1811ICT/2807ICT/7001ICT Programming Principles Workshop 9

School of Information and Communication Technology

Griffith University

|  |  |
| --- | --- |
| Goals | This workshop focusses on everything in the course up to files. |
| When | Week 10 |
| Marks | 3 |
| Due | Pre-workshop questions before the start of the above mentioned workshops  Workshop programming problems by 11:59pm on 23 May |

# Before your workshop class:

* Read all of this document.
* Review the lecture notes sections 1 to 24.
* **Complete the pre-workshop questions (1 mark) posted on the course website and submit the answers for marking**.

# Workshop activities

At any stage, when you are stuck, *ask your tutor*!

## Problem 1

*Problem:* The people of ancient land of Pacific Baza had a simple mathematical system that knew only natural numbers and addition. The genius Bazan scholar, Gringo el Possum, built a computer from wood and various animal parts. Archeologists have recovered ancient scrolls with enough scraps of programs to reconstruct the programming language he named, *Adder*.

The Adder language has only a few simple statements:

|  |  |
| --- | --- |
| quit | Exit the REPL or terminate a program. |
| input *var* | Prompt for and allow the user to enter a value for the variable named *var*. |
| print *val* | Print the value *val*. |
| *var* gets *val* | variable *var* is assigned the value *val*. |
| *var* adds *val* | variable *var* has the value *val* added to it. |

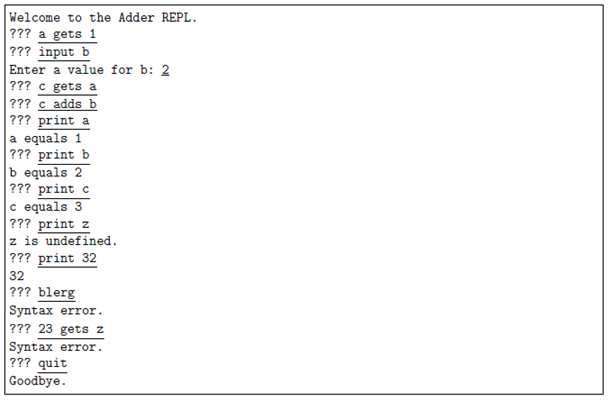
where:

* *var* is always a variable name that contains only letters; and
* *val* can be either:

**–** a variable name that contains only letters; or

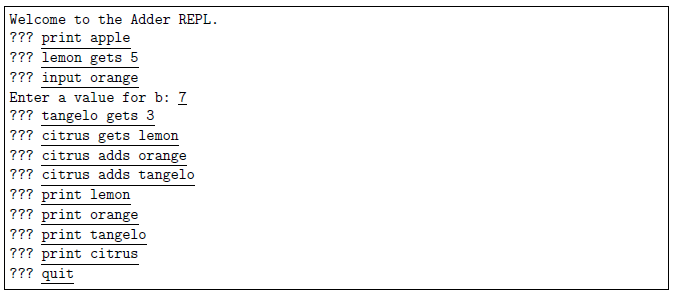
**–** a natural number that contains only digits.

The Adder REPL allows the user to enter commands interactively. For example:



Write the program for the Adder REPL. Hints: Make good use of string methods. Can you divide your program up into smaller pieces by defining functions?

*Answer*: Copy your code in the space given below and insert screenshots of your program output for the following scenario:



***Copy your code here***

class Interpreter:

    def \_\_init\_\_(self):

        print ("Welcome to the Adder REPL")

*self*.dict = {}

    def quit(self):

        print ("Goodbye")

    def input(self, var):

        val = int(input("Enter a value for {}: ".format(var)))

*self*.dict[var] = val

    def print\_var(self,var):

        if var not in *self*.dict:

            print("{} is undefined.".format(var))

        else:

            print ( "{} equals {}".format(var,*self*.dict[var]))

    def get(self, var, val):

        try:

            dec = int(val)

*self*.dict[var] = dec

        except:

            if not val.isalpha(): print("Syntax error.")

            else: *self*.dict[var] =  *self*.dict[val]

    def add(self, var, val):

        try:

            dec = int(val)

*self*.dict[var] += dec

        except:

            if not val.isalpha(): print("Syntax error.")

            else: *self*.dict[var] +=  *self*.dict[val]

    def run\_code(self):

        while True:

            line  =str(input())

            if line == "quit" :

*self*.quit()

                break

            elif "print" in line:

                try:

                    dec = int(line[6:])

                    print(dec)

                except:

                    var = line[6:]

                    if not var.isalpha(): print("Syntax error.")

                    else: *self*.print\_var(var)

            elif "input" in line:

                var = line[6:]

                if not var.isalpha(): print("Syntax error.")

                else: *self*.input(var)

            elif "gets" in line:

                var = line[:line.index("gets")-1]

                val =line[line.index("gets")+5:]

                if not (var.isalpha()): print("Syntax error.")

                else: *self*.get(var, val)

            elif "adds" in line:

                var = line[:line.index("adds")-1]

                val =line[line.index("adds")+5:]

                if not (var.isalpha()): print("Syntax error.")

