Level 1
Iterative Processes Course Notes

Contents

1 Introduction **3**

2 Dinner Order - Basic Program - Design and Build **4**

2.0.1 Planning 4

2.1 Strawberry picking 10

3 Conditions - Activities **12**

4 Guess greater , less or equal. **13**

5 Loops **16**

6 Loop Activities - improving previously made programs **18**

6.1 Higher Lower Game 20

6.1.1 Program 20

6.1.2 Planning 21

7 Validation **25**

7.1 Integer Validation 26

7.2 String Validation 29

8 Lists **31**

9 Lists and Loops **32**

9.1 Creating and modifying a class group 32

9.2 Lucky Unicorn Game 33

9.2.1 Program 33

9.2.2 Planning: Iterative Processes 34

10 Functions **41**

11 Testing **42**

12 Dictionaries **43**

13 Coffee **45**

14 Tests **57**

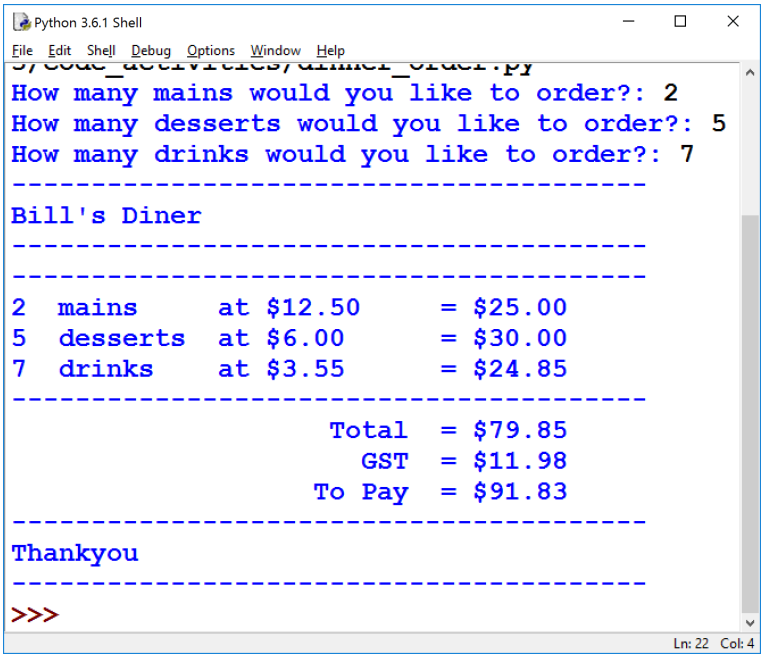
15 Strings **60**

16 Objects **60**

1 Introduction

2 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)



An example of Dinner Order program run

2.0.1 Planning

For the writing and planning:

- **Decomposition** Please write down your task decomposition.
  (This is breaking the task down into smaller pieces that will be combined to make the finished program)
  See examples below.

- **Version Log**
  Your version log should go here. Annotated screenshots are a good idea at this point

- **Component Trialling**
  Show that you have trialled each component here.
  You should also include notes that justify the major decisions you made.

- **Assembled Outcome Testing**

  Please show testing for your assembled outcome below.

  This should include a test plan followed by screenshot proof

- **Usability Testing**

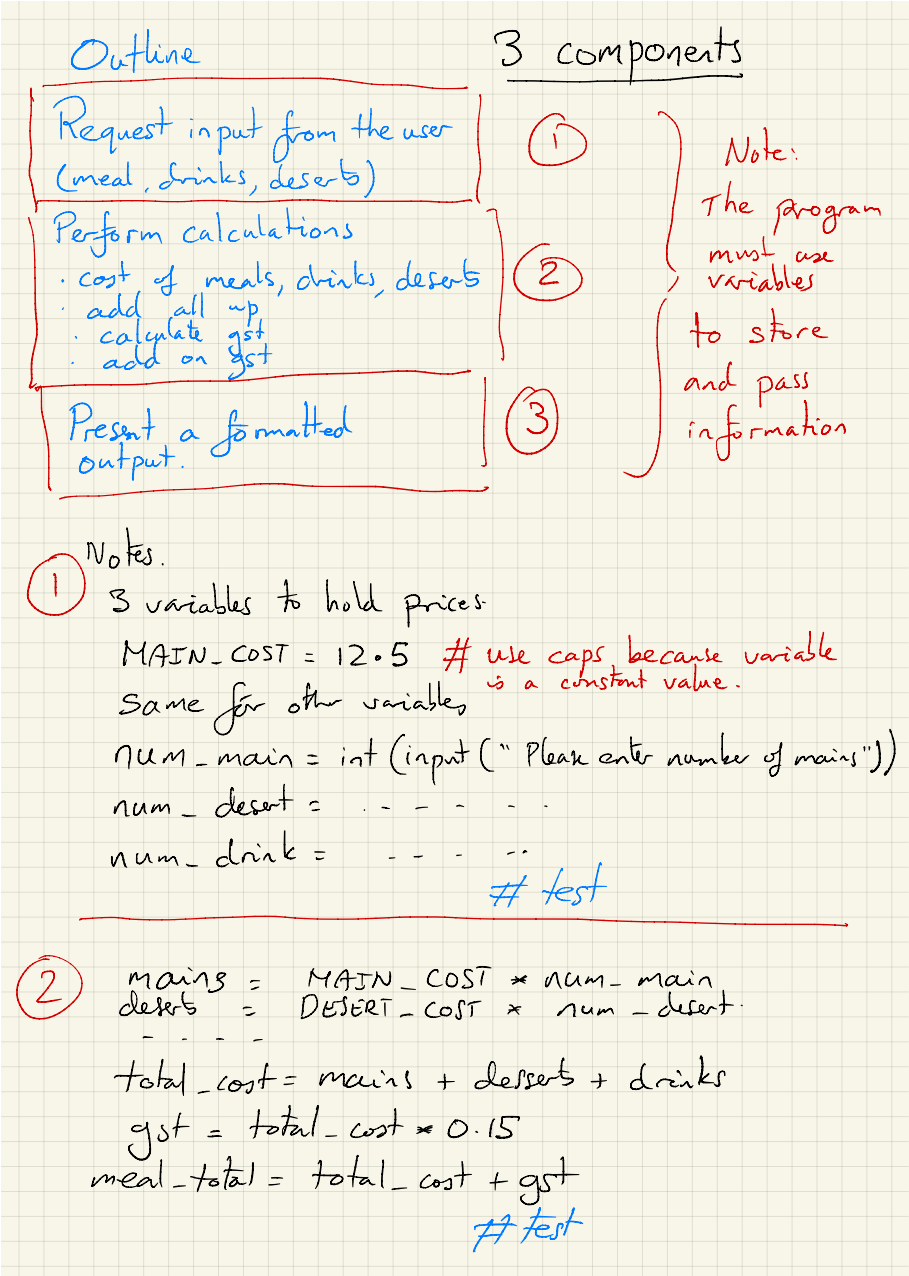
  Try the program out on someone else and make notes or video etc. Write a list of things improvements which need to be made based on your usability testing. Then write down what you changed.

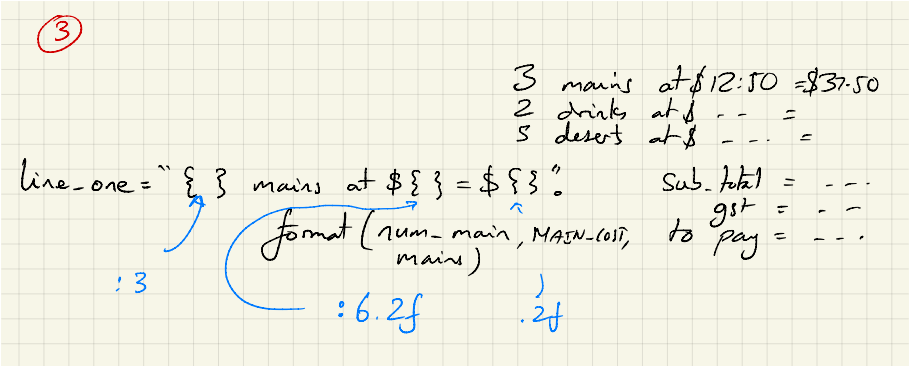
- **General Evaluation:**

  How good is may program? What can I do to improve it?

