Evaluating and creating expressions and simple statements

This section enables you to become familiar with talking about, recognising and interpreting expressions and simple statements, as well as writing them to meet a given specification. Imagine trying to read or write a book, but not being fluent with phrases and sentences - it would be impossible. It's the same here - you need to become fluent at the phrase/sentence level of the programming language, before working on larger programs.

Do your working and write your answers on separate paper - then you can use this again to practise in a few days time.

Section 1 - Describing lines of code - inc. variables, expressions, while/for, lists

In this section, you are asked to write down an explanation for each line in a short program. This checks whether you have the right language to talk about programs, which will help you immeasurably when you are thinking about your programs, and when you are talking to others about them.

1.1. For each line in the following program, **write down on your own paper what it is called, and exactly what it will do as it is executed.** Use the correct terminology for the components you refer to in the line. An explanation of the first line is given, as an example. **Do not try to execute the whole program - we'll do that later.**

i. `a = 100`
Assignment statement. The variable a is assigned the integer value 100.

ii. `b = [ 1, 6, 4, 8, 3, 4, 7, 9 ]`
iii. `le = len( b )`
iv. `for i in range( le ):`
v. `    print( a, end = " " )`
vi. `    a = a - b[ i ]`
vii. `print( a )`

1.2. Similar complexity to previous - using strings this time:

i. `b = "A string that I just created."`
ii. `le = len( b ) // 2`
iii. `for i in range( le ):`
iv. `    print( b[ i ], end = " " )`
v. `    print( b[ -1 -i ] )`

1.3. Same again, also with strings:

```
i.   inputString = input( "Please enter string:" )
ii.  newString = ""
iii. index = len( inputString )
iv.  while index > 0:
v.       index -= 1
vi.      newString += inputString[ index ]
vii. print( newString )
```

1.4. And again - more complex this time:

```
i.   b = [ 3, 6, 10, 7, 19, 24, 20, 31, 38, 44 ]
ii.  u = b[ 0 ] < b[ 1 ]
iii. p = b[ 1 ]
iv.  for i in range( 2, len( b ) ):
v.       if u and p > b[ i ]:
vi.          print( "x" + str( i-1 ), end = " " )
vii.         u = not u
viii.    elif not u and p < b[ i ]:
ix.          print( "y" + str( i-1 ), end = " " )
x.        u = not u
xi.       p = b[ i ]
```

Section 2: Evaluating expressions

2.1. Write down precisely what would be output when each of the following fragments of Python code is executed. If you think that any code fragment would result in an error, explain clearly the cause of the error.

You may assume the code immediately below, to declare variables, has been executed.

```
a=7
b=10
c=3
x = [ "D", "E", "F", "G", "H" ]
n = { “fred” : 23, “bob” : 100 }
s = "computing"
```

i.
```
print( a + b * c )
```

ii.
```
print( x[ 1:2 ] + "end" )
```

iii.
```
print( a > 6 OR b - c < 7 )
```

iv.
```
print( a // 3 )
```

v.
```
print( 3 % 7 )
```

vi.
```
print( s[ 2:4 ] > "mo" )
```

vii.
```
print( n[ fred ] * 2 )
```

2.2.
Same guidance as for Q1.

```
a=5
b=14
c=2
x = [ 10, 3, 7, 2, 8, 12 ]
y = { “fred” : 23, “bob” : 100 }
s = "large"
```

i.
```
print( 5 % 2 + 14 // 5 )
```

ii.
```
print( x[ 2:4 ] )
```

iii.
```
print( a < 4 OR 7 < a + c )
```

iv.
```
print( x[ 4 ] / 2 )
```

v. `print( y[ "fred" ] + s )`

vi. `print( "fred" > s )`

vii. `print( a + b // c - x[ y[ "bob" ] - 97 ] - len( s ) )`

2.3. Write down precisely what would be output when each of the following fragments of Python code is executed.

i. `print( int(1977 + 1.1) )`

ii. `print( float(1980 + int(4.21)) )`

iii. `print( len(str(float(1.977))) )`

iv. `print( (len("TK")+4+2+int("1"))/2 )`

v. `print( int(float(1983)*1.0) )`

vi. `print( float(len("falcon"))*len("94") )`

vii. `print( str(1977)[3]*len(str(1980)) )`

viii. `print( float(str(1977)[3])*len(str(80)) )`

ix. `print( max("1977") * 3 )`

2.4. Evaluate the following expressions:

i. `"darkside" < "lightside"`

ii. `True != True`

iii. `True != False and True`

iv. `4 < 10 or 1 < 2`

v. `False or (1 < 2 and 4 > 2)`

Section 3: Writing expressions or statements

Write an expression or a statement to satisfy each of the following:

