# Fast-track Python

## Sample manual - first two chapters

Wise Owl Training

Manual 1335 - 210 pages –

## TABLE OF CONTENTS (1 of 7)

## TABLE OF CONTENTS (2 of 7)

## TABLE OF CONTENTS (3 of 7)

# TABLE OF CONTENTS (4 of 7)

## TABLE OF CONTENTS (5 of 7)

# TABLE OF CONTENTS (6 of 7)

## TABLE OF CONTENTS (7 of 7)

# CHAPTER 1 - GETTING STARTED

## 1.1 Introducing Python

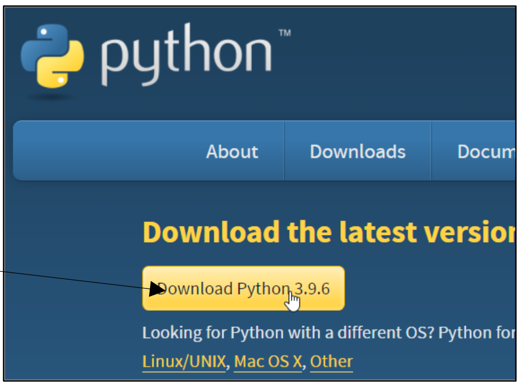
Python is the only programming language named after a BBC comedy series. It was originally created by a Dutch programmer called Guido van Rossum.

**Wise Owl's Hint**

*The author has programmed extensively in VB, C# and SQL, and is an enthusiastic convert to Python. It will let you develop powerful programs quickly, although its management of paths and packages will have you tearing your hair out!*

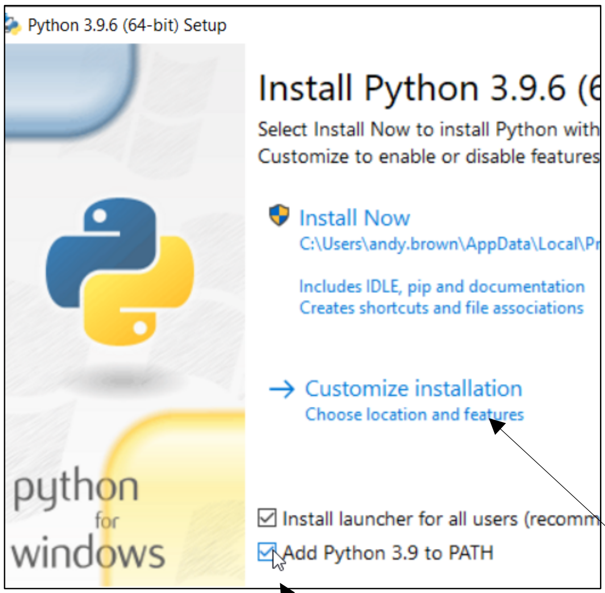
### Installing Python

You can download Python from https://www.python.org/downloads/ :


python™
About
Downloads
Docum
Download the latest versio
Download Python 3.9.6
Looking for Python with a different OS? Python fo
Linux/UNIX, Mac OS X, Other

Click on this button to install the latest version of Python at the time of writing.

Here are our recommended settings:



Python 3.9.6 (64-bit) Setup

Install Python 3.9.6 (6

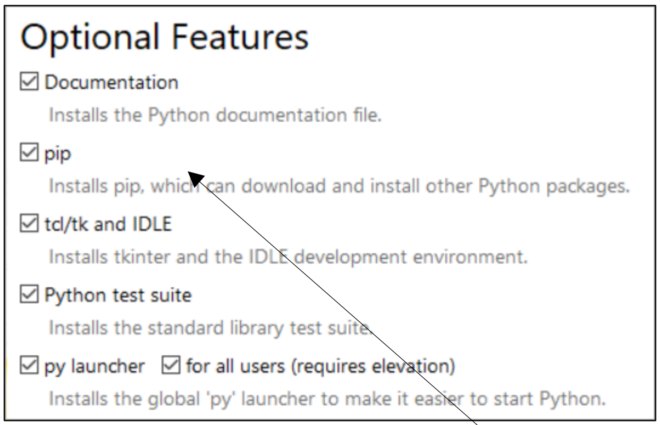
Select Install Now to install Python with
Customize to enable or disable features

Install Now
C:\Users\andy.brown\AppData\Local\Pr

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Install launcher for all users (recomm
☑ Add Python 3.9 to PATH



Optional Features

☑ Documentation
Installs the Python documentation file.