Examples of Component Planning

Basic Idea of Testing When testing we consider



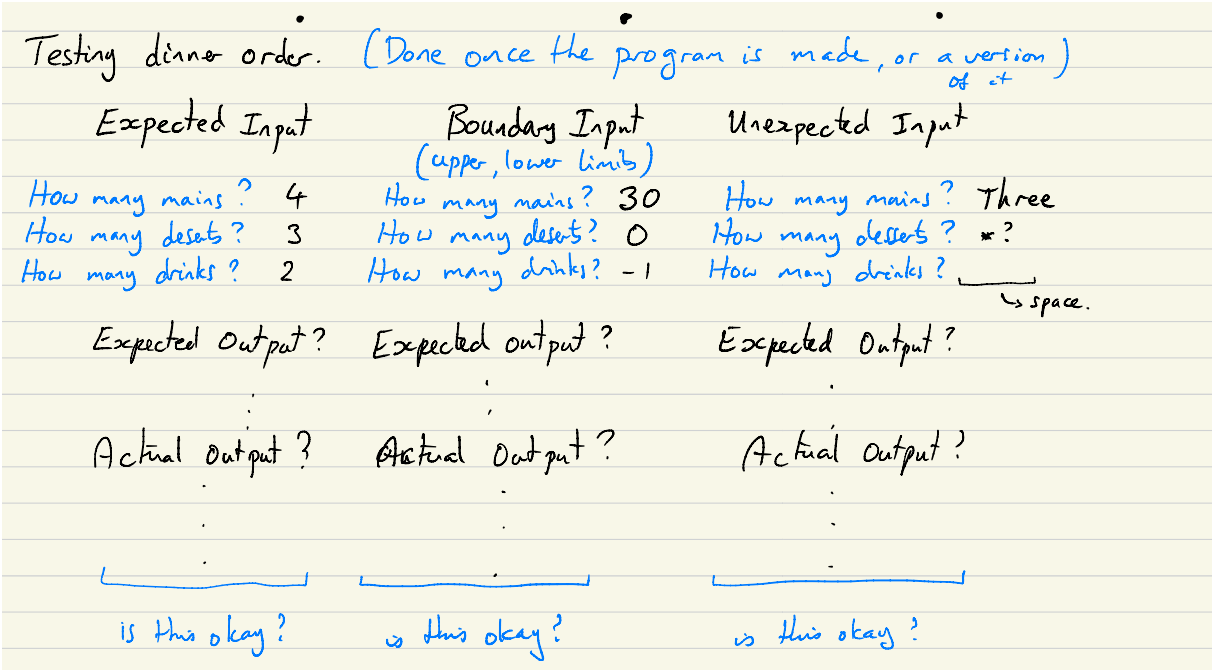


- Expected inputs (these are inputs we would normally expect the user to input).
  Does the porgram give the right outputs with the given expected inputs.

- 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.



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/
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

```
>>>
= 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 reciept 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)

Components: (parts of the program we need to be able to make).
(The components will then be assembled to make the finished program).

① Getting the user input
② Calculating the value of the meal
③ Printing out the receipt.

① (variable; polite, clear message to user.)

```
num_mains = int(input("How many mains would you like?"))
num_deserts = int(input("  "  "  desert  "  "  "))
num_drinks = int(input("  "  "  drinks  "  "  "))

print(num_mains)
print(num_deserts)
print(num_drinks)
```

← print out to test that it is working

Save with logical name: dinner_input_component

② Declare the prices as variables. UPPER CASE means will not change (constant).

```
MAIN = 12.5
DESERT = 6.0
DRINK = 3.55
```

← just give the variables values for trialling purposes.

```
num_mains = 3
num_deserts = 2
num_drinks = 4

main_cost = num_mains * 12.5
desert_cost = num_desert * 6.0
drinks_cost = num_drinks * 3.55

total_cost = main_cost + desert_cost + drink_cost

print(total_cost)
```

← test.

```
gst = total_cost * 0.15
total_inc_gst = total_cost + gst
```

print as well

Save with a logical name: dinner_cost_component

③ just copy from ②

```
MAIN = 12.5
DESERT = 6.0
DRINK = 3.55

num_mains = 3
num_deserts = 2
num_drinks = 4

main_cost = num_mains * 12.5
desert_cost = num_desert * 6.0
drinks_cost = num_drinks * 3.55

total_cost = main_cost + desert_cost + drink_cost

gst = total_cost * 0.15
total_inc_gst = total_cost + gst

#print receipt
print("-" * 40)
print("Bills' Diner")
print("- - -")

print("{:3}{:10} at ${:<10.2f} = ${:.2f}".format(num_mains, "mains", MAIN, main_cost))
```

variables go in order into brackets.

2 decimal places.

3 spaces. 10 spaces. 10 spaces, left align, 2 decimal places.

Repeat for deserts and drinks.

```
# print last part.
print("{:>27} = ${:.2f}".format("Total", total_cost))
```

Repeat for gst and total cost including gst.

```
#constants
MAIN_PRICE = 12.5
DESSERT_PRICE = 6.0
DRINKS_PRICE = 3.55

#totalling variables (initially declared as 0)
mains_total = 0.0
desserts_total = 0.0
drinks_total = 0.0
ext_gst = 0.0

# input variables
mains = int(input("How many mains would you like to order?: "))
desserts = int(input("How many desserts would you like to order?: "))
drinks = int(input("How many drinks would you like to order?: "))

#calulate costs
mains_total = mains * MAIN_PRICE
desserts_total = desserts * DESSERT_PRICE
drinks_total = drinks * DRINKS_PRICE

ext_gst = mains_total + desserts_total + drinks_total
my_gst = ext_gst*0.15
my_to_pay = ext_gst + my_gst

print("-"*40)
print("Bill's Diner")
print("-"*40)
print("{:<3}{:10}at ${:<10.2f}= ${:.2f}".format(mains,"mains", MAIN_PRICE,
    mains_total))
print("{:<3}{:10}at ${:<10.2f}= ${:.2f}".format(desserts,"desserts",
    DESSERT_PRICE, desserts_total))
print("{:<3}{:10}at ${:<10.2f}= ${:.2f}".format(drinks,"drinks", DRINKS_PRICE,
    drinks_total))
print("-"*40)
print("{:>27}= ${}".format("Total  ", ext_gst))
print("{:>27}= ${:.2f}".format("GST  ", my_gst))
print("{:>27}= ${:.2f}".format("To Pay  ", my_to_pay))
print("-"*40)
print("Thankyou")
print("-"*40)
```

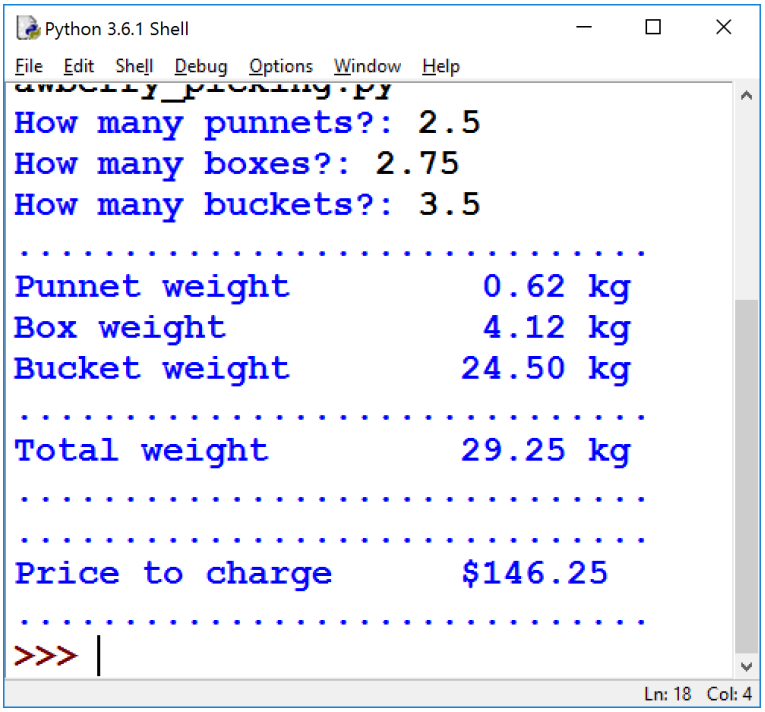
Listing 1: Dinner Order Program

2.1 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.



An example of Strawberries program run

```
# get number of punnets boxes buckets
my_punnets = float(input("How many punnets?: "))
my_boxes = float(input("How many boxes?: "))
my_buckets = float(input("How many buckets?: "))

