Python Programming Handbook

This is an informal introduction to Python programming. It introduces the beginner to some fundamental programming concepts such as: input/output, variables and conditional statements.

Overview
One point about programming must be clarified immediately: **Anyone can learn to program computers.** However, you must be willing to **spend some time studying and practising**.

There are two aspects to programming that must be mastered. One concerns **problem solving** and the other concerns the **programming language, in this case Python,** that is to be used.

You must learn how to solve problems. This is the core of programming. But you also must learn how to express your problem solution in Python, so that it can be carried out on a computer. These are two separate skills. You must try not to confuse them. It is difficult, however, to explain one without reference to the other.

Problem Solving Skills
Computer programming is about problem solving. Every computer program solves some particular problem, even programs to play games. It is impossible to write a computer program unless you understand the problem you are being asked to solve. In programming, **you solve** the problem, not the computer. A computer program describes to the computer **what** it must do and **how** it is to be done. You give the computer instructions in the form of a program, telling it what to do and how to do it. **The set of instructions required to solve a problem is a computer program**

The term **algorithm** is used to describe the **set of steps that solve a problem**. An algorithm may be expressed in English or in a programming language. A **computer program must be expressed in a programming language.** In programming, we first develop an algorithm for the problem at hand, and then we ***translate*** this algorithm to a Python program, so that it can be executed (carried out, ran) on a computer.

Sometimes we make mistakes in telling the computer what to do. We overlook part of the problem or do not understand what to do ourselves. In these cases, the computer program will not produce the 'right answer'. It is, however, still solving a problem. It is simply not the problem we wanted to solve. For this reason, it is important to thoroughly check that your programs do indeed solve the problem you intended.

Lesson 1: Output

In this lesson we learn how to perform output – to display messages on your screen.

All program code will be displayed using the `Courier font` in this text.

Output is the term used to describe information that the computer displays (**writes**) on your screen or stores (writes) on a disk drive.

The commonest form of output is that of displaying a message on your screen. In Python, we use the `print` statement to display output on the screen.

The following `print` statement will display the message:

```
My name is Beth. This is my first program
```

on the screen.

```
print( 'My name is Beth. This is my first program' )
```

This is a single Python program **statement**. A statement is a command to the computer to carry out an action. So our first Python program is composed of this single statement. To execute this program, it is stored in a file which we call `print1.py.` which contains one line:

```
print( 'My name is Beth. This is my first program' )
```

To execute the program we use the command python3 which runs the program:

```
% python3 print1.py
My name is Beth. This is my first program
```

You may also run Python programs using an IDE which will be discussed later.

We call the message that `print` displayed a **string**.

A string is a list of characters in quotes. You may put any characters that you wish in a string. Strings can be represented by characters inside either single or double quotes:

```
'My name is Beth.'
"My name is Beth."
```

You can display strings using a `print`:

```
print( 'Hello ! Goodbye!' )
print( 'rubbish 123 rubbish xyz @£$%^&*' )
```

Statements

When people communicate in any language, they use **sentences**. When you write down a sentence in English, you have a full stop at the end. This tells us where the sentence ends. You call the full stop a sentence **terminator** i.e. it indicates the end of a sentence.

Similarly, when you write programs you also use sentences to communicate with the computer. In programs, sentences are called **statements**. They also have a terminator. Python statements use the **end of line** to terminate a statement. The 2 statements below are terminated by the **newline character**.

```
print( 'Hello ! Goodbye!' )
print( 'rubbish 123 rubbish xyz @£$%^&*' )
```

After you entered **)** on the first line, you pressed the Return key on the keyboard to start a new line – we call this the **newline character**. Similarly after you entered **)** on the second line, you also pressed Return thus starting a new line.

If you make a mistake entering a statement, a computer will not understand the statement and it will display an error message. This often caused by a **spelling error** or a **missing bracket** or **quotation** mark. For example, in a programming, just as in English, you must always have the correct number of quotation marks and brackets.

Left brackets such as (, { and [are called **opening** brackets.

The quotation marks at the start of a phrase: **"** (double) and **'** (single) are called **opening** quotes.

Right brackets such as) , } and] are called **closing** brackets and quotation marks at the end of a phrase are called **closing** quotes.