3.1 Given a string named `s`, and an integer `n` which is one of the indices of `s`, write an expression that returns a string the same as `s` but with the character at position `n` removed. E.g. if `s` is `"hello"` and `n` is `1`, then `"hllo"` should be the value of the expression.

3.2 Write the correct `range` expression to give the numbers from 1 to 100.

3.3 Write the correct `range` expression to give all the indices of an array `a`, such as you might use in a for loop to access all elements of the array.

3.4 Given a string `s` of unknown length, write out the individual characters:

i. Each on a separate line

ii. On a single line, but with each character separated by a single space character

3.5 Write a Boolean expression to determine whether the integer variable `v` holds a value that could safely index a value in list `li`.

3.6 Write an expression to create a list containing 20 elements, all 0. Make it as short as you can.

3.7 Write string expressions working on the string `s` to:

i. Extract the last character of `s`.
ii. Extract the substring consisting of the last two characters of `s`.

iii. Create a new string with the middle character of `s` removed. If the length of `s` is even, so there is no middle character, take out the character just to the right of the middle.

iv. Create a new string consisting of the first and last characters of `s` only.

v. Create a new string without the last character of `s`.

3.8 Assuming you have an integer list referred to by a variable named `lis`, write statements as directed. Each answer may require more than one statement. You can assume the list has at least 5 elements.

i. Write an assignment statement to add `1` to the last element of `lis`.

ii. Write statements to add up the values of all the list elements in list `lis` after the first one, and then add this value to the value in the first element.

iii. Update `lis` to refer to a list containing all but the last value in the original list.

iv. Update `lis` to refer to a list containing only the first two, and the last two, elements of the list it is currently referring to.

Section 4 - Doing all of the same but adding in dictionaries and files

4.1. Back to explaining what lines of code mean - write a line for each numbered line of code below - what kind of statement is each of these lines, and what does the line do?

```
    # Assume that someData.txt is a text file
i.  myFile = open( "someData.txt" )
ii. lines = myFile.readlines()
iii. for i in range( len( lines ) ):
iv.     print( str( i + 1 ), end = " " )
v.      print( lines[ i ] )
```

4.2. And another example:

```
    # Assume file someData.txt containing comma-separated values
i.  myFile = open( "someData.txt" )
ii. lines = myFile.readlines()
iii.
iv. for line in lines:
v.      line = line[ :-1 ]  # Explain WHY this line is being used
vi.     items = line.split( "," )
vii.    for piece in items:
viii.       print( piece, end = " " )
ix.     print( "" )
```

4.3. Now something more like a database file. Assume that the `people.txt` file contains lines containing a string and an integer, representing a name and an age, separated by a comma:

```
i.  peopleFile = open( "people.txt" )
ii. fileLines = peopleFile.readlines()
iii. peopleTable = []

iv. for fullLine in fileLines:
v.      personData = ( fullLine[ :-1 ] ).split( "," )
vi.     newPerson = { "name": personData[ 0 ],
vii.                  "age" : int( personData[ 1 ] ) }
viii.   peopleTable = peopleTable + [ newPerson ]
```

ix.
```
for p in peopleTable:
```
x.
```
 if p[ "age" ] >= 17:
```
xi.
```
     print( p[ "name" ] )
```

4.4. Write an expression or statement to satisfy each of the following

i. You have a list of dictionaries, dictList. Write code to retrieve the value stored with the key “Quintin” of the dictionary that is stored at index 7 of the list.
ii. Add a new dictionary to the end of the list above, which contains only one entry - associating key “Jeremy” to the integer value 128.
iii. A file has been opened and assigned to variable myFile. Write code to print out all the lines in the file to the screen
iv. Variable d holds a dictionary. Each entry in the dictionary has a string key associated with a list of integers. Write code to extend or add an entry with the key “CS” - if the entry already exists, then add the value 7 to the end of the associated list - if the entry does not exist, create it with a new list containing just the value 7.