#testing
#my_punnets = 2.5
#my_boxes = 1.75
#my_buckets = 3.5

punnet_weight = my_punnets*0.25
box_weight = my_boxes*1.5
bucket_weight = my_buckets*7

total_weight = punnet_weight + box_weight + bucket_weight

total_price = total_weight * 5
print("."*30)
print("{:20}{:>6.2f} kg".format("Punnet weight" , punnet_weight))
print("{:20}{:>6.2f} kg".format("Box weight" , box_weight))
print("{:20}{:>6.2f} kg".format("Bucket weight", bucket_weight))
print("."*30)
print("{:20}{:>6.2f} kg".format("Total weight", total_weight))
print("."*30)
print("."*30)
print("{:20} ${:>6.2f}".format("Price to charge", total_price))
print("."*30)
```

Listing 2: Python example

3 Conditions - 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 there 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:

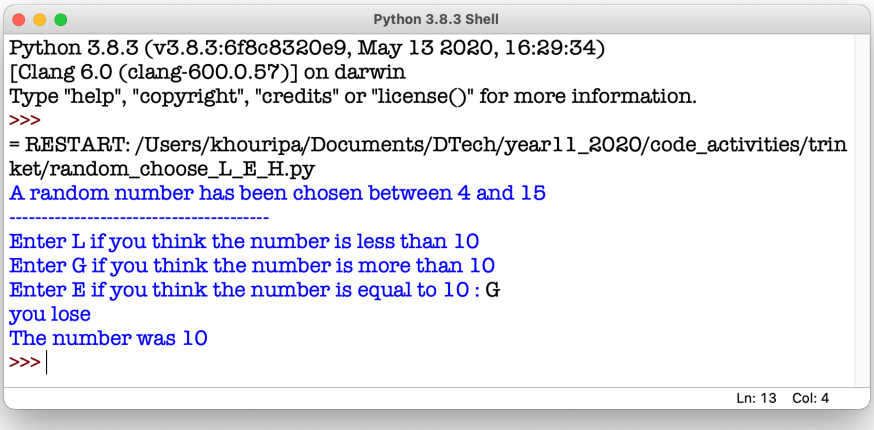
   - over the age of 20
   - **or** be 16 or older and have level 3 NCEA
   - **and** (for both conditions) the person need to be competent with English.

   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.

4 Guess greater , less or equal.

Write 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



An example of greater , less , equal program run

For this program:

- A pseudo code plan.
- Basic test grid.

Pseudo Code

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

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 = round. (lower + higher) / 2

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_rand > mid_num
    print ("win")
elif: guess is "E" and my_rand == mid_n
    print ("win")
else:
    print (lose).

End program.

Test Grid

Test Table: lower = 10, higher = 20 => middle number = 15

my_random =	user input =	expect	actual
12	L	win	
12	G	lose	
12	E	lose	
12	H	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.	

Psuedo code and test grid

```python
import random
lower = 4
higher = 15
my_random = random.randint(lower,higher)
middle_number = round((lower + higher)/2)
#print(middle_number)
#print(my_random)

print("A random number has been chosen between {} and {}".format(lower, higher))
print(40*"-")
guess = input("Enter L if you think the number is less than {0} \n"
              "Enter G if you think the number is more than {0} \n"
              "Enter E if you think the number is equal to {0} : ".format(
   middle_number))

if guess == "L" and my_random < middle_number:
  print("you win")
elif guess == "G" and my_random > middle_number:
  print("you win")
elif guess == "E" and my_random == middle_number:
  print("you win")
elif guess != "G" and guess != "L" and guess != "E":
  print("you have not entered an appropriate letter")
else:
  print("you lose")
print("The number was {}".format(my_random))
```

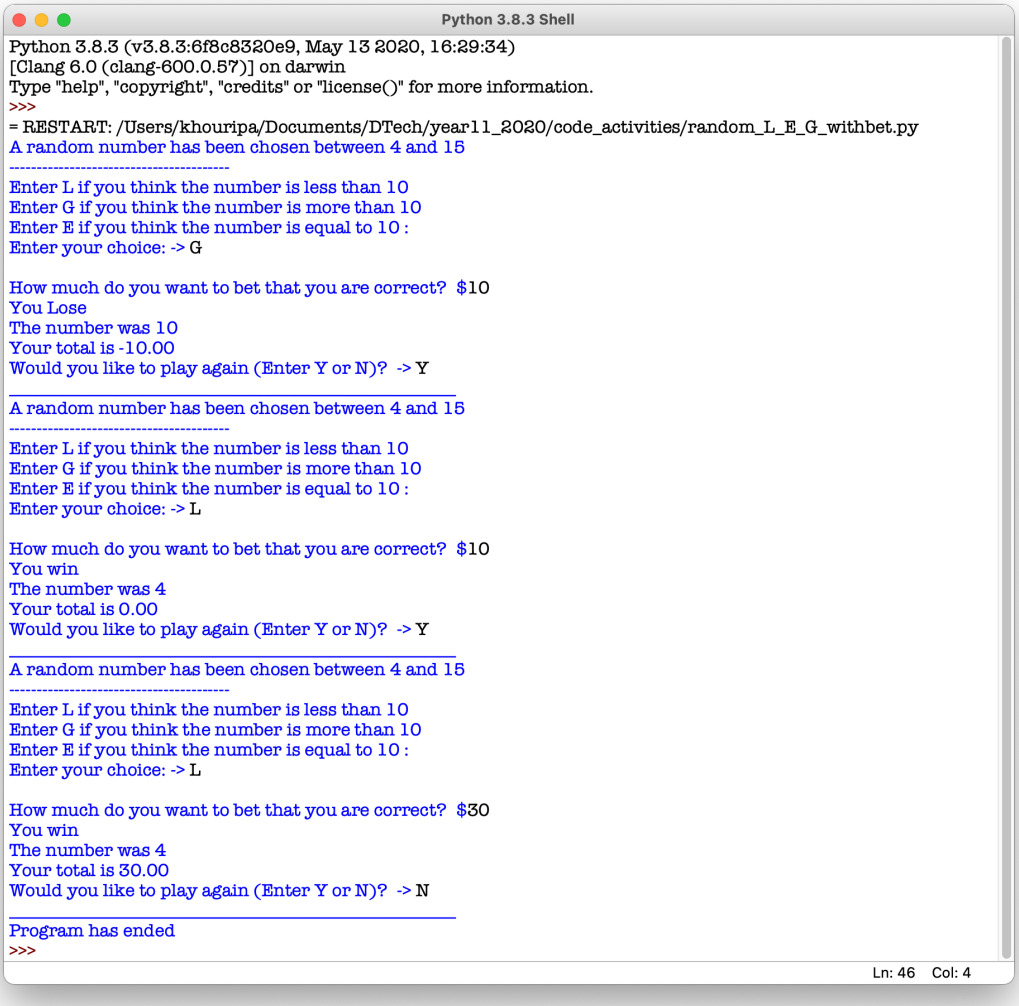
Listing 3: Python example

5 Loop 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

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



Greater less equal, with loop and betting

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

import random module
set lower =
    higher =          # limits for range 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 valuese to total money
                                total_money += bet
    elif False
        print lose
        substract 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
        set game loop to false.
end of loop
    print thankyou for playing.
```

Psuedo code: Greater less equal, with loop and betting

5.1 Higher Lower Game

5.1.1 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'.
- If the user correctly guesses the number, the game should congratulate them.

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.

Developing the game:
At the end of each game, the user should be asked if they would like to play again
For the ones below, the problems will need to be worked out in a separate small program, before they are implemented in the next version:

- Allow the user to choose the lowest and highest number that will be used as the secret number
- this should be validated, so it is probably a good idea that you have a validation function,
- to develop this properly you should be able to validate it so the lower number is less than the high number.
- 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.

Variations (recommended): Please consider adding in the following features. You should develop (and test) these as part of the problem decomposition.

- Allow the user to choose the lowest and highest number that will be used as the secret number
- 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.
- Ask the user how many rounds they would like to play in a given game. Choose a sensible maximum for this and justify your decision

- Keep track of the number of rounds played and the user's scores
- At the end of the given number of rounds, tell the user what their best, worst and average scores were. Ask them if they wish to play again.

5.1.2 Planning

Task:
As well as a working program , you need to provide evidence showing how the problem has been decomposed, how the components have been developed and trialled, and of how they have been assembled and tested to create a final, working outcome.

Decompose the problem (write down the decomposition on the template supplied)
For each part of the problem, write (and test) each piece of code.
Before you write a piece of code, you should generate a quick test plan so that you can confirm that the code works correctly.
Place your test plan and testing evidence on the supplied template.
Combine your code into a fully working program
Test and debug your program to ensure that it works for expected, boundary and unexpected values.
Ask a friend / parent to play your game.
Watch them as they do this and make note of any changes that could be made to make the game easier to use
Make the changes identified in the previous step
Retest your game to ensure that it still works correctly

- Outline / Decomposition:
  Break the problem down into parts.
  Clearly communicate the things you will need to be able to do, to make the program work.
  Provide clear evidence of the Decomposition (diagrams or lists)
- Build relevant test files (version 0 files) to solve/explore coding problems.
- Component Testing (happens after the Decomposition and at any other relevant time)
  Show that you have tested each component.
  You should have a test plan (how do you know it is working properly) and then screenshot proof for each component.
  You should also include notes that justify the major decisions you made.
- Version Log
  Log your different versions HigherLower-v1, HigherLower-v2, etc. (each of these is an iteration)
  Include a brief description of what that have achieved.

- Version Testing

  Please show testing for your assembled outcome below. This should include a test plan followed by screenshot proof Clearly communicate how you can verify that the version is working properly.