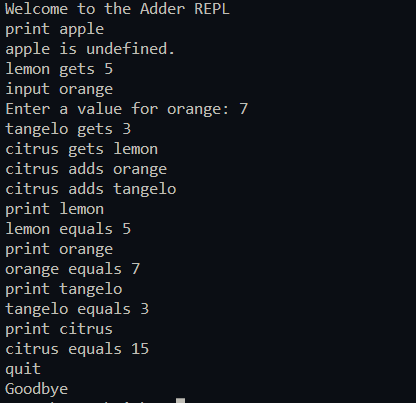
                else: *self*.add(var, val)

            else: print("Syntax error.")

Interpreter = Interpreter()

Interpreter.run\_code()

***Insert your screenshot here***



## Problem 2

*Problem:* Write an Adder interpreter, that prompts for and executes an Adder script. For example if the file children.ad contains:

input sons   
input daughters   
children gets sons   
children adds daughters   
print children   
quit

The interpreter would run like this

Script name: children.ad

Enter a value for sons: 3   
Enter a value for daughters: 4   
children equals 7

Hint: This should involve a few small modifications to your REPL from Problem 1.

*Answer*: Copy your code in the space given below and insert screenshots of your program output for the following scenario:

* Use the file P2.ad as the source file.

***Copy your code here***

class Children:

    def \_\_init\_\_(self):

*self*.dict = {}

    def input(self, var):

        val = int(input("Enter a value for {}: ".format(var)))

*self*.dict[var] = val

    def print\_var(self,var):

        if var not in *self*.dict:

            print("{} is undefined.".format(var))

        else:

            print ( "{} equals {}".format(var,*self*.dict[var]))

    def get(self, var, val):

        try:

            dec = int(val)

*self*.dict[var] = dec

        except:

            if not val.isalpha(): print("Syntax error.")

            else: *self*.dict[var] =  *self*.dict[val]

    def add(self, var, val):

        try:

            dec = int(val)

*self*.dict[var] += dec

        except:

            if not val.isalpha(): print("Syntax error.")

            else: *self*.dict[var] +=  *self*.dict[val]

    def run\_code(self):

        file = open('children.txt', 'r')

        print ("Script name: children.txt")

        while True:

            line = file.readline()

            if line == "quit" :

                break

            elif "print" in line:

                try:

                    dec = int(line[6:-1])

                    print(dec)

                except:

                    var = line[6:-1]

                    if not var.isalpha(): print("Syntax error.")

                    else: *self*.print\_var(var)

            elif "input" in line:

                var = line[6:-1]

                if not var.isalpha(): print("Syntax error.")

                else: *self*.input(var)

            elif "gets" in line:

                var = line[:line.index("gets")-1]

                val =line[line.index("gets")+5:-1]

                if not (var.isalpha()): print("Syntax error.")

                else: *self*.get(var, val)

            elif "adds" in line:

                var = line[:line.index("adds")-1]

                val =line[line.index("adds")+5:-1]

                if not (var.isalpha()): print("Syntax error.")

                else: *self*.add(var, val)

            else: print("Syntax error.")

        file.close()

Children = Children()

Children.run\_code()

***Insert your screenshot here***

A picture containing text, font, screenshot, typography

Description automatically generated

## Problem 3

*Problem:* A road map defines locations as map references like B3, where B is the *x*-coordinate value and 3 is the *y*-coordinate.

A

B

C

D

E

F

Y

Z

1

2

3

4

5

6

25

26

…

⫶

B3

E2

Y25

The grid lines are 0.5 km apart.

Write a program that allows the user to enter a trip as a sequence of any number of map references on one line, and reports the total length of the trip, assuming they can travel in straight lines. For example:

Enter trip map references: C2 B5 Y25

Total distance = 16.8 km

For badly formatted map references, your program should exit, reporting the first bad map reference.

Enter trip map references: E6 E4 D7 d43 F5

Bad reference: d43

Hints: You need to *split* the input line into separate references; each reference starts with one character which must be an upper case letter, and the rest must be only digits; and Pythagoras will help. The function exit() can abort the program if you detect an error in the input.

*Answer*: Copy your code in the space given below and insert screenshots of your program output for the following two scenarios:

* A1 B2 C3 D20 S15 W25 Z26
* D2 F23 Ja E23 Z2

***Copy your code here***

import math

class point:

    def \_\_init\_\_(self, x, y):

*self*.x = x

*self*.y = y

def input\_points(line, points):

    arr = line.split(" ")

    for word in arr:

        x = ord(word[0])

        if x <65 or x >90: return word

        try: y = int(word[1:])

        except: return word

        points.append(point(x-64,y))

    return True

def distance(points):

    road = 0

    for i in range(len(points)-1):

        road += math.sqrt((points[i].x - points[i+1].x)\*\*2 + (points[i].y - points[i+1].y)\*\*2)

    return round(road/2,1)

line = str(input("Enter trip msap references: "))

points = []

if input\_points(line,points) != True:

    print ("Bad reference:",input\_points(line,points))

    exit()

print (distance(points))

***Insert your screenshot here***





# Submission and marking

The pre-workshop can be accessed and submitted online using the provided link in the course website. Students get 1 mark if they get >50% in pre-workshop questions, or 0.5 mark if they get 0%-50% in pre-workshop questions, or 0 marks without any attempt.

For workshop tasks, please submit this document with copied codes and inserted screenshots using the provided submission link in the course website. Students get 2 marks if they complete two or more problems correctly, or 1 mark if they complete one problem correctly, or 0 marks without any attempt.