*A simple rule is that **for every opening** bracket or quotation mark you must have a **corresponding closing** bracket or quotation mark.*

Example 1.1: Matching brackets and quotes:

```
'Hello'
( age  > 10 )
'x'
list[10]
```

Breaking this rule causes a **syntax error**. The following are examples of syntax errors:

```
age > 10)
'x
list[10
prin( 'Hello ! Goodbye!' )
```

The last error is due to misspelling `print`

What to do when an error occurs
When a syntax error occurs, you must work out what mistake(s) you have made. This means checking the statements of your program and seeing where the syntax is incorrect. You then **edit** your program to correct the mistake. When you have corrected your program, it can be executed. **To execute a program or to run a program means that the computer carries out the instructions making up the program.**

A Python program is made up of a group of one or more statements. These statements allow us to control the computer. Using them, we can **display information on the screen**, **read information from the keyboard** and **process information** in a variety of ways.

We can classify statements as:

Input/Output (I/O) statements (e.g. display information on screen)
Variable manipulation statements (e.g. to do arithmetic) and
Conditional statements (e.g. to make decisions)

Lesson 1 Exercises – try these in your own time

1. What is the output of the following print statement?

   ```
   print( 'Have a great day!')
   ```

   a. 'Have a great day!'

   b. 'Have a great day'

   c. Have a great day!

   d. 'Have a great day!'

2. What is the output of the following statements?
   ```
   print('Hi there!')
   print('How are you doing?')
   ```
   a. Hi there! How are you doing?
   b. How are you doing? Hi there!
   c. 'Hi there!'
      How are you doing?'
   d. Hi there!
      How are you doing?

3. Write a program that prints a message saying

   I love Python!

4. Write a program that prints a message saying your name and your age, e.g.

   My name is Colin. I am 20 years old!

5. Write a program to display the message 'Welcome to Python' three times, on separate lines using three `print` statements.

6. Write a program to display the message 'Python is awesome!' two times, on separate lines, using only one `print` statement and `\n`

7. What are the syntax errors in the following statements:

```
print( 'Hello ! Goodbye! )
print( Hello ! Goodbye!' )
print( 'Hello ! Goodbye!'
print 'Hello ! Goodbye!' )
prlnt( 'Hello ! Goodbye!' )
```

Lesson 1 Assignments – you email these to UCD

1. Write a program that prints out your favourite food, followed by a blank line, followed by your favourite colour, followed by a blank line, followed by your favourite animal. Use either the `\n` character in one `print` statement or use separate `print` statements.

2. Print a square made up of 4 @ characters per line using a single `print` statement.

```
@@@@
@@@@
@@@@
@@@@
```

Lesson 2: Input and Variables

Input is the term used to describe the transfer of information from the keyboard (or a disk) to the computer. We can use the word **read** for input e.g read information from the keyboard. The question arises – where do we store the information we read in. This introduces the concept of **variables**.

A variable is a container for information.

This means that we can store information in a variable. It is called a variable because at any time we can change (**vary**) the information it stores.

So when we input information, we store it in a one or more variables. We **give each variable a unique name**, which we use to identify it. The following are examples of variable names we could use in a Python program:

```
colour
my_age
pension_age
name
taxcode
tax_rate
temperature
name
hourly_pay
```

Fundamental principle of writing clear programs

Choose meaningful names for variables

Using meaningful variable names it makes your programs easier to understand. For example, if you are writing a program which deals with pension ages then you could use any of the following variable names to store the pension age but which one makes is easiest to understand:

```
pension_age
pa
p
x
pna
```

The variable name `pension_age` is the obvious choice. When you see this name you automatically know what it the variable is being used for. If you use a name like `p` or `x` then the name gives you no idea about what the variable is being used for.

We use the Python **`input`** statement to read information from the keyboard, into a variable.

Example 1: Write a program to ask the user to enter their favourite colour. The program reads this colour that the user types on the keyboard and displays a message followed by the colour entered by the user.

```
favourite_colour = input('Enter your favourite colour: ')

print( favourite_colour )
```

If we execute the program the following appears on the screen (the bolded text is that entered by the user on the keyboard. We will use this convention throughout the text).

```
Enter your favourite colour: blue
blue
```

The variable `favourite_colour` is used to store the characters that the user types on the keyboard that is, it will store a list of characters.

