python example

```

1 import random
2
3 char_list = ["C", "O", "F", "E"]
4
5 # empty list
6 string_list = []
7
8
9 #-----
10 def make_coffee():
11     C_count = 0
12     O_count = 0
13     F_count = 0
14     E_count = 0
15
16     char_string = ""
17     while True:
18         my_char = random.choice(char_list)
19         char_string += my_char
20         # count number of C, O,
21         C_count = char_string.count("C")
22         O_count = char_string.count("O")
23         F_count = char_string.count("F")
24         E_count = char_string.count("E")
25         if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
26             string_list.append(char_string)
27             break
28             # increment count

```

```

29
30
31     #print(char_string)
32     #print( len(char_string) )
33     return len(char_string)
34 #-----
35
36
37 total_trials = 1000000
38 trials = 1
39
40 total_string_lengths = 0
41
42 while trials <= total_trials:
43     my_number = make_coffee()
44     total_string_lengths += my_number
45     #print(my_number)
46     trials +=1
47 #print(total_string_lengths)
48 average = round(total_string_lengths / total_trials)
49 print("On average you will need {} visits".format(average))
50 #print(string_list)
51
52 string_list.sort(key=len)
53 #print(string_list)
54 #print first item (thing)
55 print(string_list[0])
56 #print last item
57 print(string_list[total_trials-1])
58 print( len( string_list[total_trials-1] ) )
59
60
61
62
63
64
65
66
67
68
69 #print(string_list)
70
71
72
73 #string_list.sort(key = len)

```

Listing 33: Python example

```

1 import random
2
3 char_list = ["C", "O", "F", "E"]
4
5 string_list = []
6
7
8 def run_trial():
9     counter = 1
10    C_count = 0
11    O_count = 0
12    F_count = 0
13    E_count = 0
14    char_string = ""
15    while True:
16        my_char = random.choice(char_list)
17        char_string += my_char
18        # count number of C, O,
19        C_count = char_string.count("C")
20        O_count = char_string.count("O")
21        F_count = char_string.count("F")
22        E_count = char_string.count("E")
23        if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
24            string_list.append(char_string)
25            break
26        # increment count
27        counter += 1
28    #print(char_string)
29    return len(char_string)
30
31
32
33
34 total_trials = 100
35 trials = 1
36 total_chars = 0
37 while trials <= total_trials:
38     total_chars += run_trial()
39     trials += 1
40
41
42 print(total_chars)
43 print(total_trials)
44 my_average = total_chars/total_trials
45 print(my_average)
46
47 #print(string_list)

```

```

48 string_list.sort(key = len)
49 #print(string_list)
50 print(string_list[0])
51 print(string_list[-1])
52 print(len(string_list[0]))
53 print(len(string_list[-1]))
54
55 my_dict = {}
56 for i in range(len(string_list[0]), len(string_list[-1])+1):
57     my_dict[i]=0
58     #print(i)
59
60 #print(my_dict)
61 for x in string_list:
62     my_dict[len(x)]+=1
63     #print(x)
64 print(my_dict)
65 my_sum = 0
66 for x in my_dict:
67     my_sum += my_dict[x]
68 print(my_sum)
69 # dictionary not the same lenght as string list
70 for x,y in my_dict.items():
71     temp = int(y)
72     temp = temp/total_trials
73     print(temp)
74     my_dict[x] = temp
75 print(my_dict)

```

Listing 34: Python example

```

1 import random
2
3 char_list = ["C", "O", "F", "E"]
4
5
6 chars_in_seven = 0
7
8 C_count = 0
9 O_count = 0
10 F_count = 0
11 E_count = 0
12
13
14 trials = 0
15 trial_total = 1000000
16 while trials < trial_total:
17     counter = 1

```

```

18     char_string = ""
19     while counter <= 7:
20         my_char = random.choice(char_list)
21         char_string += my_char
22         # count number of C, O, F, E
23         C_count = char_string.count("C")
24         O_count = char_string.count("O")
25         F_count = char_string.count("F")
26         E_count = char_string.count("E")
27
28         # increment count
29         counter += 1
30     # test string at end
31     if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
32         chars_in_seven +=1
33     #print(char_string)
34     trials += 1
35
36 print(chars_in_seven/trial_total)
37 #print(char_string.count("C"))