- Usability Testing

  Try the program out on someone else and make notes or video etc. Write a list of things improvements which need to be made based on your usability testing. Then write down what you changed.

- Post Usability Test...

  Show that your post usability testing program works correctly Social and End User Considerations...

- General Evaluation:

  How good is may program? What can I do to improve it? Repeat the relevant parts of the above process

```
import random

mainLoop="y"
while mainLoop=="y":
    #declare variables
    name=""
    secretNumber=0
    guess=0
    count=0
    #get user input and validate
    getUserNameLoop="y"
    while getUserNameLoop=="y":
        name=input("Please enter your name   ")
        if len(name)<2 or len(name)>30 or name.isalpha()!=True:
            print("Please enter a valid name")
            continue
        else:
            #name has been captured correctly
            getUserNameLoop="n"
    # welcome
    print("Hello {}, you are about to play a guessing game".format(name))
    print("where you need to guess the secret number between 1 and 50")
    secretNumber=random.randint(1,50)
    #testing help
    print("Testing help: the secret number is {}".format(secretNumber))
    ready=input("Press enter to start  ")
    #start game
    gameLoop="y"
    while gameLoop=="y":
        # get guess and validate
        getGuessInputLoop="y"
        while getGuessInputLoop=="y":
            try:
                guess=int(input("Please guess the secret number between 1 and 50
    "))
            except ValueError:
                print("Please enter a valid integer")
                continue
            if guess<1 or guess>50:
                print("Please enter a number between 1 and 50")
            else:
                # integer correctly entered
                getGuessInputLoop="n"
        #guess count
        count=count+1
        #respond to input
        if guess<secretNumber:
```

```
            print("Your guess is too small")
        elif guess>secretNumber:
            print("Your guess is too large")
        else:
            #number is found, end loop and print responses
            print("You have found it!!")
            gameLoop="n"
            print("You took {} tries to find it".format(count))
            if count<5:
                print("You are really fast")
            elif count<7:
                print("Pretty good")
            else:
                print("You can improve, I think")
    #exited game loop
    mainLoop=input("Do you want to play again?  y/n     ")
#exited main loop
print("The program has ended")
```

Listing 4: Python example

```
>>>
= 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
>>>
```

6 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)

6.1 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.

```
hamburger_price = 7.5
#note the casting int(    ----   )
#this converts the string input to an integer
num_burgers = int(input("How many hamburgers would you like? -> "))

total_cost = num_burgers*hamburger_price
print("Please pay ${:.2f}".format(total_cost))
```

Listing 5: Python example

This works properly assuming the user enters a valid input.
This is 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 <mod
ule>
    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.

```
hamburger_price = 7.5

try:
    num_burgers = int(input("How many hamburgers would you like? -> "))
except ValueError:
    print("The entry is not a valid integer")

#total_cost = num_burgers*hamburger_price
#print("Please pay ${:.2f}".format(total_cost))
```

Listing 6: 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:

```
hamburger_price = 7.5
#set a condition for the loop
have_value = False
while have_value == False:
    try:
        num_burgers = int(input("How many hamburgers would you like? -> "))
    except ValueError:
        # a pleasant, helpful error message.
        print("Unfortunately you have not given a whole number for the hamburgers. Please try again")
        # introduce a 'continue', this means the program will jump straight back to the top of the loop.
        continue
    #if we get to this point then we have a valid integer, so end the loop
    have_value = True

# we now know that we have what we need, so can proceed
total_cost = num_burgers*hamburger_price
print("Please pay ${:.2f}".format(total_cost))
```

Listing 7: 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 the validation loop

```
hamburger_price = 7.5
#set a condition for the loop
have_value = False
while have_value == False:
    try:
        num_burgers = int(input("How many hamburgers would you like? -> "))
    except ValueError:
        # a pleasant, helpful error message.
        print("Unfortunately you have not given a whole number for the hamburgers. Please try again")
        # introduce a 'continue', this means the program will jump straight back to the top of the loop.
        continue
        # this will now request re-entry if the number of burgers is outside an acceptable range
    if num_burgers < 1 or num_burgers > 10:
        print("Please enter at least one burger and no more than 10")
        continue
    #if we get to this point then we have a valid integer, so end the loop
    have_value = True

# we now know that we have what we need, so can proceed
total_cost = num_burgers*hamburger_price
print("Please pay ${:.2f}".format(total_cost))
```

Listing 8: 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
>>>
```

6.2 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?).

```
# set a loop variable
have_name = False

while have_name == False:
    # request name
    name = input("Please enter your name: -> ")

    # check the number of characters in the name and
    # print error message if it is outside of an acceptable range.
    # and continue to top of loop is it is
    if len(name)< 2 or len(name) > 20:
        print("This doesn't look like a valid name, please try again")
        continue

    # if have got here, then all okay so end loop
    have_name = True

print("Welome {}".format(name))
```

Listing 9: 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: -> ksdhksfjhkdsfhkdhfkdfhkdfhkdhkdsfh
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.

```
chars = ["a","b","c","d","e","f","g","h","i","j"," k","l","m","n","o","p","q","r"
    ,"s","t","u","v","w","x","y","z", " ", "-"]
have_name = False
while have_name == False:

    name = input("Please enter your name: -> ")
    # check name lenght and continue to the top
    if len(name)< 2 or len(name) > 20:
        print("This doesn't look like a valid name, please try again")
```

```
        continue
    # assume all the charcters are acceptable
    chars_okay = True
    # loop through the letters in the name (that have been set to lower case)
    for x in name.lower():
        # if an unacceptable character is found
        # print a useful message
        # set chars_okay to be false
        if x not in chars:
            print("{} is not an acceptable character in a name.".format(x))
            chars_okay = False
    # once the lettter loop is finished
    # check if chars_okay is false and if it is
    # continue to top of loop
    if chars_okay == False:
        continue

    # if no problems
    have_name = True

print("Welome {}".format(name.title()))
```

Listing 10: 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
Welome 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

- 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

7 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.
There names are: Agatha , Melinda, Harriet, Hine, Dandelion and Te Aroha
To create a list we use square brackets and commas.

```
# each item in the list has an index number 0,1,2,3,4,5
# the first item is at index 0 (Agatha)
# the length of the list is 6 (because it has 6 items in it)
# but the last index number is 6-1 = 5 ( because the counting starts from 0 )

my_class = ["Agatha" ," Melinda", "Harriet", "Hine", "Dandelion" , "Te Aroha"]
```

Listing 11: Python example

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

```
my_class = ["Agatha" ,"Melinda", "Harriet", "Hine", "Dandelion" , "Te Aroha"]

#print out a name at a specific index (use the square bracket notation)
# this prints out the fourth person in the last
print(my_class[3])
print()
# print the list out "informally"
print(my_class)
print()
# print using a loop (this is the usual way to do it)
# this a shorthand way of doing it,
# where the "x" is, we can use anything we like and it
# will hold the value on each iteration of the loop
for x in my_class:
    print(x)
print()
# find out how long the list is (you often need to do this)
class_length = len(my_class)
output = "There are {} students in the class\n".format(class_length)
print(output)
# sort the list
my_class.sort()
print(my_class)
print()
# shuffle the list
# for this we need the random module(should be up the top)
import random
random.shuffle(my_class)
```