When you use a variable name with `print` it will display the **value** of the variable i.e. the string `blue` in the example above.

The `input()` statement does two tasks: it displays the string in quotes and then reads text from the keyboard, (for example the word *blue* may be entered), and it places this text in the variable `favourite_colour`.

We have seen that the `print` statement is used to display output on the screen. It can be used to display strings and numbers in the same `print` statement.

Example 2: Write a program to ask the user to enter their favourite colour. The program reads this colour that the user types on the keyboard and displays a message followed by the colour entered by the user.

```
favourite_colour = input('Enter your favourite colour: ')

print('Yuk ! I hate ', favourite_colour )
```

If we execute the program the following appears on the screen:

```
Enter your favourite colour: blue
Yuk ! I hate blue
```

The statement
```
print('Yuk ! I hate ', favourite_colour )
```

instructs the computer to display the message *Yuk ! I hate* followed by the value of the variable `favourite_colour` i.e. *blue* in this example.

We use the expression 'the value of a variable' to mean 'the value contained in a variable'.

We take the phrase 'the value of `favourite_colour` is `blue`' to mean 'the value contained in the variable `favourite_colour is` blue'. We will use the shorter form from now on.

Make sure you understand the difference between:

```
print( 'favourite_colour' )
```
and
```
print( favourite_colour )
```

In the first case, the word (string) `favourite_colour` appears on the screen.

In the second case, the **value** of a variable called `favourite_colour is` displayed which could be anything, for example the word *blue* or whatever value the user has given the variable like *red*, *pink* and *orange*. You can store **many words** in a string variable.

Rules for Variable Names

Python and every programming language, has rules on how you name variables:

- **A variable name can only contain the following**:
  - **letters** (lowercase and uppercase, ie a–z and A–Z)
  - **digits** (0–9)
  - the underscore character '_'

- A variable name **cannot start** with a digit

- Variable names in Python are **case-sensitive –** it distinguishes between uppercase and lower case letters so that `colour` and `Colour` are **different variables**

- There are a number of **reserved words** or **keywords** that have built-in meanings in Python and cannot be used as variable names (e.g. if, `return, def, del, break, for, in, else, while, import`)

The following are legal or valid variable names in Python:

```
Colour, name, firstname, surname, class1, class 602,
first_name, second_name, address_line1.
```

The use of the underscore '_' character is very useful in creating meaningful names made up of 2 or more words.

Do not confuse the underscore character with the minus sign '-'. The minus sign (or any other symbol) cannot be used in a variable name. thus `first-name` is not a valid variable name.

Comments

In a Python program, any text after **#** is called a comment and **is ignored by Python**. This text is there to help explain to someone reading the program, what the program does and how the program works. **Comments are an important component of programs**. This is because when you read your programs some time after writing them, you may find them difficult to understand, if you have not included comments to explain what you were doing. They are even more important if someone else will have to read your programs e.g. your tutor who is going to grade them!

It is a useful idea to start all programs with comments which give the name of the file containing the program, the purpose of the program, the authors name and the date on which the program was written, as the first comments in any program. Example 2 could be written as:

```
# colour.py: Prompt user to enter colour and display a message
# Author: Joe Carthy
# Date: Oct 20 2022

favourite_colour = input('Enter your favourite colour: ')

print('Yuk ! I hate ', favourite_colour )
```

Lesson 2 Exercises

1. Valid or invalid variable names
    a. Is the variable name `TotalMarks` correct?
    b. Is the variable name `number-of-students correct?`
    c. Is the variable name `firstName` correct?
    d. Is the variable name `myVar1` correct?
    e. Is the variable name `customerName` correct?
    f. Is the variable name `productPrice` correct?
    g. Is the variable name 3`rdStudent` correct?
    h. Is the variable name `isAvailable`? correct?
    i. Is the variable name `total-sales` correct?
    j. Is the variable name `customer_email` correct?

3. Write a program that asks the user for their name using `input`. Store the name in a variable and display a personalized greeting using the variable.