```

Listing 35: Python example

```

1 import random
2
3 char_list = ["C", "O", "F", "E"]
4
5
6 chars_in_eight = 0
7
8 C_count = 0
9 O_count = 0
10 F_count = 0
11 E_count = 0
12
13
14 trials = 0
15 trial_total = 1000000
16 while trials < trial_total:
17     counter = 1
18     char_string = ""
19     while counter <= 8:
20         my_char = random.choice(char_list)
21         char_string += my_char
22         # count number of C, O, F, E
23         C_count = char_string.count("C")
24         O_count = char_string.count("O")
25         F_count = char_string.count("F")

```

```

26     E_count = char_string.count("E")
27
28     # increment count
29     counter += 1
30
31     # test string at end
32     if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
33         chars_in_eight +=1
34     #print(char_string)
35
36 trials += 1
37
38 print(chars_in_eight/trial_total)
39 #print(char_string.count("C"))

```

Listing 36: Python example

```

1 import random
2
3 char_list = ["C", "O", "F", "E"]
4
5
6
7 C_count = 0
8 O_count = 0
9 F_count = 0
10 E_count = 0
11
12 char_string = ""
13 while True:
14     my_char = random.choice(char_list)
15     char_string += my_char
16     # count number of C, O,
17     C_count = char_string.count("C")
18     O_count = char_string.count("O")
19     F_count = char_string.count("F")
20     E_count = char_string.count("E")
21     if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
22         break
23     # increment count
24
25
26 print(char_string)
27 print( len(char_string) )
28
29 #print(char_string.count("C"))

```

Listing 37: Python example

```
1 import random
```

```

2
3 char_list = ["C", "O", "F", "E"]
4
5
6
7
8 #-----
9 def make_coffee():
10     C_count = 0
11     O_count = 0
12     F_count = 0
13     E_count = 0
14
15     char_string = ""
16     while True:
17         my_char = random.choice(char_list)
18         char_string += my_char
19         # count number of C, O,
20         C_count = char_string.count("C")
21         O_count = char_string.count("O")
22         F_count = char_string.count("F")
23         E_count = char_string.count("E")
24         if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
25             break
26         # increment count
27
28
29     #print(char_string)
30     #print( len(char_string) )
31     return len(char_string)
32 #-----
33
34
35 total_trials = 100
36 trials = 1
37
38 total_string_lengths = 0
39
40 while trials <= total_trials:
41     my_number = make_coffee()
42     total_string_lengths += my_number
43     #print(my_number)
44     trials +=1
45 #print(total_string_lengths)
46 average = round(total_string_lengths / total_trials)

```

```
47 print("On average you will need {} visits".format(average))
```

Listing 38: Python example

```
1 import random
2
3 char_list = ["C", "O", "F", "E"]
4
5 string_list = []
6
7
8 def run_trial():
9     counter = 1
10    C_count = 0
11    O_count = 0
12    F_count = 0
13    E_count = 0
14    char_string = ""
15    while True:
16        my_char = random.choice(char_list)
17        char_string += my_char
18        # count number of C, O,
19        C_count = char_string.count("C")
20        O_count = char_string.count("O")
21        F_count = char_string.count("F")
22        E_count = char_string.count("E")
23        if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
24            string_list.append(char_string)
25            break
26        # increment count
27        counter += 1
28    #print(char_string)
29    return len(char_string)
30
31
32
33
34 total_trials = 1000000
35 trials = 1
36 total_chars = 0
37 while trials <= total_trials:
38     total_chars += run_trial()
39     trials += 1
40
41
42 print(total_chars)
43 print(total_trials)
44 my_average = total_chars/total_trials
```

```

45 print(my_average)
46
47 #print(string_list)
48 string_list.sort(key = len)
49 #print(string_list)
50 print(string_list[0])
51 print(string_list[-1])
52 print(len(string_list[0]))
53 print(len(string_list[-1]))
54
55 my_dict = {}
56 for i in range(len(string_list[0]), len(string_list[-1])+1):
57     my_dict[i]=0
58     #print(i)
59
60 #print(my_dict)
61 for x in string_list:
62     my_dict[len(x)]+=1
63
64 print(my_dict)
65 my_sum = 0
66 for x in my_dict:
67     my_sum += my_dict[x]
68 print(my_sum)
69 # dictionary not the same lenght as string list
70 for x,y in my_dict.items():
71     temp = int(y)
72     temp = temp/total_trials
73     my_dict[x]= temp
74 print(my_dict)
75
76 my_file = open("distribution_next.txt","a")
77 for x,y in my_dict.items():
78     my_string = "{},{:.6f}\n".format(x,y)
79     my_file.write(my_string)
80
81 #my_file.write("hello")
82 #my_file.write("\n")
83 #my_file.write("goodbye")
84 my_file.close()
85 #my_file = open("MyFile.txt","w")
86 #my_file.write("")
87 #my_file.close()

