

# Python Programming

## Section 2 – Variables & Operations

### UEE60411 Advanced Diploma of Computer Systems Engineering

UEENEED103A

Evaluate and modify Object Oriented code programs

For further information:

Steve Gale

ICT – Science and Technology

Ph: (03) 5225 0940

Email: [sgale@gordontafe.edu.au](mailto:sgale@gordontafe.edu.au)

# 1 – Variables

A **variable** is an area of the computer's memory which holds data. As the program executes (runs), the variable's contents can change (hence the name). The data may be in many different forms, such as:

- a string of characters (**string**)
- a whole number (**integer**)
- a decimal number (**float**)
- or even a collection of other variables (**list**, called an “array” in many other languages).

There are some other types we'll learn about as the course continues, but these will do for the moment.

## Variable Declaration – Explicit vs Implicit

Many, if not most, programming languages require the programmer to **explicitly** declare a variable before it is used. When you study Java, for example, to create (declare) a variable called age as an integer (a whole number), you would use the following code to both declare it and give it a value (**initialise** it). This line not only creates the variable, but also declares its data type (integer).

```
int age = 25;
```

Doing the same thing in Python requires the following code:

```
age = 25
```

Python doesn't need a data type for this. It handles numbers a little differently from many other languages – declaring the variable to have a *numeric* (that is, it's not a string “25”) value of 25 is enough for it to recognise that “age” is meant to be a number variable, and the lack of decimal points after it tells Python that the variable will be an **integer**, not a **float**.

If you wanted to make sure it was a floating point number, you would use:

```
age = 25.0
```

Now, here is the key difference between Java (**explicit** declaration) and Python (**implicit** declaration).

If you attempt to use a variable in Java that has not been declared, the program will generate an error when you try to compile it. It simply will not let you use a variable that has not been explicitly declared.

Python, on the other hand will allow you to simply create a variable “on the fly” anywhere in your program. This is called **implicit** declaration, and it does carry some level of risk. It can make debugging your code more difficult. Imagine the following scenario:

You initialise a variable near the beginning of your program:

```
age = 0
```

Further down – perhaps hundreds of lines of code further down – you want to change the value of `age` to 25. So you use the code:

```
age = 25
```

So far, so good. But still further along in your program, you want to change the value to 35, so you enter the appropriate code to do so. Unfortunately, this time you have a typo and write:

```
agr = 35
```

If you were using explicit declaration (such as in Java), the compiler would tell you that “`agr`” is an unknown variable. In Python, however, the program will happily create an entirely *new* variable called “`agr`” and continue on. Ferreting out errors of this kind can be frustrating, so it always pays to be very careful when using variables in Python to ensure you have used the correct spelling.

## Creating a variable in Python

The above section gives you a few hints on creating a variable in Python, but here we’ll look at exactly how to do it, and how we should name variables.

To create a variable, you give it a name, and **assign** it a value. We do this with the “`=`” sign, which in programming terms is called an “**assignment operator**”. We use this term because we want to differentiate the “`=`” sign from the “`==`” operator which we use to test if two variables are equal in value. We’ll get onto this in more detail further into the course.

Some examples of variables are:

```
month = "January"
```

```
month = 1
```

```
age = 44
```

```
balance = 3,242.55
```

```
isNumber = True
```

That last one is an interesting case. It uses the word “True”, but without the double-quotation marks to mark it as a string. You could also use `False`, but if you tried to use `Voldemort`, you would get an error. Why?

The answer is that True/False reflect a special data type called a **boolean**. A boolean can only be true or false, so Python recognises those key words (`True`, `False`) as valid values, rather than incorrectly placed character strings.