4. What is the output, if any, of the following program:

```
# print('hello\n')
# print('bye bye\n')
```

5. Write a program that prompts the user to enter their favourite colour and favourite animal using the `input`. Store these values in separate variables and display them in a sentence :

```
My favourite colour is … and my favourite animal is ..
```

Lesson 2 Assignments

i. Write a program that prompts the user to enter their favourite number using `input`. Store the number in a variable and print a message that includes the user's favourite number.

ii. Create a program that simulates a hospital registration system. Prompt the user to enter the following information:

Name
Surname
Age
Height (in cm)

Store each piece of information in a separate variable with an appropriate name. Finally, print the information in the following format:

Name: Joe
Surname: Carthy
Age: 100
Height: 180

Lesson 3: Variables and Assignment

In Lesson 2 we used `input` to give a variable its value.. Giving a variable a value is called **assignment**. We can use assignment to give a value to a variable directly in a program without input. We may give the variable a value or compute a value based on the values of other variables. For example, suppose we have a variable called `metres`, to which we wish to give the value `12`. In Python we write:

```
metres = 12
```

This is usually read as '**`metres` is assigned the value 12**'. Of course, we could use any value instead of `12`. Other examples of assigning values to variables are:

```
centimetres = 50
litres = 10
metres = 4
colour = 'red'
name = 'Joe Carthy'
pay_per_hour = 10.5
```

Example L3.1: Write a program to convert 5 metres to centimetres. A simple Python program to do this is given below.

```
#convert1.py: converts metres to centimetres
#Author: Joe Carthy
#Date: 21/10/2023

metres = 5
centimetres = metres * 100
print('5 m is ', centimetres, 'cms',)
```

Executing this program produces as output:

```
5 m is 500 cms
```

Here we use the value of the variable `metres` to compute the value of the variable `centimetres`.

Other examples of such an assignment are:

```
gallons = 4
pints = gallons * 8
kilometres = 4
metres = 18
cms = (kilometres * 100000) + (metres * 100)
```

The program to convert metres to centimetres as presented in Example L3.1 is very limited in that it always produces the same answer. It always converts 5 metres to centimetres. A

more useful version would prompt the user to enter the number of metres to be converted and display the result:

Example L3.2: Converting metres to centimetres, version 2.

```
#convert2.py: converts metres to centimetres version 2
#Author: Joe Carthy
#Date: 21/10/2023

m = input('Enter number of metres: ')

metres = float ( m )

centimetres = metres * 100

print(metres, 'metres is ', centimetres, 'cms',)
```

Executing this program produces as output:

```
Enter number of metres: 4
4.0 metres is 400.0 cms
```

Variable Types

Important note: The `input` function reads from the keyboard and returns a list of characters i.e. a string. Thus the variable m in the example above contains the string '4' and not the number 4.

This is very confusing for beginners to programming. A fundamental aspect of variables is that they have a **type**. The type of a variable tells you what kind of data it stores.

In our programs we will use three types: **int** (whole numbers), **float** (numbers with decimal point) and **string** (list of characters).

When you are working with numbers and wish to do arithmetic with them (add, subtract, multiply and divide) then you must use either the type **int** or **float**.

So it is crucial to understand the difference between the number 42 and the string '42' as used in the following:

```
a = 42
b = a * 2
```

This results in `b` having the value 84. A and b are of type **int** in this case.

```
x = '42'        # x is type string
y = x * 2       # y is type string
```

This results in `y` having the value `4242` – string of characters.

When you ***'multiply'*** a string variable by `n` you get `n` copies of the string e.g.

The code:

```
x = 'bye'
y = x * 3
```

gives `y` the string value `'byebyebye'`

This brings us back to Example L3.2 and the statements

```
m = input('Enter number of metres: ')

metres = float ( m )
```

The variable `m` is of type string.

The `float` function converts the string `m` to a number with a decimal point (real number).

This means that `metres` now contains a number which we can do arithmetic with.

The output of L3.2 is 'crowded' in that there is no blank line before or after the output or between the two lines of output. This makes it hard to read the output. You can use the '\n' character in strings to start new lines.

The version below fixes this issue by putting one '\n' in the `input`() function and 3 in the `print`() function.