```

Listing 39: Python example

## 19 Tests

```
1 # roll a dice
2 import random
3 two_d_list = []
4
5
6
7 def searchfor(n):
8     if len(two_d_list) == 0:
9         two_d_list.append([n, 1])
10        return
11    added = False
12    for sublist in two_d_list:
13        if sublist[0] == n:
14            sublist[1] += 1
15            added = True
16            break
17    if added == False:
18        two_d_list.append([n, 1])
19
20 for i in range(0, 1500):
21     diceroll = random.randint(1, 6) + random.randint(1, 6) + random.randint(1, 6)
22     searchfor(diceroll)
23     #print(diceroll)
24
25
26 two_d_list.sort()
27 print(two_d_list)
```

Listing 40: Dice Roll

```
1 guess_list = [45, 78, 90, 21, 7]
2
3 guess=int(input("Enter your guess"))
4
5
6 while guess in guess_list:
7     guess = int(input("You have already tried this number,\n
8 have another guess"))
9 guess_list.append(guess)
10 print(guess_list)
```

Listing 41: Python example

```
1 import time
2 start= time.time()
3 t= time.localtime()
```

```

4 print(t)
5 print(start)
6
7
8
9
10 word = "halloween"
11 char_list = list(word)
12 print(char_list)
13 char_list_chosen=["l", "w", "e"]
14 outstring = ""
15 for x in char_list:
16     if x in char_list_chosen:
17         outstring += " "+x+" "
18     else:
19         outstring += " _ "
20 print(outstring)
21
22
23
24
25
26
27 question_one = {
28     "question": "What is the capital of France?",
29     "A": "Paris",
30     "B": "Moscow",
31     "C": "Shanghai",
32     "answer": "C"
33 }
34
35 print(question_one["question"])
36 for x,y in question_one.items():
37     if x != "question" and x != "answer":
38         print("{} {})".format(x,y))
39
40 for x,y in question_one.items():
41     if len(x)== 1:
42         print("{} {})".format(x,y))
43
44 answer = input("Enter your choice ")
45 if answer == question_one["answer"]:
46     print(" Brilliant ")
47 end=time.time()
48 print(end)
49 print(end-start)
50 count = 0

```

```

51 while count < 3:
52     temp={}
53     temp["english"] = input("Enter your english word")
54     temp["thai"] = input("Enter your thai word")
55     print(temp)
56     count +=1

```

Listing 42: Python example

```

1 my_num=0.0000067
2 print("{:.6f}".format(my_num) )

```

Listing 43: Python example

```

1 word = "halloween"
2
3 def pic(c):
4     if c >= 1:
5         print(" "*30+" - - - -")
6     if c >=2:
7         print(" "*30+" | | ")
8     if c >=3:
9         print(" "*30+" () | ")
10    if c >= 4:
11        print(" "*30+" | | ")
12    if c >= 5:
13        print(" "*30+" --- | ")
14        print(" "*30+" | | ")
15        print(" "*30+" / \ | ")
16        print(" "*30+" _ _ | ")
17 pic(5)
18 char_list = list(word)
19 char_individual_list=[]
20 char_list_chosen=[]
21
22 for x in char_list:
23     if x not in char_individual_list:
24         char_individual_list.append(x)
25
26 wrong_count = 0
27 while len(char_individual_list)>0:
28     outstring = ""
29     for x in char_list:
30         if x in char_list_chosen:
31             outstring += " "+x+" "
32         else:
33             outstring += " _ "
34     print("{}\n\n".format(outstring))