☑ pip
Installs pip, which can download and install other Python packages.

☑ tcl/tk and IDLE
Installs tkinter and the IDLE development environment.

☑ Python test suite
Installs the standard library test suite.

☑ py launcher ☑ for all users (requires elevation)
Installs the global 'py' launcher to make it easier to start Python.

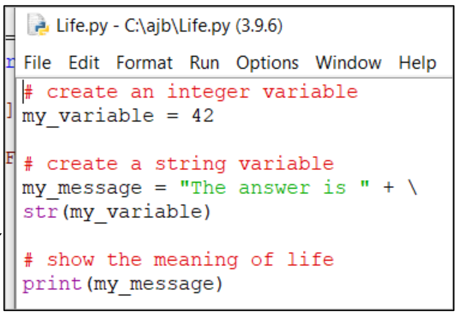
a) Managing paths in Python is a pain! Do yourself a favour and tick this box to help Windows programs run Python easily.

b) You can choose which bits of Python you want to install, although it's probably best just to go with the defaults.

You'll certainly want to install **pip** (which will help you import modules to accomplish different tasks), **tcl/tk** (which will help you create GUI applications) and **IDLE** (a Python editor – see next page).

## 1.2 Choosing an Editor

Python comes with its own built-in editor called IDLE (named after Eric), but it's a bit primitive:


Life.py - C:\ajb\Life.py (3.9.6)
File Edit Format Run Options Window Help

```
# create an integer variable
my_variable = 42

# create a string variable
my_message = "The answer is " + \
str(my_variable)

# show the meaning of life
print(my_message)
```

A program written in IDLE. Although it's a great package for getting started, it doesn't have true Intellisense (which in this owl's view rules it out as a serious development environment).

### Choices of Editor

Here are some possible editors that you could choose:

| Editor | Notes |
|---|---|
| *Visual Studio Code* | A generic code editor maintained by Microsoft but available free of charge (don't confuse it with Visual Studio, which is a completely different program – see below). |
| *PyCharm* | An editor devoted to writing Python code. Reviews online suggest that it can be very slow to work with, and some users will need to upgrade to the paid Premium edition. |
| *Visual Studio* | If you already spend time working in Visual Studio, you may find it easiest to use this as your development environment (although it's a bit of a big beast!). |
| *Jupyter Notebooks* | If you work in machine learning or AI you may well choose this powerful coding environment. |

**Wise Owl's Hint**

*There are many other Python editors out there with names like Atom and Sublime, as well as tools which will manage your Python code such as Anaconda. This courseware uses IDLE to get started, then switches to Visual Studio Code.*

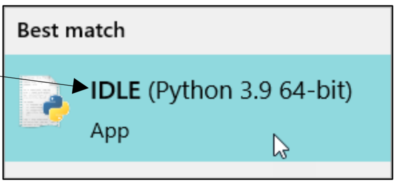
## 1.3 Using IDLE

IDLE was (allegedly) named after Eric Idle, one of the Monty Python team. There's also a Python editor called *Eric*, but none called *Cleese* or *Palin* that Wise Owl know of.

The acronym does work well, it has to be said.

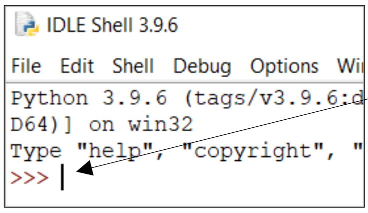
When you install Python, you should automatically get IDLE at the same time:

The latest version of the IDLE Python editor (at the time of writing).