It also uses a shortcut to avoid using a string variable, `m`, in the previous examples. It does this by converting the string from `input` to a float in one statement:

```
    metres = float (input('\nEnter number of metres: '))
```

Example L3.3: Converting metres to centimetres, version 4

```
# convert4.py: converts metres to centimetres version 3
# Outputs extra blank lines to make it easier to read the output

#Author: Joe Carthy
#Date: 21/10/2022

metres = float (input('\nEnter number of metres: '))

centimetres = metres * 100

print('\n', metres, 'metres is ', centimetres, ' centimetres\n\n'  )
```

When you run it, notice the extra blank lines

```
Enter number of metres: 3.5

3.5 metres is 350.0 centimetres
```

Some Fun making the computer beep!

When you use '\a' (called the BEL character) in `print`, the computer makes a beep sound – it does not display anything. So the program below simply plays 3 beeps.

```
# beep.py: Just for fun – beep 3 times !!

print('\a \a \a')
```

Example L3.4: As another example of the use of I/O and variables consider a simple calculator program. This program prompts for two numbers, adds them and displays the sum.

```
# calc.py: Calculator program to add 2 numbers
# Author: Joe Carthy
# Date: 01/10/2023

number1 = float(input('\nEnter first number: '))

number2 = float(input('\nEnter second number: '))

sum = number1 + number2

print('\n\nThe sum of', number1, 'and', number2, 'is', sum, '\n\n')
```

`calc.py` outputs:

```
Enter first number: 2.4

Enter second number: 5.76

The sum of 2.4 and 5.76 is 8.16
```

Variables must be defined before you use them other statements

Variables must be defined before you use them – you must give them a value.

If a variable is 'not defined' (not assigned a value), trying to use it will generate an error.

So if you run the 1 line program:

```
print ('x = ', x)
```

An error is displayed because the variable `x` has not been defined. The error message may not be very user friendly such as: that below – the last line is the helpful one:

```
Traceback (most recent call last):
File "<string>", line 1, in <module>
NameError: name 'x' is not defined
```

The most common reason for this error is mis-spelling the name of.variable as in the code below

```
metres = 5
cms = metres / 100
print (f'{metrs} = {cms} centimetres')
```

Here we have misspelled `metres` in the `print` statement and an error is displayed:

```
Traceback (most recent call last):
  File "<string>", line 3, in <module>
NameError: name 'metrs' is not defined
```

More on print() function and displaying variables

It can get quite complicated when we output strings and variables using `print` as in the statement

```
print('\n\nThe sum of', number1, 'and', number2, 'is', sum, '\n\n')
```

There is a simpler way to display this message with `print` using **f-strings**. We put the character **f** as the first item in `print`:

```
print(f'\n\nThe sum of {number1} and {number2} is {sum} \n\n')
```

which produces identical output to the earlier `print`

```
The sum of 2.4 and 5.76 is 8.16
```

When using an f-string, **we enclose any variable we wish to display in {}** brackets.

`print` will display the value of each variable in {}.

This avoids having to have separate strings in quotes, separated by commas, as in the earlier version of print.

As another example consider the following variables and how we want to display them:

Name = 'Joe Bloggs'
rate = 10.00
num_hours = 40
pay = rate * num_hours

Without using an f-string we display using:

```
print('Pay for ', name, 'at ', rate, 'per hour is', pay)
```

which outputs

```
Pay for Joe Bloggs at 10.0 per hour is 400.0
```

We can display the same output with a simpler `print` using **f-strings**:

```
print(f'Pay for {name} at {rate} per hour is {pay}')
```

displays the same output as the first `print()` above.

```
Pay for Joe Bloggs at 10.00 per hour is 400.00
```

Displaying a fixed number of decimal places

Python will display the result of numeric calculation to many decimal places.

For example,

```
x = 19/3.768

print(f'x = {x}' )

will output on my Mac computer:

x = 5.042462845010616
```

In most of our calculations it is enough to display result with 2 decimal places.

We use an f-string to do this by following the variable in {} with **:*.number of decimal points*f** you wish to output e.g. {x:.2f} specifies to display x to 2 decimal places.

You can change the number from 2 to whatever you wish, to have that number of places displayed after the decimal point.