## Naming Variables

As a general rule:

- Variables should be given a meaningful name; using “**x**” may be fine in a program with one variable and ten lines of code, but when you have hundreds of them and a thousand lines of code, it can become very confusing.
- Variables should only use alpha-numerics (a-z, A-Z, 0 – 9). There *are* some other characters you can use in Python, but alpha-numerics is a good habit to get into, as most languages only allow you to use them.
- Variables should start with a *lower case* letter.
- If a variable has to have more than one word, start it with a lower case, and then capitalise the first character of the second and subsequent words (e.g. **albusPercivalWulfricBrianDumbledore**). This is referred to as “**camel case**”, because the word has one or more “humps” in the middle.
- Variables should never have spaces in their names. If you absolutely *have* to represent a space, use an underscore. E.g. **preferably\_dont**

## Arithmetic Operators

There’s usually not much point using variables unless we’re going to change them, and doing that usually requires one of more **arithmetic operators** in our programs. These are the operators that allow us to perform calculations. The table below summarises the various operators we can use in Python, including more detailed explanation of some of the less common ones.

Operator (arithmetic)	Name	Example	Description
<b>+</b>	Add	<b>1 + 1</b>	(result = 2); add two values
<b>-</b>	Subtract	<b>5 - 2</b>	(result = 3); subtract second value from first
<b>*</b>	Multiply	<b>2 * 10</b>	(result = 20); multiply two values together
<b>/</b>	Divide	<b>10 / 5</b>	(result = 2); divide first value by second
<b>%</b>	Modulus	<b>10 % 3</b>	(result = 1); give remainder of dividing first value by second
<b>**</b>	Exponent	<b>2**3</b>	(result = 8); increases first number by the power of the second.
<b>//</b>	Integer (or “floor”) division	<b>10//3</b>	(result = 3); gives the <b>whole number</b> result of dividing two numbers. Often used in conjunction with <b>modulus</b>

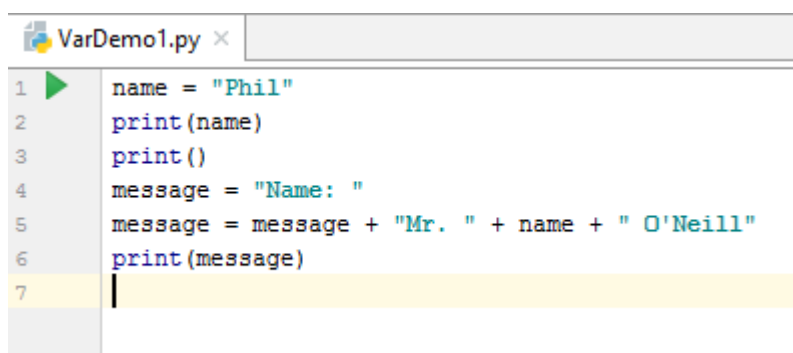
## String Concatenation

As described above, using the “**+**” sign will normally add two values together. However, using a **+** with multiple **string** variables will **concatenate**, or join, the strings together. This is quite useful when you want to combine a variable and a string together into the same output.

To see this, we’ll combine a few of the techniques we’ve been looking at into a single simple Python program and test it.

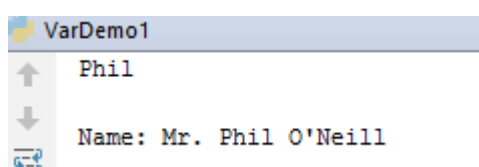
## Variable Demo Program

1. Create a new Python program called **VarDemo1.py**
2. Enter the following code into the program



```
1 name = "Phil"
2 print(name)
3 print()
4 message = "Name: "
5 message = message + "Mr. " + name + " O'Neill"
6 print(message)
7
```

3. Run the program, and you should get the following result:



```
VarDemo1
Phil
Name: Mr. Phil O'Neill
```

Can you work out what is happening? See if you can trace through the code, step-by-step, and figure out what it is doing at each step. This is called **desk checking** a program, and is an important part of program development.

Below is an analysis of the desk check for the **VarDemo1** program.

Line No.	What is happening
1	We create a variable called <b>name</b> . This a bit of memory set aside to hold a value, in this case, the character string " <b>Phil</b> "
2	We tell the program to print the contents of the <b>name</b> variable to the screen
3	We tell the program to print a blank line to the screen
4	We create a new variable called <b>message</b> . This bit of memory is assigned (initialised) the character string " <b>Name: </b> "
5	We now add to the <b>message</b> variable. We tell it that it will now equal itself (" <b>Name: </b> "), then the string " <b>Mr. </b> ", then the contents of the <b>name</b> variable (which is " <b>Phil</b> "), and finally the string " <b>O'Neill</b> ".
6	We finally print the <b>message</b> variable to the screen.


