Appendix A: Working With Strings

Note: The notation `[start, [end]]` means *start* and *end* are optional. If only one number is provided, it is taken to be *start*.

marks the start of a comment
''' marks the start and end of a multiline comment
The actual code is in `monotype` font.
=> marks the start of the output

count (sub, [start, [end]])

Returns the number of times the substring *sub* appears in the string.
This function is case-sensitive.

[Example]

```
# In the examples below, 's' occurs at index 3, 6 and 10

# count the entire string
'This is a string'.count('s')
=> 3

# count from index 4 to end of string
'This is a string'.count('s', 4)
=> 2

# count from index 4 to 10-1
'This is a string'.count('s', 4, 10 )
=> 1

# count 'T'. There's only 1 'T' as the function is case sensitive.
'This is a string'.count('T')
=> 1
```

endswith (suffix, [start, [end]])

Returns `True` if the string ends with the specified *suffix*, otherwise returns `False`.
suffix can also be a tuple of suffixes to look for.
This function is case-sensitive.

[Example]

```
# 'man' occurs at index 4 to 6

# check the entire string
'Postman'.endswith('man')
=> True
```

```
# check from index 3 to end of string
'Postman'.endswith('man', 3)
=> True

# check from index 2 to 6-1
'Postman'.endswith('man', 2, 6)
=> False

# check from index 2 to 7-1
'Postman'.endswith('man', 2, 7)
=> True

# Using a tuple of suffixes (check from index 2 to 6-1)
'Postman'.endswith(('man', 'ma'), 2, 6)
=> True
```

find/index (sub, [start, [end]])

Returns the index in the string where the first occurrence of the substring *sub* is found.
`find()` returns `-1` if *sub* is not found.
`index()` returns `ValueError` if *sub* is not found.
This function is case-sensitive.

[Example]

```
# check the entire string
'This is a string'.find('s')
=> 3

# check from index 4 to end of string
'This is a string'.find('s', 4)
=> 6

# check from index 7 to 11-1
'This is a string'.find('s', 7,11 )
=> 10

# sub is not found
'This is a string'.find('p')
=> -1

'This is a string'.index('p')
=> ValueError
```

isalnum()

Returns `True` if all characters in the string are alphanumeric and there is at least one character, `False` otherwise.
Alphanumeric does not include whitespaces.

[Example]

```
'abcd1234'.isalnum()
=> True

'a b c d 1 2 3 4'.isalnum()
=> False

'abcd'.isalnum()
=> True

'1234'.isalnum()
=> True
```

isalpha()

Returns `True` if all characters in the string are alphabetic and there is at least one character, `False` otherwise.

[Example]

```
'abcd'.isalpha()
=> True

'abcd1234'.isalpha()
=> False

'1234'.isalpha()
=> False

'a b c'.isalpha()
=> False
```

isdigit()

Returns `True` if all characters in the string are digits and there is at least one character, `False` otherwise.

[Example]

```
'1234'.isdigit()
=> True

'abcd1234'.isdigit()
=> False

'abcd'.isdigit()
=> False
```

```
'1 2 3 4'.isdigit()
=> False
```

islower()

Returns `True` if all cased characters in the string are lowercase and there is at least one cased character, `False` otherwise.

[Example]

```
'abcd'.islower()
=> True

'Abcd'.islower()
=> False

'ABCD'.islower()
=> False
```

isspace()

Returns `True` if there are only whitespace characters in the string and there is at least one character, `False` otherwise.

[Example]

```
' '.isspace()
=> True

'a b'.isspace()
=> False
```

istitle()

Returns `True` if the string is a titlecased string and there is at least one character

[Example]

```
'This Is A String'.istitle()
=> True

'This is a string'.istitle()
=> False
```

isupper()

Returns `True` if all cased characters in the string are uppercase and there is at least one cased character, `False` otherwise.

[Example]

```
'ABCD'.isupper()
=> True


'Abcd'.isupper()
=> False

'abcd'.isupper()
=> False
```

join()

Returns a string in which the argument provided is joined by a separator.

[Example]

```
sep = '-'
myTuple = ('a', 'b', 'c')
myList = ['d', 'e', 'f']
myString = "Hello World"

sep.join(myTuple)
=> 'a-b-c'

sep.join(myList)
=> 'd-e-f'

sep.join(myString)
=> 'H-e-l-l-o- -W-o-r-l-d''
```

lower()

Returns a copy of the string converted to lowercase.