To print `x` to 2 decimal points

```
x = 19/3.768

print(f'x = {x:.2f}')
```

outputs

```
x = 5.04
```

Lesson 3 Exercises

1. What is the data type of each variable?
    a. What is the data type of the variable 'age'? `age = 25`
    b. What is the data type of the variable 'name'? `name = 'John Doe'`
    c. What is the data type of the variable 'price'? `price = 9.99`
    d. What is the data type of the variable 'quantity'? `quantity = 10`
    e. What is the data type of the variable 'message'? `message = 'Hello'`
    f. What is the data type of the variable 'discount'? `discount = 0.2`

2. Write a program to convert 10 dollars to kyats using an exchange rate of 1 dollar = 2100 kyats.

```
10 dollars = 21000 kyats
```

Use `print` with f-strings in all of the following programs

3. Write a program that takes a single length (a float) and calculates the following:
    - The area of a square with side of that length. (length * length)
    - The volume of a cube with side of that length. (length ** 3)
    - The area of a circle with diameter of that length (3.14 * (length/2)**2))

```
Enter length: 4

Area of square:     16.0
Volume of cube:     64.0
Area of circle:          12.56
```

4. Write a program that takes an amount (a float), and calculates the tax due according to a tax rate of 20%

```
Enter amount for tax at 20%: 200.0

Tax: 40.00
```

5. Write a program to simulate a cash register for a single purchase. The program reads the unit cost of an item and the numbers of items purchased. The program displays the total cost for that number of units:

```
Enter unit cost: 5
Enter number of units: 6

Total cost of 6 units: 30.00
```

6. Modify the programs 3, 4 and 5 above to display the output to one decimal point e.g.

```
Area of circle:          12.5
```

Lesson 3 Assignments

Use `print` with f-strings in all of the following programs

1. Ask the user to enter a temperature in Celsius and convert it to Fahrenheit using the formula:

   Fahrenheit = (Celsius * 1.8) + 32.

   Print the converted temperature in Fahrenheit.

```
Enter temperature in Celsius: 100

100 degrees Celsius = 212.0 degrees Fahrenheit
```

2. Convert dollars to kyat as follows:
   a. Display 'Dollar to Kyat conversion program'
   b. Ask the user to enter an amount in dollars.
   c. Ask the user to enter the kyat exchange rate for dollars.
   d. Calculate kyat amount by multiplying the dollar amount by the exchange rate.
   e. Print out the dollar and kyat amounts

```
Dollar to Kyat conversion program
Enter amount in dollars: 10
Enter dollar to kyat exchange rate: 2100
10 dollars = 21000 kyats
```

3. Write a program to calculate how much someone gets paid per week based on the number of hours they work per week. The program asks the user to enter the number of hours worked and the rate per hour and then displays the total pay, with a blank line between each line of output:

```
Enter number of hours worked: 20.5

Enter rate per hour: 10

Total pay = 205.0
```

4. Write a program to display your total savings for 3 weeks based on saving $10 in Week 1, $15 in Week 2, and $20 in Week 3. Use 3 variables, one for each week's savings and one the total amount saved. Then calculate the total amount of money saved over the three weeks by adding the 3 variables. Print the result as follows:

```
You saved a total of 45 dollars
Week 1 you saved 10 dollars
Week 2 you saved 10 dollars
Week 3 you saved 10 dollars
```

5. You sell 10 cups of lemonade at $2.50 each. Calculate the total amount of money you earned by multiplying the number of cups sold by the price per cup using 3 variables, one for the number of cups, one for the cost per cup and one for the total amount sold. Print the result as follows, with a blank line between each line of output:

```
Number of cups sold: 10

Price per cup: 2.50

Total sold: 25.00
```

Appendix 1: Solutions

Lesson 1 Solutions

1. What is the output of the following print statement?

```
print( 'Have a great day!')
```

c. Have a great day!

a. What is the output of the following statements?

```
print('Hi there!')
print('How are you doing?')
```

Hi there!
How are you doing?

b. Write a program that prints a message saying

I love Python!

```
print('I love Python! ')
```

c. Write a program that prints a message saying your name and your age, e.g.