### Running Single Commands (Interactive Mode)

You can run any single command by typing it in at the command prompt and pressing ↵ :

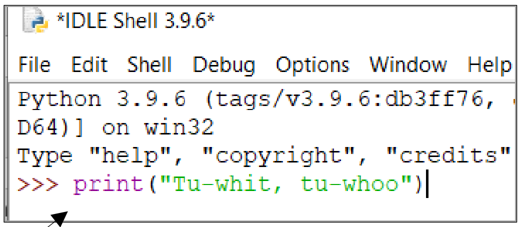


```
Python 3.9.6 (tags/v3.9.6:d
D64)] on win32
Type "help", "copyright", "
>>> |
```

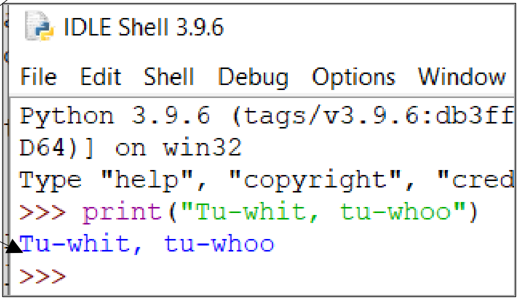
a) The >>> text is called the command prompt - it's waiting for you to type in a valid Python command.

b) The print command just displays the information in parentheses in IDLE.

c) The output of your command is the message that you chose to print.



```
Python 3.9.6 (tags/v3.9.6:db3ff76,
D64)] on win32
Type "help", "copyright", "credits"
>>> print("Tu-whit, tu-whoo")|
```



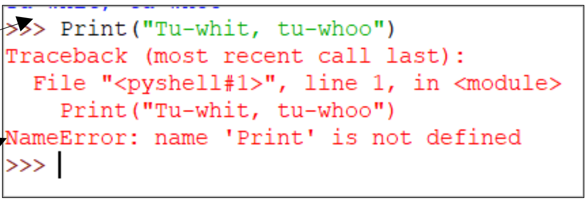
```
Python 3.9.6 (tags/v3.9.6:db3ff
D64)] on win32
Type "help", "copyright", "cred
>>> print("Tu-whit, tu-whoo")
Tu-whit, tu-whoo
>>>
```

### Colour-Coding / Case Sensitivity

Note that Python is a case-sensitive language!

Here IDLE hasn't colour-coded the word **Print**, because it doesn't recognise it as a valid Python command ...

... and displays an error when you press ↵ to try to run the command.

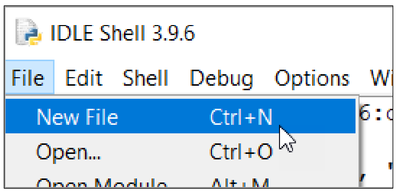


```
>>> Print("Tu-whit, tu-whoo")
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    Print("Tu-whit, tu-whoo")
NameError: name 'Print' is not defined
>>> |
```

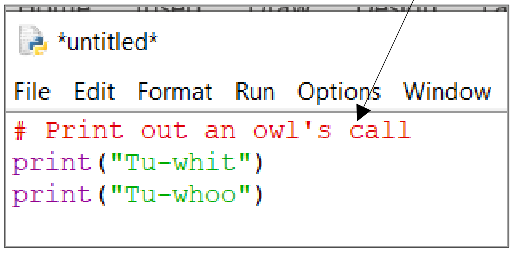
## Creating, Saving and Running Programs

If you want to execute a sequence of commands, you don't have to run each one individually; instead you can save them in a *file*:

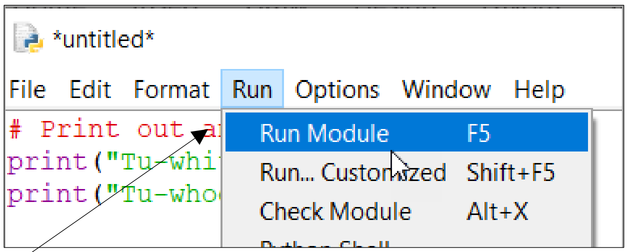
a) Choose to create a new file from the **File** menu (as this shows, you can instead press Ctrl + N ).



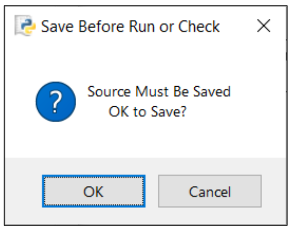
b) Type in a sequence of valid Python statements (the # symbol denotes a comment, which will be ignored – more on comments shortly).