```
print(my_class)
print()
# add a new name to a list
my_class.append("Phoebe")
print(my_class)
print()
# check if someone is in the list
check_name = "Jane"
if check_name in my_class:
    print("{} is in the class".format(check_name))
else:
    print("{} is not in the class".format(check_name))
print()
# find the index number of a particular name (where is Harriet??)
index_num = my_class.index("Harriet")
print("{} is at index number {}".format("Harriet" , index_num))
print()
# remove someone from the list
my_class.remove("Harriet")
print(my_class)
print("{} is no longer in the list".format("Harriet"))
```

Listing 12: Python example

See also:

- Python Lists: https://www.w3schools.com/python/python_lists.asp
- Python List Methods: https://www.w3schools.com/python/python_lists_methods.asp

8 Lists and Loops

8.1 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:

One 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)

8.2 Lucky Unicorn Game

8.2.1 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.

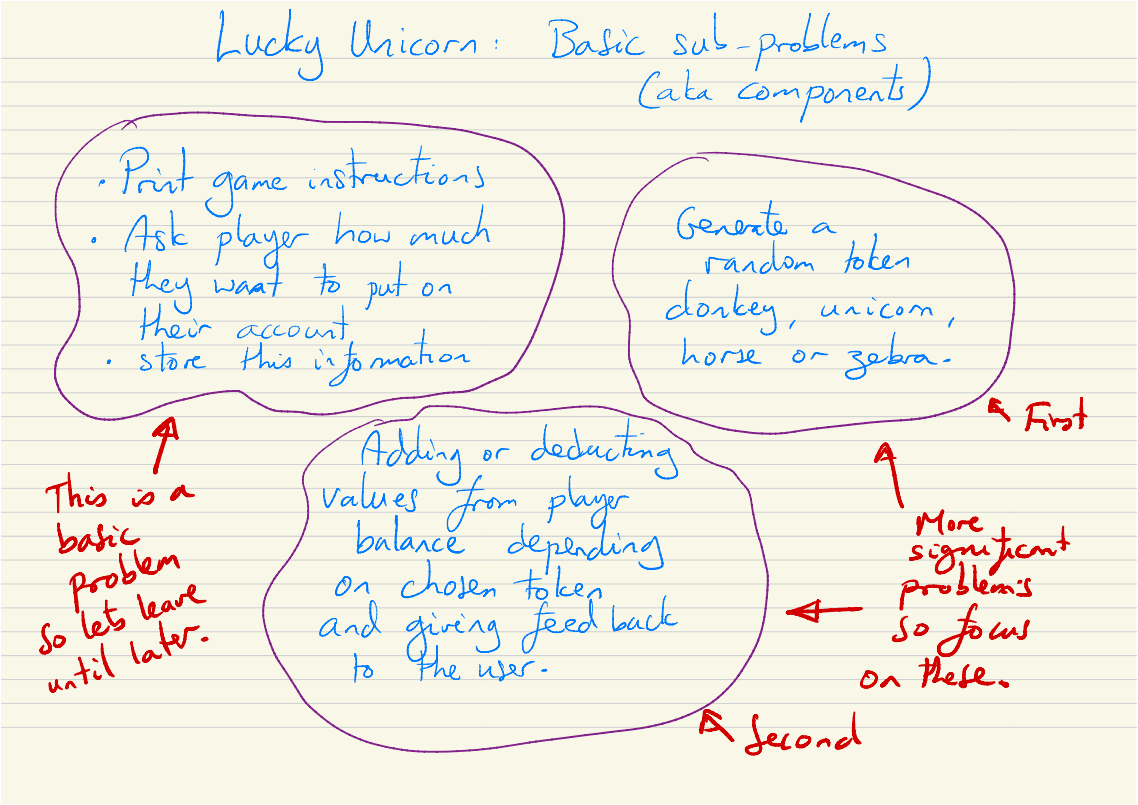
Once students have no more money, the game should end (although players do have the option of quitting while they are ahead).

Variations (optional) Once you have a working game, you are welcome to develop the outcome further.
Here are some variations you could consider...

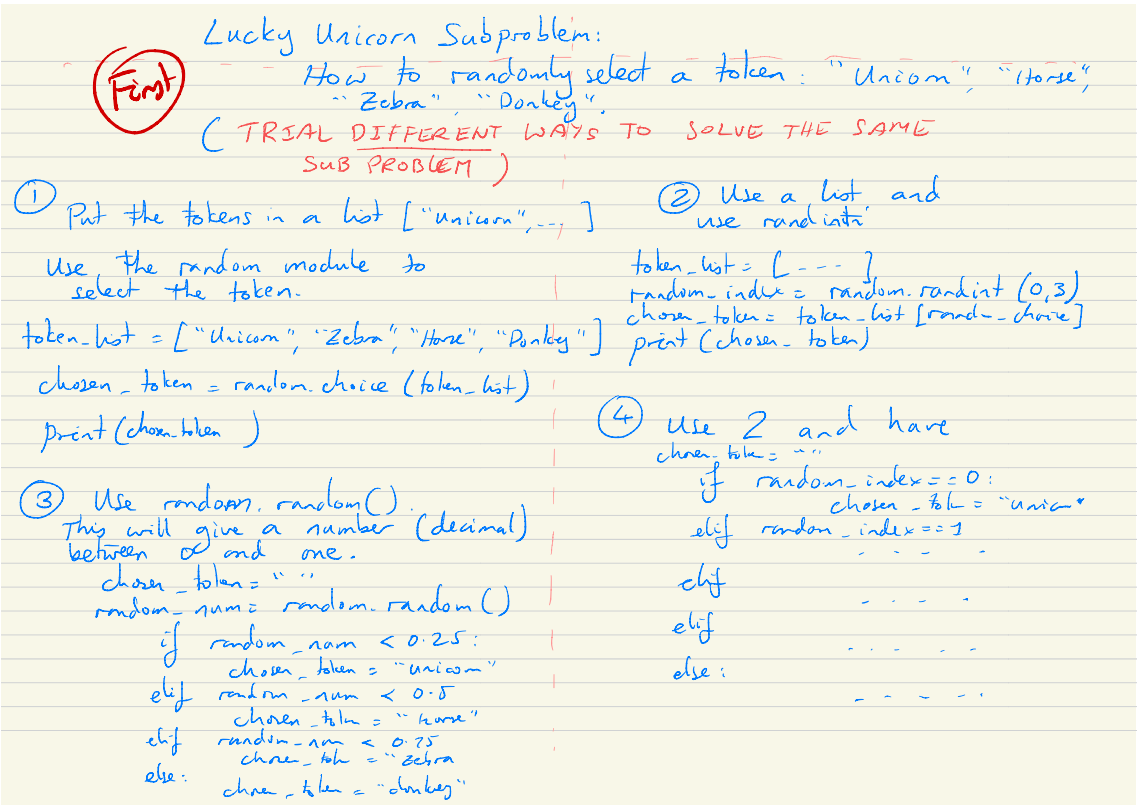
- Change the tokens / context (so rather than having zebra, horses, donkeys and unicorns the game could involve other items)
- Allow users to bet more than $1 per round and adjust the pay-outs proportionally
  (be careful to set up you game so that the house has a long term advantage)

8.2.2 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 "trialling" 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



Breaking the problem down into components



Planning options for the first sub-problem

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

get_token_1.py - /Users/khouripa/Documents/DTech/year11_2020/co...

```
# 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
unicorn
```

get_token_2.py - /Users/khouripa/Documents/DTech/year11_2020/code_activities/l...

```
#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
```

get_token_3.py - /Users/khouripa/Documents/DTech/year11_2020/code_activities/luckyunicorn/sub_pr...

```
#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
>>>
```

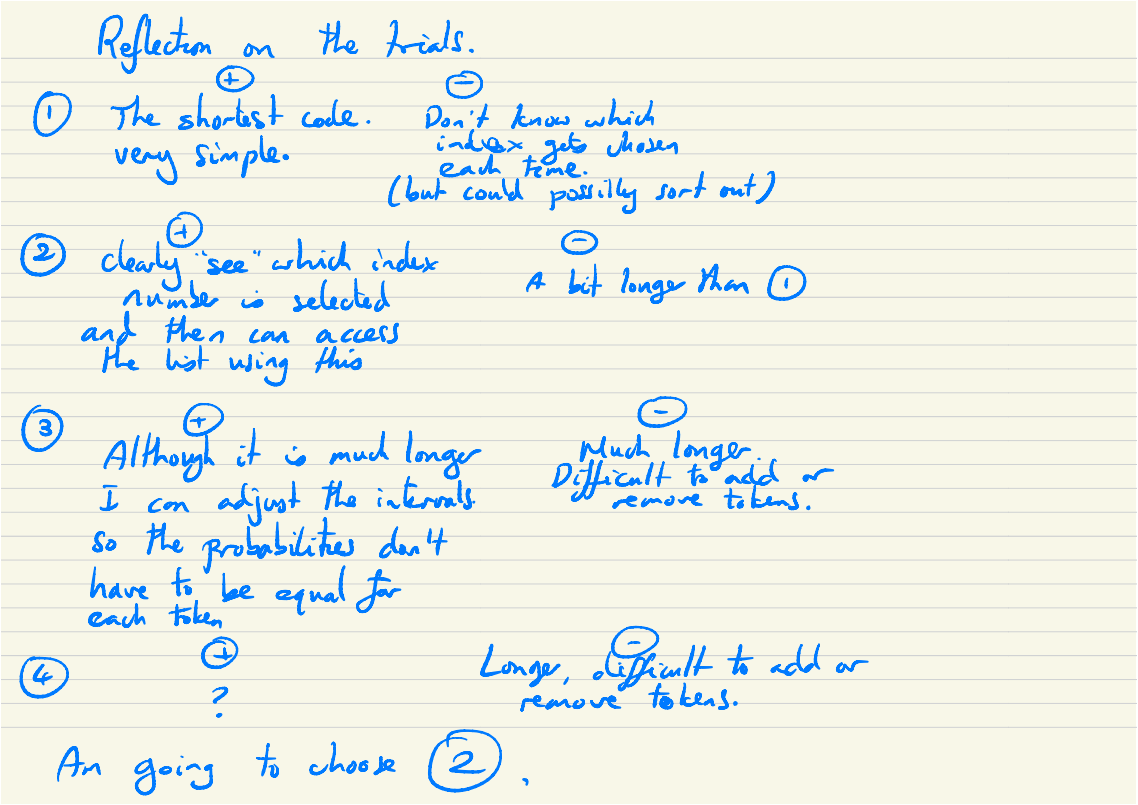
get_token_4.py - /Users/khouripa/Documents/DTech/year11_2020/code_activities/luckyunicorn/sub_pr...