[Example]

```
'Hello Python'.lower()
=> 'hello python'
```

replace(old, new[, count])

Returns a copy of the string with all occurrences of substring old replaced by new.
count is optional. If given, only the first *count* occurrences are replaced.
This function is case-sensitive.

[Example]

```
# Replace all occurences
'This is a string'.replace('s', 'p')
```

```
=> 'Thip ip a ptring'

# Replace first 2 occurences
'This is a string'.replace('s', 'p', 2)
=> 'Thip ip a string'
```

split([sep [,maxsplit]])

Returns a list of the words in the string, using *sep* as the delimiter string.
sep and *maxsplit* are optional.
If *sep* is not given, whitespace is used as the delimiter.
If *maxsplit* is given, at most *maxsplit* splits are done.
This function is case-sensitive.

[Example]

```
# Split using whitespace as delimiter
'This is a string'.split()
=> ['This', 'is', 'a', 'string']

# Split using comma followed by a whitespace as the delimiter
'This, is, a, string'.split(', ')
=> ['This', 'is', 'a', 'string']

# Split using comma followed by a whitespace as the delimiter
# Only do 2 splits
'This, is, a, string'.split(', ', 2)
=> ['This', 'is', 'a, string']
```

splitlines ([keepends])

Returns a list of the lines in the string, breaking at line boundaries.
Line breaks are not included in the resulting list unless *keepends* is given and `True`.

[Example]

```
# Split lines separated by \n
'This is the first line.\nThis is the second line'.splitlines()
=> ['This is the first line.', 'This is the second line.']

# Split multi line string (e.g. string that uses the ''' mark)
'''This is the first line.
This is the second line.'''.splitlines()
=> ['This is the first line.', 'This is the second line.']

# Split and keep line breaks
'This is the first line.\nThis is the second line.'.splitlines(True)
=> ['This is the first line.\n', 'This is the second line.']

'''This is the first line.
This is the second line.'''.splitlines(True)
```

```
=> ['This is the first line.\n', 'This is the second line.']
```

startswith (prefix[, start[, end]])

Returns `True` if string starts with the prefix, otherwise returns `False`.
prefix can also be a tuple of prefixes to look for.
This function is case-sensitive.

[Example]

```
# 'Post' occurs at index 0 to 3

# check the entire string
'Postman'.startswith('Post')
=> True

# check from index 3 to end of string
'Postman'.startswith('Post', 3)
=> False

# check from index 2 to 6-1
'Postman'.startswith('Post', 2, 6)
=> False

# check from index 2 to 6-1
'Postman'.startswith('stm', 2, 6)
=> True

# Using a tuple of prefixes (check from index 3 to end of string)
'Postman'.startswith(('Post', 'tma'), 3)
=> True
```

strip ([chars])

Returns a copy of the string with the leading and trailing characters *char* removed.
If *char* is not provided, whitespaces will be removed.
This function is case-sensitive.

[Example]

```
# Strip whitespaces
'   This is a string   '.strip()
=> 'This is a string'

# Strip 's'. Nothing is removed since 's' is not at the start or end of the
string
'This is a string'.strip('s')
=> 'This is a string'

# Strip 'g'.
'This is a string'.strip('g')
```

```
=> 'This is a strin'
```

upper()

Returns a copy of the string converted to uppercase.

[Example]

```
'Hello Python'.upper()
=> 'HELLO PYTHON'
```

Appendix B: Working With Lists

=> marks the start of the output

append( )

Add item to the end of a list

[Example]

```
myList = ['a', 'b', 'c', 'd']
myList.append('e')
print (myList)
=> ['a', 'b', 'c', 'd', 'e']
```

del

Remove items from a list

[Example]

```
myList = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l']

#delete the third item (index = 2)
del myList[2]
print (myList)
=> ['a', 'b', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l']

#delete items from index 1 to 5-1
del myList[1:5]
print (myList)
=> ['a', 'g', 'h', 'i', 'j', 'k', 'l']

#delete items from index 0 to 3-1
del myList [ :3]
print (myList)
=> ['i', 'j', 'k', 'l']

#delete items from index 2 to end
del myList [2:]
print (myList)
=> ['i', 'j']
```

extend( )