```

```

35     char = input("guess a letter: ")
36     if char in char_individual_list:
37         char_list_chosen.append(char)
38         char_individual_list.remove(char)
39     else:
40         wrong_count += 1
41         pic(wrong_count)
42     if len(char_individual_list) == 0:
43         print("you got it")
44
45
46 outstring = ""
47 for x in char_list:
48     if x in char_list_chosen:
49         outstring += " "+x+" "
50     else:
51         outstring += " _ "
52 print(outstring)

```

Listing 44: Python example

## 20 Strings

```

1 my_string="adkadkkdjk jkjdhkdh"
2 my_split=my_string.split("a")
3 print(my_split)
4 my_list=list(my_string)
5 print(my_list)

```

Listing 45: Python example

```

1 my_string = "woollooomooloo"
2
3 my_letters=[]
4 count_letters = []
5
6 for x in my_string:
7     if x not in my_letters:
8         my_letters.append(x)
9         count_letters.append(my_string.count(x))
10
11 print(my_letters)
12 print(count_letters)

```

Listing 46: Python example

## 21 Objects

```

1 class Shark:
2
3     def __init__(self, name, age):
4         self.name = name
5         self.age = age
6         """ constructor """
7         print("This is the constructor")
8
9     def swim(self):
10        """ what the shark is doing """
11        print(self.name+" is swimming")
12
13    def be_awesome(self):
14        """ what the shark is being """
15        print(self.name+" is being awesome")
16
17    def _age(self):
18        """ how old is it """
19        print(self.name + " is "+str(self.age)+" years old")
20
21
22 def main():
23     sammy = Shark("Sammy", 5)
24     stevie = Shark("Stevie", 3)
25     sammy.swim()
26     stevie.be_awesome()
27     sammy._age()
28
29 if __name__ == "__main__":
30     main()

```

Listing 47: Python example

```

1 class Store:
2
3     def __init__(self):
4         self.colour = "rgb(0,0,0,0)"
5
6
7
8
9
10 class Shark:
11
12     def __init__(self, name, age):
13         self.name = name
14         self.age = age
15         """ constructor """

```

```

16     store.colour = "rgb(255,255,255,1)"
17     print("This is the constructor")
18
19     def swim(self):
20         """ what the shark is doing"""
21         print(self.name+" is swimming")
22
23     def be_awesome(self):
24         """ what the shark is being"""
25         print(self.name+" is being awesome")
26
27     def _age(self):
28         """ how old is it """
29         print(self.name + " is "+str(self.age)+" years old")
30
31 store = Store()
32 shark = Shark("sam", 10)
33 print(store.colour)

```

Listing 48: Python example

```

1 class Question:
2
3     def __init__(self, question, answer, score):
4
5         self.question = question
6         self.answer = answer
7         self.score = score
8
9         self.user_answer = ""
10        self.user_score = 0
11        self.response_to_user = ""
12
13    def make_question(self):
14        msg = self.question
15        self.user_answer = input(msg)
16        if self.user_answer == self.answer:
17            self.response_to_user = "Correct"
18            self.user_score = self.score
19        else:
20            self.response_to_user = "Incorrect"
21
22    def feedback(self):
23        strOne = "For the question: "+self.question+"\n"
24        strTwo = "You answered: "+self.user_answer+"\n"
25        strThree = "Your answer was "+self.response_to_user
26        print(strOne + strTwo + strThree)

```

```
28 #q_one = Question("What is the capital of France?", "Paris", 10)
29 #q_one.make_question()
30 #q_one.feedback()
31 q_list = []
32 q_list.append(Question("What is the capital of France?", "Paris", 10))
33 q_list.append(Question("What is the capital of Belgium?", "Brussels", 10))
34 q_list.append(Question("What is the capital of Holland?", "Amstredam", 10))
35
36 count = 0
37 while count < len(q_list):
38     q_list[count].make_question()
39     count += 1
40
41 count = 0
42 while count < len(q_list):
43     q_list[count].feedback()
44     count += 1
```

Listing 49: Python example