```
#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)
```

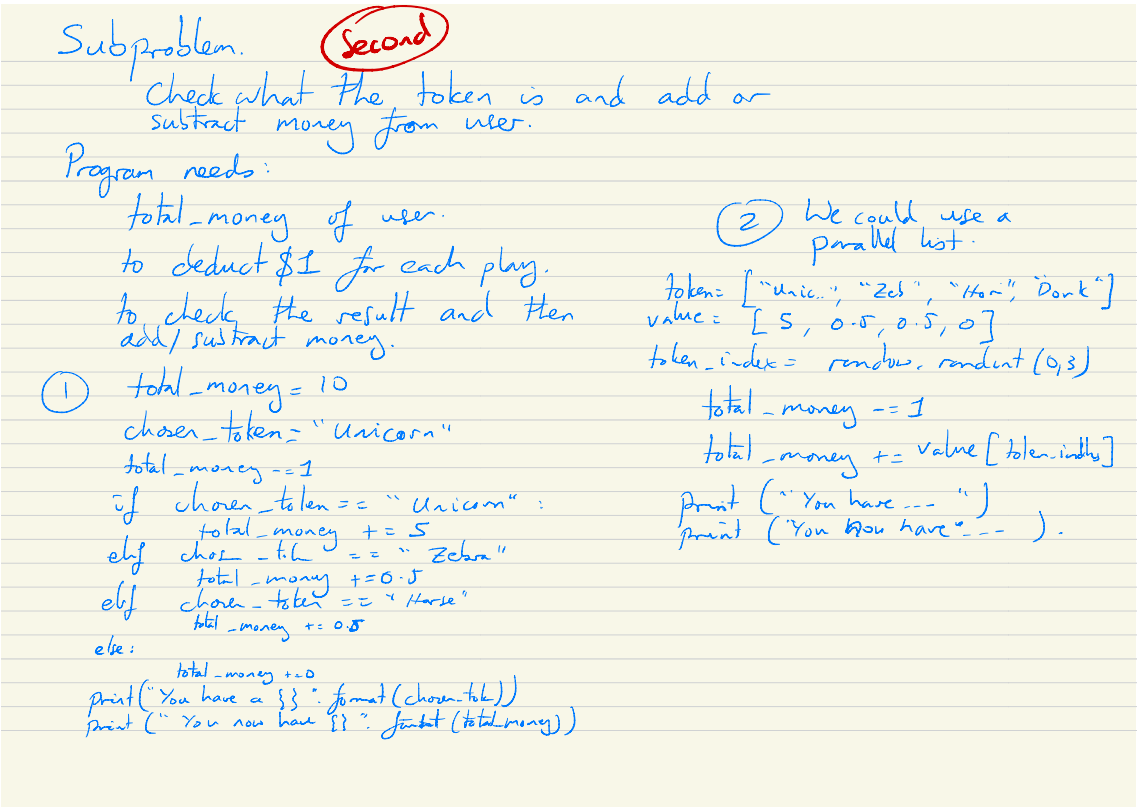
Ln: 15 Col: 4

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

Trialling different ways to solve the same sub-problem



A reflection on the trials and a choice



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

```
>>>
= 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
>>>
```

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/luckyunicorn/sub_proj
ests/add_subtract_feed_back_1.py
You got a horse
You now have $9.50
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/luckyunicorn/sub_proj
ests/add_subtract_feed_back_1.py
You got a unicorn
You now have $14.00
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/luckyunicorn/sub_proj
ests/add_subtract_feed_back_1.py
You got a unicorn
You now have $14.00
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/luckyunicorn/sub_proj
ests/add_subtract_feed_back_1.py
You got a zebra
You now have $9.50
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/luckyunicorn/sub_proj
ests/add_subtract_feed_back_1.py
You got a zebra
You now have $9.50
>>>
= RESTART: /Users/khouripa/Documents/DTech/year11_2020/code_activities/luckyunicorn/sub_proj
ests/add_subtract_feed_back_1.py
You got a donkey
You now have $9.00
>>>
```

Improvement after the first trial

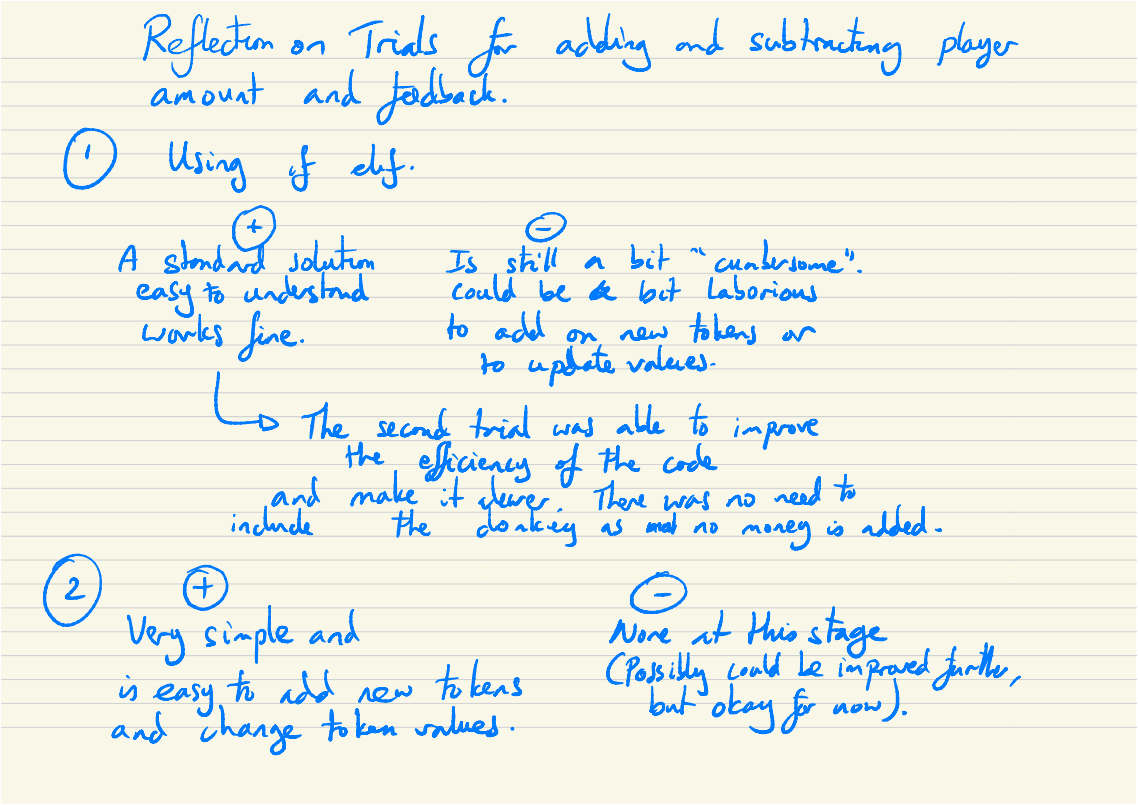
add_subtract_feed_back-a.py - /Users/khouripa/Documents/DTech/year11_2020/code_activities/luckyunicorn/sub_problem...

```
# after the first trial of this idea
# I modified it to simplify the
# conditions so it is now mush 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/luck
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/luck
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/luck
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/luck
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)



A reflection on the trials and a choice

```
import random
# print output for start of Lucky Unicorn Game
print("------ Welcome to the Lucky Unicorn Game ---------")
print("Enter between $1 and $10 to play")
print("It costs $1 per round")
print("If you get a unicorn you will get $5 back")
print("If it is a Zebra or Horse you get 50c back")
print("A donkey gives nothing back")
print("------ Start -------------------------------------")
print()
# get amount of money to play with
tokens = ["horse", "zebra" , "donkey", "unicorn"]
player_amount=int(input("Please enter a value between 1 and 10: -> "))
print("You have entered ${:.2f}".format(player_amount))
#game play
play=""
while play=="":
    player_token = random.choice(tokens)
    player_amount -= 1
    if player_token == "unicorn":
        player_amount += 5
        print("Congratulations, you had a {}".format(player_token))
    elif player_token == "horse" or player_token == "zebra":
        player_amount +=0.5
        print("You had a {}".format(player_token))
    else:
        print("Unfortunately you had a {}".format(player_token))
    print("Your balance is now ${:.2f}".format(player_amount))
    play=input("Press <Enter> to continue or quit(q)")
```

Listing 13: Python example

9 Functions