Combine two lists

[Example]

```
myList = ['a', 'b', 'c', 'd', 'e']
```

```
myList2 = [1, 2, 3, 4]
myList.extend(myList2)
print (myList)
=> ['a', 'b', 'c', 'd', 'e', 1, 2, 3, 4]
```

in

Check if an item is in a list

[Example]

```
myList = ['a', 'b', 'c', 'd']
'c' in myList
=> True

'e' in myList
=> False
```

insert( )

Add item to a list at a particular position

[Example]

```
myList = ['a', 'b', 'c', 'd', 'e']
myList.insert(1, 'Hi')
print (myList)
=> ['a', 'Hi', 'b', 'c', 'd', 'e']
```

len( )

Find the number of items in a list

[Example]

```
myList = ['a', 'b', 'c', 'd']
print (len(myList))
=> 4
```

pop( )

Get the value of an item and remove it from the list
Requires index of item as the argument

[Example]

```
myList = ['a', 'b', 'c', 'd', 'e']

#remove the third item
member = myList.pop(2)
print (member)
```

```
=> c

print (myList)
=> ['a', 'b', 'd', 'e']

#remove the last item
member = myList.pop( )
print (member)
=> e

print (myList)
=> ['a', 'b', 'd']
```

remove( )

Remove an item from a list. Requires the value of the item as the argument.

[Example]

```
myList = ['a', 'b', 'c', 'd', 'e']

#remove the item 'c'
myList.remove('c')
print (myList)
=> ['a', 'b', 'd', 'e']
```

reverse()

Reverse the items in a list

[Example]

```
myList = [1, 2, 3, 4]
myList.reverse()
print (myList)
=> [4, 3, 2, 1]
```

sort()

Sort a list alphabetically or numerically

[Example]

```
myList = [3, 0, -1, 4, 6]
myList.sort()
print(myList)
=> [-1, 0, 3, 4, 6]
```

sorted()

Returns a new sorted list without sorting the original list.

Requires a list as the argument

[Example]

```
myList = [3, 0, -1, 4, 6]
myList2 = sorted(myList)

#Original list is not sorted
print (myList)
=> [3, 0, -1, 4, 6]

#New list is sorted
print (myList2)
=> [-1, 0, 3, 4, 6]
```

Addition Operator: +

Concatenate List

[Example]

```
myList = ['a', 'b', 'c', 'd']
print (myList + ['e', 'f'])
=> ['a', 'b', 'c', 'd', 'e', 'f']

print (myList)
=> ['a', 'b', 'c', 'd']
```

Multiplication Operator: *

Duplicate a list and concatenate it to the end of the list

[Example]

```
myList = ['a', 'b', 'c', 'd']
print (myList*3)
=> ['a', 'b', 'c', 'd', 'a', 'b', 'c', 'd', 'a', 'b', 'c', 'd']

print (myList)
=> ['a', 'b', 'c', 'd']
```

Note:
The + and * symbols do not modify the list. The list stays as ['a', 'b', 'c', 'd'] in both cases.

Appendix C: Working With Tuples

=> marks the start of the output

del

Delete the entire tuple

[Example]

```
myTuple = ('a', 'b', 'c', 'd')
del myTuple
print (myTuple)
=> NameError: name 'myTuple' is not defined
```

in

Check if an item is in a tuple

[Example]

```
myTuple = ('a', 'b', 'c', 'd')
'c' in myTuple
=> True

'e' in myTuple
=> False
```

len( )

Find the number of items in a tuple

[Example]

```
myTuple = ('a', 'b', 'c', 'd')
print (len(myTuple))
=> 4
```

Addition Operator: +

Concatenate Tuples

[Example]

```
myTuple = ('a', 'b', 'c', 'd')
print (myTuple + ('e', 'f'))
=> ('a', 'b', 'c', 'd', 'e', 'f')

print (myTuple)
=> ('a', 'b', 'c', 'd')
```

Multiplication Operator: *

Duplicate a tuple and concatenate it to the end of the tuple

[Example]

```
myTuple = ('a', 'b', 'c', 'd')
print(myTuple*3)
=> ('a', 'b', 'c', 'd', 'a', 'b', 'c', 'd', 'a', 'b', 'c', 'd')

print (myTuple)
=> ('a', 'b', 'c', 'd')
```

Note: The + and * symbols do not modify the tuple. The tuple stays as ['a', 'b', 'c', 'd'] in both cases.