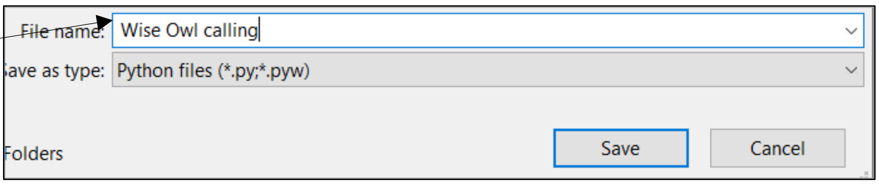
```
# Print out an owl's call
print("Tu-whit")
print("Tu-whoo")
```


*untitled*
File Edit Format Run Options Window Help
Run Module F5
Run... Customized Shift+F5
Check Module Alt+X
Python Shell

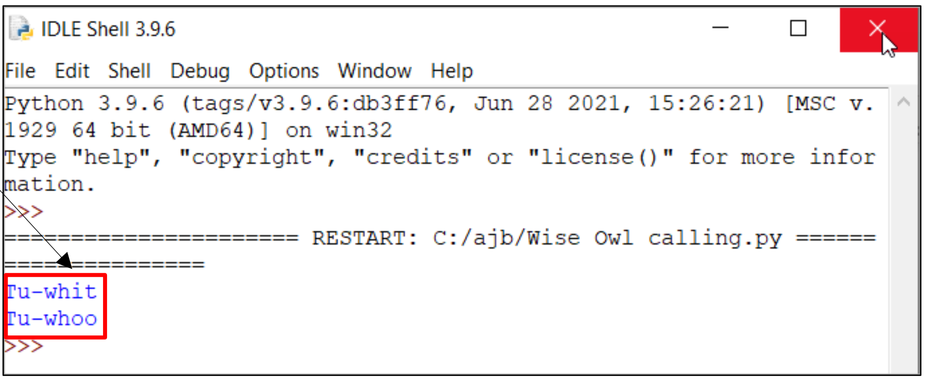
c) Choose this option to run your code (although it's quicker to press F5 instead as a short-cut). You'll be prompted to save your code in a file – until you do this you won't be able to run it.



d) Your file will be saved with an extension of **.py**.


File name: Wise Owl calling
Save as type: Python files (*.py;*.pyw)
Folders
Save
Cancel

e) You can now see the output from your program. When you've finished, click on the cross at the top right to close down the IDLE shell and return to your program.



```
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.
1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more infor
mation.
>>>
===================== RESTART: C:/ajb/Wise Owl calling.py ======
===============
Tu-whit
Tu-whoo
>>>
```

Wise Owl's Hint

*You can obviously open existing files that you've previously created to run their code instead.*

# CHAPTER 2 - BASIC CODING

## 2.1 Comments

Good programmers add *comments* to their code, to explain what it is meant to do!

### Single-line Comments

Most Python comments begin with a single # character:

```
# A simple program to print
# out a wise owl's call
# using two separate lines

# Here's the first line ...
print("Tu-whit")

# ... and here's the second
print("Tu-whoo")
```

Each separate line has to begin with its own # character. The red lines in this program will be completely ignored by the Python interpreter.

### Multi-line Comments

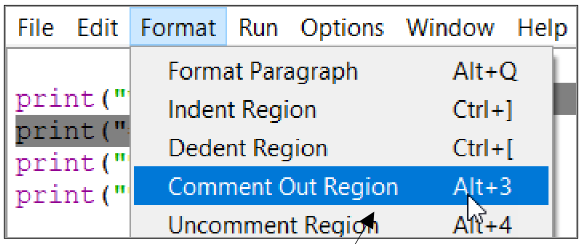
You can use three double-quotation marks in a row to mark out multiple comment lines:

```
"""
A simple program to print
out a wise owl's call
using two separate lines
"""

# Here's the first line ...
print("Tu-whit")

# ... and here's the second
print("Tu-whoo")
```

These lines will be treated as comments (confusingly, IDLE chooses to show them in green, not red).