```
import random
def addition_question():
    a = random.randint(5,9)
    b = random.randint(7,10)
    my_sum = a + b
    print("{} + {} =".format(a,b), end="")
    return my_sum


question_number = addition_question()
answer = int(input(" "))
if answer == question_number:
    print("Correct")
```

Listing 14: Python example

```


my_string = "hello"

def test_function():
    global my_string
    my_string = "bob"

test_function()
print(my_string)
```

Listing 15: Python example

10 Testing

11 Dictionaries

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = car.setdefault("model", "Bronco")

print(x)

quiz_one = {
    "question" : "What is the capital of France? ",
    "answer"   : "Paris"
    }
quiz_two = {
    "question" : "What is the capital of Spain? ",
    "answer"   : "Madrid"
    }
quiz_three = {
    "question" : "What is the capital of Germany? ",
    "answer"   : "Berlin"
    }
print(quiz_one.get("question"))

user_answer = input( quiz_one.get("question") )
if user_answer == quiz_one.get("answer"):
    print("Correct")
user_answer = input( quiz_two.get("question") )
if user_answer == quiz_two.get("answer"):
    print("Correct")
user_answer = input( quiz_three.get("question") )
if user_answer == quiz_three.get("answer"):
    print("Correct")

quiz_set = [
    {
    "question" : "What is the capital of France? ",
    "answer"   : "Paris"
    },
    {
    "question" : "What is the capital of Spain? ",
    "answer"   : "Madrid"
    },
    {
    "question" : "What is the capital of Germany? ",
```

```
    "answer"    : "Berlin"
    }
    ]

for x in quiz_set:
    user_answer = input( x.get("question") )
    if user_answer == x.get("answer"):
        print("Correct")
```

Listing 16: Python example

```
question_one = {
    "question": "What is the capital of France?",
    "A": "Paris",
    "B": "Moscow",
    "C": "Shanghai",
    "answer": "C"
    }
print(question_one["A"])
print(question_one["B"])
print("-"*30)

for x,y in question_one.items():
    if x in ["A", "B", "C"]:
        print("{} , {}".format(x,y))
```

Listing 17: Python example

12 Coffee

```python
import random

char_list = ["C", "O", "F", "E"]



def run_trial():
    counter  = 1
    C_count = 0
    O_count = 0
    F_count = 0
    E_count = 0
    char_string = ""
    while True:
        my_char = random.choice(char_list)
        char_string += my_char
        # count number of C, O,
        C_count = char_string.count("C")
        O_count = char_string.count("O")
        F_count = char_string.count("F")
        E_count = char_string.count("E")
        if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
            break
        # increment count
        counter += 1
    #print(char_string)
    return len(char_string)


total_trials = 100
trials = 1
total_chars = 0
while trials <= total_trials:
    total_chars += run_trial()
    trials += 1


print(total_chars)
print(total_trials)

my_average = total_chars/total_trials
print(my_average)
```

Listing 18: Python example

```
import random

my_list=["c", "o", "f", "e"]

my_string = ""
count = 1
while count<100:
    my_letter = random.choice(my_list)
    my_string += my_letter
    #print(my_letter)
    if my_string.count("c")>=1 and my_string.count("o")>=1 \
       and my_string.count("f")>=2 and my_string.count("e")>=2:
        break
    count += 1
for x in my_list:
    print("{} * {}".format(x,my_string.count(x)))
print(my_string)
print("count is {}".format(count))
```

Listing 19: Python example

```
import random

my_list=["c", "o", "f", "e"]

my_string = ""
count = 1
while count<100:
    my_letter = random.choice(my_list)
    my_string += my_letter
    #print(my_letter)
    if my_string.count("c")>=1 and my_string.count("o")>=1 \
       and my_string.count("f")>=2 and my_string.count("e")>=2:
        break
    count += 1
for x in my_list:
    print("{} * {}".format(x,my_string.count(x)))
print(my_string)
print("count is {}".format(count))
```

Listing 20: Python example

```
import random

my_list=["c", "o", "f", "e"]

string_list = []

```

```
counter = 0
while counter < 1000:
    my_string = ""
    count = 1
    while True:
        my_letter = random.choice(my_list)
        my_string += my_letter
        #print(my_letter)
        if my_string.count("c")>=1 and my_string.count("o")>=1 \
           and my_string.count("f")>=2 and my_string.count("e")>=2:
            break
        count += 1
    string_list.append(my_string)
    counter += 1
    # print(my_string)
#print(string_list)
string_list.sort(key=len)
print(string_list[0])
print(string_list[- 1])

total_times = 0
number_trials = 0
for x in string_list:
    total_times += len(x)
    number_trials += 1
print(total_times)
print(number_trials)
```

Listing 21: Python example

```
import random

char_list = ["C", "O", "F", "E"]


chars_in_six = 0

C_count = 0
O_count = 0
F_count = 0
E_count = 0


trials = 0
trial_total = 100000
while trials < trial_total:
    counter  = 0
    char_string = ""
```

```
    while counter < 6:
        my_char = random.choice(char_list)
        char_string += my_char
        # count number of C, O,
        C_count = char_string.count("C")
        O_count = char_string.count("O")
        F_count = char_string.count("F")
        E_count = char_string.count("E")
        if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
            chars_in_six +=1
        # increment count
        counter += 1
    #print(char_string)
    trials += 1

print(chars_in_six/trial_total)
#print(char_string.count("C"))
```

Listing 22: Python example

```
import random

char_list = ["C", "O", "F", "E"]

# empty list
string_list = []


#----------------------------
def make_coffee():
    C_count = 0
    O_count = 0
    F_count = 0
    E_count = 0

    char_string = ""
    while True:
        my_char = random.choice(char_list)
        char_string += my_char
        # count number of C, O,
        C_count = char_string.count("C")
        O_count = char_string.count("O")
        F_count = char_string.count("F")
        E_count = char_string.count("E")
        if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
            string_list.append(char_string)
            break
        # 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 23: Python example

```
import random

char_list = ["C", "O", "F", "E"]

string_list = []


def run_trial():
    counter  = 1
    C_count = 0
    O_count = 0
    F_count = 0
    E_count = 0
    char_string = ""
    while True:
        my_char = random.choice(char_list)
        char_string += my_char
        # count number of C, O,
        C_count = char_string.count("C")
        O_count = char_string.count("O")
        F_count = char_string.count("F")
        E_count = char_string.count("E")
        if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
            string_list.append(char_string)
            break
        # increment count
        counter += 1
    #print(char_string)
    return len(char_string)




total_trials = 100
trials = 1
total_chars = 0
while trials <= total_trials:
    total_chars += run_trial()
    trials += 1


print(total_chars)
print(total_trials)
my_average = total_chars/total_trials
print(my_average)

#print(string_list)
```

```
string_list.sort(key = len)
#print(string_list)
print(string_list[0])
print(string_list[-1])
print(len(string_list[0]))
print(len(string_list[-1]))

my_dict = {}
for i in range(len(string_list[0]), len(string_list[-1])+1 ):
    my_dict[i]=0
    #print(i)

#print(my_dict)
for x in string_list:
    my_dict[len(x)]+=1
    #print(x)
print(my_dict)
my_sum = 0
for x in my_dict:
    my_sum += my_dict[x]
print(my_sum)
# dictionary not the same lenght as string list
for x,y in my_dict.items():
    temp = int(y)
    temp = temp/total_trials
    print(temp)
    my_dict[x]= temp
print(my_dict)
```

Listing 24: Python example

```
import random

char_list = ["C", "O", "F", "E"]


chars_in_seven = 0

C_count = 0
O_count = 0
F_count = 0
E_count = 0


trials = 0
trial_total = 1000000
while trials < trial_total:
    counter  = 1
```

```
    char_string = ""
    while counter <= 7:
        my_char = random.choice(char_list)
        char_string += my_char
        # count number of C, O, F, E
        C_count = char_string.count("C")
        O_count = char_string.count("O")
        F_count = char_string.count("F")
        E_count = char_string.count("E")

        # increment count
        counter += 1
    # test string at end
    if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
        chars_in_seven +=1
    #print(char_string)
    trials += 1

print(chars_in_seven/trial_total)
#print(char_string.count("C"))
```

Listing 25: Python example

```
import random

char_list = ["C", "O", "F", "E"]


chars_in_eight = 0

C_count = 0
O_count = 0
F_count = 0
E_count = 0


trials = 0
trial_total = 1000000
while trials < trial_total:
    counter  = 1
    char_string = ""
    while counter <= 8:
        my_char = random.choice(char_list)
        char_string += my_char
        # count number of C, O, F, E
        C_count = char_string.count("C")
        O_count = char_string.count("O")
        F_count = char_string.count("F")
```

```
        E_count = char_string.count("E")

        # increment count
        counter += 1
    # test string at end
    if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
        chars_in_eight +=1
    #print(char_string)
    trials += 1

print(chars_in_eight/trial_total)
#print(char_string.count("C"))
```

Listing 26: Python example