## Commenting your code

One last thing before we move onto some variable exercises. Computer code can be complex and often not immediately obvious in its intent. Different programmers have different styles, so it can be very difficult to come along after time has passed and immediately read your own code, let alone someone else's.

For this reason, it is **always** good programming practice to **comment** your code. That is, to add notes to your program as a reminder of what each “bit” does. You don’t need to comment every line of code, but it is good practice to summarise each section. It is also a good idea to add the name of the programmer, their contact details, the date of the creation and last modification of the program, and which version it is at the beginning.

We can add comments by preceding text with a “#” (“hash”, or “pound” symbol in America). The rest of that line will be ignored by the Python interpreter/compiler.

Below is an example of the **VarDemo1** program with comments included:



```
1  # Author: Phil O'Neill
2  # Contact: poneill@gordontafe.edu.au
3  # Date Written: 07/02/2018
4  # Date Modified: 13/02/2018
5  # Version: 1.1
6
7  # initialise name variable and print to screen
8  name = "Phil"
9  print(name)
10
11  # print empty line
12  print()
13
14  # initialise and then modify message variable, printing it to screen
15  message = "Name: "
16  message = message + "Mr. " + name + " O'Neill"
17  print(message)
```

You can probably see that your comments could almost form a pseudocode version of the program.

## 2 – Variable Exercises: Strings

Open and examine the sample file called **Sample02.py**

Note the use of the `\n` escape character to give a new line.

1. Modify **Sample02.py** by going to “File --> Save As” and giving it the new name **Concat1.py**. Now modify the code to display your name on one line, and your address and suburb/town on two more lines.

---

```
NAME: Phil O'Neill

ADDRESS: Gordon TAFE
Boundary Rd. East Geelong
```

2. Modify **Concat1.py** as **Concat2.py**, so that it will display the names of the days of the week, one per line, under the heading “DAYS:”. Under a second heading, “SEASONS:”, display the seasons.

---

```
DAYS:
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday

SEASONS:
Summer
Autumn
Winter
Spring
```

3. Create a Python program called **Concat3.py**  
Initialise four variables – **friend1** to **friend4**  
Display **friend1** and **friend3** on one line, and the other two on the line below, using two instances of `print()`.  
(There's a bit of a trick to this one: you can print multiple variables in one `print()` statement by separating them with commas. Try it, and ask your teacher if you need help.)

---

```
Jarrel Bob
Phil Leanne
```

4. Create a Python program called **Concat4.py** with three variables: `firstName`, `middleName`, and `lastName`. Initialise the variables to "Robert", "William", and "McPherson", respectively. Using concatenation, display the names as follows using six `print()` statements (make sure you include the commas and spaces!)

(HINT: the third line can be a little tricky, because Python tries to add spaces around the components of a `print()` statement. You can use a bit of code `sep=""` to avoid this. It sets the "separator" between the words to be an empty string. However, then you'll get the first and middle names jammed together, so you will need to add an actual empty space between them. The code would be:

```
print(lastName, ", ", firstName, " ", middleName, sep="")
```

```
Robert William McPherson
Robert McPherson
McPherson, Robert William
Robert
William
McPherson
```

5. Write a Python program called **Var05.py**  
This program should declare two variables called `season1` and `season2`.  
They should be initialised to "Winter" and "Spring".  
Display, using the variables, "Winter comes before Spring" on one line.

```
Winter comes before Spring
```

6. Modify **Var05.py** to be **Var06.py**  
Change the output so that it displays across three lines.

(HINT: Try this without using the `sep=""` snippet mentioned in #4, above. What happens? Why?)

```
Autumn
comes before
Winter
```

7. Modify **Var06.py** to **Var07.py**  
Declare another variable called `message`.  
Use concatenation to assign the sentence about the seasons to `message`, then display `message`.

```
Autumn
comes before
Winter
```



8. Create a Python program called **Var08.py** that has three variables – **name**, **address**, and **phone**. Initialise these with your details.  
Create a fourth variable called **details**. Use the **name**, **address**, and **phone** variables to assign a string to **details** that places your name, address, and phone number on three separate lines. Display the details using a single **print()**.