My name is Colin. I am 20 years old!

```
print('My name is Colin. I am 20 years old! ')
```

d. Write a program to display the message 'Welcome to Python' three times, on separate lines using three `print` statements.

```
print('Welcome to Python! ')

print('Welcome to Python! ')

print('Welcome to Python! ')
```

e. Write a program to display the message 'Python is awesome!' two times, on separate lines, using only one `print` statement and \n

```
print('Python is awesome!\n Python is awesome!\n ')
```

f. What are the syntax errors in the following statements:

```
print( 'Hello ! Goodbye! )- missing closing '
print( Hello ! Goodbye!' ) – missing opening '
print( 'Hello ! Goodbye!'  - missing closing )
print 'Hello ! Goodbye!' ). – missing opening (
prlnt( 'Hello ! Goodbye!' ) – misspelt print
```

Lesson 2 Solutions

1. Valid or invalid variable names
a. Is the variable name `TotalMarks` correct - **Yes**
b. Is the variable name `number-of-students correct?` **NO** – cannot use –in variable name
c. Is the variable name `firstName` correct? **Yes**
d. Is the variable name `myVar1` correct? **Yes**
e. Is the variable name `customerName` correct? **Yes**
f. Is the variable name `productPrice` correct? **Yes**
g. Is the variable name 3`rdStudent` correct? **NO** – cannot start with a digit
h. Is the variable name `isAvailable`? correct? Yes
i. Is the variable name `total-sales` correct? **NO** – cannot use –in variable name
j. Is the variable name `customer_email` correct? **Yes**

2. Write a program that asks the user for their name using `input`. Store the name in a variable and display a personalized greeting using the variable.

```
name = input('Enter your name: ')

print('Hello, how are you ', name )
```

3. What is the output, if any, of the following program:

```
# print('hello\n')
# print('bye bye\n')
```

No output because any text following # is treated as a comment and ignored by Python

4. Write a program that prompts the user to enter their favourite colour and favourite animal using the `input`. Store these values in separate variables and display them in a sentence :

```
favourite_colour = input('Enter your favourite colour: ')
favourite_animal = input('Enter your favourite animal: ')


print('My favourite colour is ', favourite_colour, 'and my favourite animal is', favourite_animal )
```

Lesson 3 Solutions

Q1:

a. The data type of the variable 'age' is integer.

b. The data type of the variable 'name' is string.

c. The data type of the variable 'price' is float.

d. The data type of the variable 'is_valid' is boolean.

e. The data type of the variable 'quantity' is integer.

f. The data type of the variable 'message' is string.

g. The data type of the variable 'discount' is float.

2. Write a program to convert 10 dollars to kyats using an exchange rate of 1 dollar = 2100 kyats.

```
# convert dollars to kyats

dollars = 10
kyats = dollars * 2100
print(f '{dollars} dollars = {kyats} kyats ' )
```

8. Write a program that takes a single length (a float) and calculates the following:
    - The area of a square with side of that length. (length * length)
    - The volume of a cube with side of that length. (length ** 3)
    - The area of a circle with diameter of that length (3.14 * (length/2)**2))

```
# calculate area of square, volume of cube and area of circle

length = float(input('Enter length: '))

area_of_square  = length * length
cube_volume = length ** 3
area_of_circle = 3.14 * ((length / 2)**2)
print(f' Area of square: }area_of_square:.2f}' )
print(f' Volume of cube: {cube_volume:.2f}' )
print(f' Area of circle: {area_of_circle:.2f}' )
```

7. Write a program that takes an amount (a float), and calculates the tax due according to a tax rate of 20%

```
Enter amount for tax at 20%: 200.0

Tax: 40.00
```

```
# calculate tax due at 20%

amount = float(input('Enter amount for tax at 20%: '))

tax = amount * 0.20
print(f'Tax: {tax:.2f}' )
```

8. Write a program to simulate a cash register for a single purchase. The program reads the unit cost of an item and the numbers of items purchased. The program displays the total cost for that number of units:

```
Enter unit cost: 5
Enter number of units: 6

Total cost of 6 units: 30.00
```

```
# calculate total cost as number of unit * unit cost

unit_cost  = float(input('Enter unit cost: '))

number_units  = float(input('Enter number of units: '))

total = unit_cost * number_units

print(f'\nTotal cost of {number_units} units: {total:.2f}')
```