```
import random

char_list = ["C", "O", "F", "E"]



C_count = 0
O_count = 0
F_count = 0
E_count = 0

char_string = ""
while True:
    my_char = random.choice(char_list)
    char_string += my_char
    # count number of C, O,
    C_count = char_string.count("C")
    O_count = char_string.count("O")
    F_count = char_string.count("F")
    E_count = char_string.count("E")
    if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
        break
    # increment count


print(char_string)
print( len(char_string) )

#print(char_string.count("C"))
```

Listing 27: Python example

```
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)
```

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

Listing 28: Python example

```
import random

char_list = ["C", "O", "F", "E"]

string_list = []


def run_trial():
    counter  = 1
    C_count = 0
    O_count = 0
    F_count = 0
    E_count = 0
    char_string = ""
    while True:
        my_char = random.choice(char_list)
        char_string += my_char
        # count number of C, O,
        C_count = char_string.count("C")
        O_count = char_string.count("O")
        F_count = char_string.count("F")
        E_count = char_string.count("E")
        if C_count >= 1 and O_count >= 1 and F_count >= 2 and E_count >= 2:
            string_list.append(char_string)
            break
        # increment count
        counter += 1
    #print(char_string)
    return len(char_string)




total_trials = 1000000
trials = 1
total_chars = 0
while trials <= total_trials:
    total_chars += run_trial()
    trials += 1


print(total_chars)
print(total_trials)
my_average = total_chars/total_trials
```

```
print(my_average)

#print(string_list)
string_list.sort(key = len)
#print(string_list)
print(string_list[0])
print(string_list[-1])
print(len(string_list[0]))
print(len(string_list[-1]))

my_dict = {}
for i in range(len(string_list[0]), len(string_list[-1])+1 ):
    my_dict[i]=0
    #print(i)

#print(my_dict)
for x in string_list:
    my_dict[len(x)]+=1

print(my_dict)
my_sum = 0
for x in my_dict:
    my_sum += my_dict[x]
print(my_sum)
# dictionary not the same lenght as string list
for x,y in my_dict.items():
    temp = int(y)
    temp = temp/total_trials
    my_dict[x]= temp
print(my_dict)

my_file = open("distribution_next.txt","a")
for x,y in my_dict.items():
    my_string = "{},{:.6f}\n".format(x,y)
    my_file.write(my_string)

#my_file.write("hello")
#my_file.write("\n")
#my_file.write("goodbye")
my_file.close()
#my_file = open("MyFile.txt","w")
#my_file.write("")
#my_file.close()
```

Listing 29: Python example

13 Tests

```
# roll a dice
import random
two_d_list = []



def searchfor(n):
    if len(two_d_list)== 0:
        two_d_list.append([n,1])
        return
    added = False
    for sublist in two_d_list:
        if sublist[0]==n:
            sublist[1] += 1
            added = True
            break
    if added == False:
        two_d_list.append([n,1])

for i in range(0,1500):
    diceroll = random.randint(1,6)+random.randint(1,6)+random.randint(1,6)
    searchfor(diceroll)
    #print(diceroll)


two_d_list.sort()
print(two_d_list)
```

Listing 30: Dice Roll

```
guess_list = [45,78,90,21,7]

guess=int(input("Enter your guess"))


while guess in guess_list:
    guess = int(input("You have already tried this number,\
have another guess"))
guess_list.append(guess)
print(guess_list)
```

Listing 31: Python example

```
import time
start= time.time()
t= time.localtime()
```

```
print(t)
print(start)



word = "halloween"
char_list = list(word)
print(char_list)
char_list_chosen=["l", "w", "e"]
outstring = ""
for x in char_list:
    if x in char_list_chosen:
        outstring += " "+x+" "
    else:
        outstring += " _ "
print(outstring)




question_one = {
    "question": "What is the capital of France?",
    "A": "Paris",
    "B": "Moscow",
    "C": "Shanghai",
    "answer": "C"
    }

print(question_one["question"])
for x,y in question_one.items():
    if x != "question" and x != "answer":
        print("{}) {}".format(x,y))

for x,y in question_one.items():
    if len(x)== 1:
        print("{}) {}".format(x,y))

answer = input("Enter your choice ")
if answer == question_one["answer"]:
    print(" Brilliant ")
end=time.time()
print(end)
print(end-start)
count = 0
```

```
while count < 3:
    temp={}
    temp["english"]= input("Enter your english word")
    temp["thai"] = input("Enter your thai word")
    print(temp)
    count +=1
```

Listing 32: Python example

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

Listing 33: Python example

```
word = "halloween"

def pic(c):
    if c >= 1:
        print(" "*30+"    _ _ _ _")
    if c >=2:
        print(" "*30+"    |     |")
    if c >=3:
        print(" "*30+"   ( )    |")
    if c >= 4:
        print(" "*30+"    |     |")
    if c >= 5:
        print(" "*30+"   ---    |")
        print(" "*30+"    |     |")
        print(" "*30+"   / \    |")
        print(" "*30+"        __|")
pic(5)
char_list = list(word)
char_individual_list=[]
char_list_chosen=[]

for x in char_list:
    if x not in char_individual_list:
        char_individual_list.append(x)

wrong_count = 0
while len(char_individual_list)>0:
    outstring = ""
    for x in char_list:
        if x in char_list_chosen:
            outstring += " "+x+" "
        else:
            outstring += " _ "
    print("{}\n\n".format(outstring))
```

```
    char = input("guess a letter: ")
    if char in char_individual_list:
        char_list_chosen.append(char)
        char_individual_list.remove(char)
    else:
        wrong_count += 1
        pic(wrong_count)
    if len(char_individual_list) == 0:
        print("you got it")


outstring = ""
for x in char_list:
    if x in char_list_chosen:
        outstring += " "+x+" "
    else:
        outstring += " _ "
print(outstring)
```

Listing 34: Python example

14 Strings

```
my_string="adkadkkdjk  jkjdhkdhdh"
my_split=my_string.split("a")
print(my_split)
my_list=list(my_string)
print(my_list)
```

Listing 35: Python example

```
my_string = "woolloomooloo"

my_letters=[]
count_letters = []

for x in my_string:
    if x not in my_letters:
        my_letters.append(x)
        count_letters.append(my_string.count(x))

print(my_letters)
print(count_letters)
```

Listing 36: Python example

15 Objects

```
class Shark:

    def __init__(self, name, age):
        self.name = name
        self.age = age
        """ constructor """
        print("This is the constructor")

    def swim(self):
        """ what the shark is doing"""
        print(self.name+" is swimming")

    def be_awesome(self):
        """ what the shark is being"""
        print(self.name+" is being awesome")

    def _age(self):
        """ how old is it """
        print(self.name +" is "+str(self.age)+" years old")


def main():
    sammy = Shark("Sammy", 5)
    stevie = Shark("Stevie", 3)
    sammy.swim()
    stevie.be_awesome()
    sammy._age()

if __name__ =="__main__":
    main()
```

Listing 37: Python example

```
class Store:

    def __init__(self):
        self.colour = "rgb(0,0,0,0)"









class Shark:

    def __init__(self, name, age):
        self.name = name
        self.age = age
        """ constructor """
```

```
        store.colour = "rgb(255,255,255,1)"
        print("This is the constructor")

    def swim(self):
        """ what the shark is doing"""
        print(self.name+" is swimming")

    def be_awesome(self):
        """ what the shark is being"""
        print(self.name+" is being awesome")

    def _age(self):
        """ how old is it """
        print(self.name +" is "+str(self.age)+" years old")

store = Store()
shark = Shark("sam", 10)
print(store.colour)
```

Listing 38: Python example

```
class Question:

    def __init__(self, question,answer, score):

        self.question = question
        self.answer = answer
        self.score = score

        self.user_answer = ""
        self.user_score = 0
        self.response_to_user = ""

    def make_question(self):
        msg = self.question
        self.user_answer = input(msg)
        if self.user_answer == self.answer:
            self.response_to_user = "Correct"
            self.user_score = self.score
        else:
            self.response_to_user = "Incorrect"

    def feedback(self):
        strOne = "For the question: "+self.question+ "\n"
        strTwo = "You answered: "+self.user_answer +"\n"
        strThree = "Your answer was "+self.response_to_user
        print(strOne + strTwo + strThree)

```

```
#q_one = Question("What is the capital of France?", "Paris", 10)
#q_one.make_question()
#q_one.feedback()
q_list = []
q_list.append(Question("What is the capital of France?", "Paris", 10))
q_list.append(Question("What is the capital of Belgium?", "Brussels", 10))
q_list.append(Question("What is the capital of Holland?", "Amstredam", 10))

count = 0
while count < len(q_list):
    q_list[count].make_question()
    count += 1

count = 0
while count < len(q_list):
    q_list[count].feedback()
    count += 1
```

Listing 39: Python example