Appendix D: Working With Dictionaries

=> marks the start of the output

clear( )

Removes all elements of the dictionary, returning an empty dictionary

[Example]

```
dic1 = {1: 'one', 2: 'two'}
print (dic1)
=> {1: 'one', 2: 'two'}

dic1.clear()
print (dic1)
=> { }
```

del

Deletes the entire dictionary

[Example]

```
dic1 = {1: 'one', 2: 'two'}
del dic1
print (dic1)
=> NameError: name 'dic1' is not defined
```

get( )

Returns a value for the given key.
If the key is not found, it'll return the keyword `None`.
Alternatively, you can state the value to return if the key is not found.

[Example]

```
dic1 = {1: 'one', 2: 'two'}
dic1.get(1)
=> 'one'

dic1.get(5)
=> None

dic1.get(5, "Not Found")
=> 'Not Found'
```

in

Checks if an item is in a dictionary

[Example]

```
dic1 = {1: 'one', 2: 'two'}

# based on the key
1 in dic1
=> True

3 in dic1
=> False

# based on the value
'one' in dic1.values()
=> True

'three' in dic1.values()
=> False
```

items( )

Returns a list of dictionary's pairs as tuples

[Example]

```
dic1 = {1: 'one', 2: 'two'}
dic1.items()
=> dict_items([(1, 'one'), (2, 'two')])
```

keys( )

Returns list of the dictionary's keys

[Example]

```
dic1 = {1: 'one', 2: 'two'}
dic1.keys()
=> dict_keys([1, 2])
```

len( )

Find the number of items in a dictionary

[Example]

```
dic1 = {1: 'one', 2: 'two'}
print (len(dic1))
=> 2
```

update( )

Adds one dictionary's key-values pairs to another. Duplicates are removed.

[Example]

```
dic1 = {1: 'one', 2: 'two'}
dic2 = {1: 'one', 3: 'three'}

dic1.update(dic2)
print (dic1)
=> {1: 'one', 2: 'two', 3: 'three'}

print (dic2)  #no change
=> {1: 'one', 3: 'three'}
```

values( )

Returns list of the dictionary's values

[Example]

```
dic1 = {1: 'one', 2: 'two'}
dic1.values()
=> dict_values(['one', 'two'])
```

Appendix E: Project Answers

Exercise 1.1

```
def printInstructions(instruction):
  print(instruction)
```

Exercise 1.2

```
def getUserScore(userName):
  try:
    input = open('userScores.txt', 'r')
    for line in input:
      content = line.split(', ')
      if content[0] == userName:
        input.close()
        return content[1]
    input.close()
    return '-1'
  except IOError:
    print("File not found. A new file will be created.")
    input = open('userScores.txt', 'w')
    input.close()
    return '-1'
```

Exercise 1.3

```
def updateUserScore(newUser, userName, score):
  from os import remove, rename

  if newUser == True:
    input = open('userScores.txt', 'a')
    input.write(userName + ', ' + score + '\n')
    input.close()
  else:
    temp = open('userScores.tmp', 'w')
    input = open('userScores.txt', 'r')
    for line in input:
      content = line.split(', ')
      if content[0] == userName:
        temp.write(userName + ', ' + score + '\n')
      else:
        temp.write(line)

    input.close()
    temp.close()

    remove('userScores.txt')
    rename('userScores.tmp', 'userScores.txt')
```

Exercise 2.1

```
class Game:
  def __init__(self, noOfQuestions = 0):
    self._noOfQuestions = noOfQuestions

  @property
  def noOfQuestions(self):
    return self._noOfQuestions

  @noOfQuestions.setter
  def noOfQuestions(self, value):
    if value < 1:
      self._noOfQuestions = 1
      print("\nMinimum Number of Questions = 1")
      print("Hence, number of questions will be set to 1")
    elif value > 10:
      self._noOfQuestions = 10
      print("\nMaximum Number of Questions = 10")
      print("Hence, number of questions will be set to 10")
    else:
      self._noOfQuestions = value
```

Exercise 2.2

```
class BinaryGame(Game):
  def generateQuestions(self):
    from random import randint
    score = 0

    for i in range(self.noOfQuestions):
      base10 = randint(1, 100)
      userResult = input("\nPlease convert %d to binary: " %(base10))
      while True:
        try:
          answer = int(userResult, base = 2)
          if answer == base10:
            print("Correct Answer!")
            score = score + 1
            break
          else:
            print("Wrong answer. The correct answer is
{:b}.".format(base10))
            break
        except:
          print("You did not enter a binary number. Please try again.")
          userResult = input("\nPlease convert %d to binary: " %(base10))

    return score
```