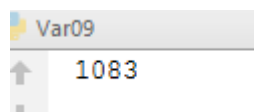
```
Fred Flintstone
47 Rocky Road, Stony Rises
0456 987 909
```

## 2 – Variable Exercises: Numeric Variables

Open and examine the sample file called **Sample04.py**

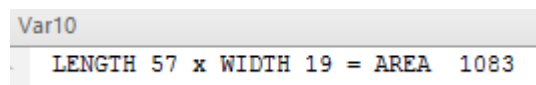
Note how the numeric variables and string literals are combined in the **print()** statement.

9. Create a Python program called **Var09.py** that declares three integer variables: **length**, **width**, and **area**.  
Initialise them as:  
**length**: 57  
**width**: 19  
remember that you can initialise **area** as the result of the calculation.  
Calculate the area by multiplying the **width** times the **length**, and assigning the result to the **area** variable.  
Display the **area** only



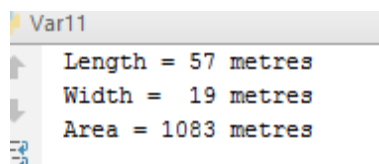
```
Var09
1083
```

10. Save **Var09.py** as **Var10.py** and modify it so that it displays:



```
Var10
LENGTH 57 x WIDTH 19 = AREA 1083
```

11. Save **Var10.py** as **Var11.py** and modify it so that it displays the output over three lines:



```
Var11
Length = 57 metres
Width = 19 metres
Area = 1083 metres
```

12. Write a Python program called **Var12.py**. It should have three variables called **unitPrice**, **itemsPurchased**, and **totalPrice**. Initialise the variables with the following values:

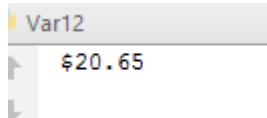
**unitPrice**: 2.95

**itemsPurchased**: 7

**totalPrice**: **unitPrice** \* **itemsPurchased**

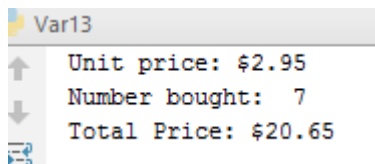
It should display the total price only:

**HINT**: you can force a number to a number of decimal places by using a string that includes a **{}** placeholder, and the variable to be printed inside a **.format()** function. In this example, you would use: **print("\${0:.2f}".format(totalPrice))**



```
Var12
$20.65
```

13. Save **Var12.py** as **Var13.py** and modify it so that it produces the following output:

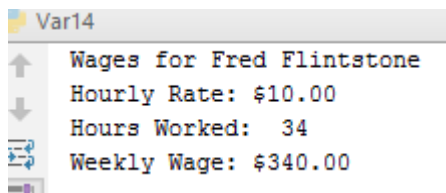


```
Var13
Unit price: $2.95
Number bought: 7
Total Price: $20.65
```

14. Write a Python program called **Var14.py**.

It should declare and initialise variables to hold a **name**, a **wage rate**, and the **number of hours worked**. It should calculate the weekly wages earned by multiplying the **rate** by the **hours worked**.

The output should appear as follows:



```
Var14
Wages for Fred Flintstone
Hourly Rate: $10.00
Hours Worked: 34
Weekly Wage: $340.00
```

15. Write a Python program called **Var15.py**

It should declare four **whole number variables** to hold the number of pounds, shillings, pennies, and total pennies. Initialise them with the following values:

pounds: 2

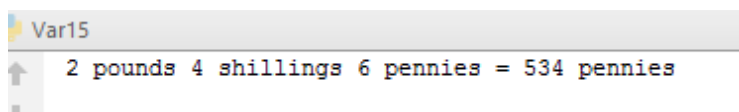
shillings: 4

pennies: 6

total pennies: 0

Your program should then calculate the total number of pennies (there are 12 pennies in a shilling, and 20 shillings in one pound)

Display the result on a single line as follows:



```
Var15
2 pounds 4 shillings 6 pennies = 534 pennies
```