Exercise 2.3

```
class MathGame(Game):
  def generateQuestions(self):
    from random import randint
    score = 0
    numberList = [0, 0, 0, 0, 0]
    symbolList = ['', '', '', '']
    operatorDict = {1:' + ', 2:' - ', 3:'*', 4:'**'}

    for i in range(self.noOfQuestions):
      for index in range(0, 5):
        numberList[index] = randint(1, 9)
      #refer to explanation below
      for index in range(0, 4):
        if index > 0 and symbolList[index - 1] == '**':
          symbolList[index] = operatorDict[randint(1, 3)]
        else:
          symbolList[index] = operatorDict[randint(1, 4)]

      questionString = str(numberList[0])

      for index in range(0, 4):
        questionString = questionString + symbolList[index] +
str(numberList[index+1])

      result = eval(questionString)

      questionString = questionString.replace("**", "^")

      userResult = input("\nPlease evaluate %s: "%(questionString))

      while True:
        try:
          answer = int(userResult)
          if answer == result:
            print("Correct Answer!")
            score = score + 1
            break
          else:
            print("Wrong answer. The correct answer is
{:d}.".format(result))
            break
        except:
          print("You did not enter a valid number. Please try again.")
          userResult = input("\nPlease evaluate %s: "%(questionString))

    return score
```

```
'''
Explanation

Starting from the second item (i.e. index = 1) in symbolList, the line if
index > 0 and symbolList[index-1] == '**': checks if the previous item in
symbolList is the ** symbol.

If it is, the statement symbolList[index] = operatorDict[randint(1, 3)]
will execute. In this case, the range given to the randint function is from
1 to 3. Hence, the ** symbol, which has a key of 4 in operatorDict  will
NOT be  assigned to symbolList[index].

On the other hand, if it is not, the statement symbolList[index] =
operatorDict[randint(1, 4)] will execute. Since the range given to the
randint function is 1 to 4, the numbers 1, 2, 3 or 4 will be generated.
Hence, the symbols +, -, * or ** will be assigned to symbolList[index].

'''
```

Exercise 3.1

```
from gametasks import printInstructions, getUserScore, updateUserScore
from gameclasses import Game, MathGame, BinaryGame
```

Exercise 3.2

```
try:
  mathInstructions = '''
In this game, you will be given a simple arithmetic question.
Each correct answer gives you one mark.
No mark is deducted for wrong answers.
'''

  binaryInstructions = '''
In this game, you will be given a number in base 10.
Your task is to convert this number to base 2.
Each correct answer gives you one mark.
No mark is deducted for wrong answers.
'''
  mg = MathGame()
  bg = BinaryGame()

  userName = input("\nPlease enter your username: ")

  score = int(getUserScore(userName))

  if score == -1:
    newUser = True
    score = 0
  else:
    newUser = False
```

```
  print("\nHello %s, welcome to the game." %(userName))
  print("Your current score is %d." %(score))

  userChoice = 0

  while userChoice != '-1':
    game = input("\nMath Game (1) or Binary Game (2)?: ")
    while game != '1' and game != '2':
      print("You did not enter a valid choice. Please try again.")
      game = input("\nMath Game (1) or Binary Game (2)?: ")

    numPrompt = input("\nHow many questions do you want per game (1 to
10)?: ")
    while True:
      try:
        num = int(numPrompt)
        break
      except:
        print("You did not enter a valid number. Please try again.")
        numPrompt = input("\nHow many questions do you want per game (1 to
10)?: ")


    if game == '1':
      mg.noOfQuestions = num
      printInstructions(mathInstructions)
      score = score + mg.generateQuestions()
    else:
      bg.noOfQuestions = num
      printInstructions(binaryInstructions)
      score = score + bg.generateQuestions()

    print("\nYour current score is %d." %(score))

    userChoice = input("\nPress Enter to continue or -1 to end: ")

  updateUserScore(newUser, userName, str(score))
```

Exercise 3.3

```
except Exception as e:
  print("An unknown error occurred. Program will exit.")
  print("Error: ", e)
```