### Commenting Out Lines (and Uncommenting)

If you want to avoid running certain lines without removing them, you can *comment them out*:



```
print("
print("
print("
print("
```

```
##print("What noise do owls make?")
##print("========================")
print("Tu-whit")
print("Tu-whoo")
```

a) Select part or all of the lines you want to comment out and press Alt + 3 or choose this menu option.

b) For some reason IDLE puts two hashes in front of each commented out line.

You can select commented out lines and press Alt + 4 to reinstate them (or choose the **Uncomment Region** menu option shown above).

## 2.2 Variables

A *variable* is a space inside your computer which holds a single bit of information (be it a number, date, string of text or other value).

Two examples of Python variables.

```
# a variable to hold a company name
company_name = "Wise Owl Training"

# a variable to hold a number
answer = 42
```

Wise Owl's Hint

*The Python naming convention is to avoid camel case but instead use underscores to divide the parts of a variable name. Thus you might call the second variable above **meaning_of_life**, but wouldn't call it **MeaningOfLife**.*

### Declaring Variables

Here's how you declare an integer variable in 4 commonly used programming languages:

| Language | Variable declaration |
|---|---|
| *C#* | `// integer variable`<br>`int answer = 42;` |
| *SQL* | `-- declare an integer variable`<br>`DECLARE @answer int = 42` |
| *JavaScript* | `// declare a variable`<br>`var answer = 42;` |
| *Visual Basic* | `'create integer variable`<br>`Dim answer As Integer = 42` |

In Python, by contrast, you don't declare a variable before using it (the act of assigning a value to a variable automatically declares it at the same time).

```
# an integer variable to hold a number
answer = 42

# now print out this value (note misprint)
print(answe)
```

```
Traceback (most recent call last):
  File "P:/Manuals/new/Python/Files/Var
    print(type(answe))
NameError: name 'answe' is not defined
```

Here we've created a variable called **answer**, but are trying to print out the value of an (uncreated) variable called **answe**.

The Python interpreter points out the error of your ways when you try to run your program.

Wise Owl's Hint

*For experienced programmers in other languages this will be one of the weirdest things to get used to about Python, but it's a good idea (to the extent that you will now probably resent having to formally declare variables in other languages).*

## Variable Types

Here are some of the common data types in Python:

| Data type | What it contains |
|---|---|
| **str** | Any string of text |
| **int** | Any whole number |
| **float** | Any decimal number |
| **bool** | Something which can be either true or false |

Wise Owl's Hint

*It is typical of Python that the old **long** type used to denote very large integers is no longer needed, and **int** covers everything from 0 to infinity!*

## Determining Type

Python determines the type of a variable from the value you assign to it. You can see this by using the **type** function to investigate a variable's data type:

```
# set variable to hold a number
some_variable = 42
print(type(some_variable))

# change it to hold text
some_variable = "Wise Owl Training"
print(type(some_variable))

# now store a floating-point number
some_variable = 3.14
print(type(some_variable))

# now store a true/false value
some_variable = True
print(type(some_variable))
```

This program will assign different values to the same variable. After assigning each value, we print out the type of the variable.

Here's what this program would output:

```
<class 'int'>
<class 'str'>
<class 'float'>
<class 'bool'>
>>> |
```

## Assigning Values to Variables

You can assign values to individual variables as we've already seen by using this convention:

```
variable_name = value_for_variable
```

If you're assigning two or more variables to the same value, you can do this in a single line:

```python
# three variables to hold the same thing
bird_name = company_mascot = pooh_friend = "Owl"

# change the text of one of them
company_mascot = 'Wise Owl'

# prove this by printing out values
print(bird_name)
print(company_mascot)
print(pooh_friend)
```

This code would create 3 variables, but they would all contain the same value if we hadn't subsequently changed the value of one of them.

```
Owl
Wise Owl
Owl
>>>
```

You can do multiple assignments on the same line (although this owl thinks it makes your code harder to read):

```python
# create 3 variables in one line
company_name, answer, months_in_year = "Wise Owl", 42, 12

print(company_name)
print(answer)
print(months_in_year)
```

This code would create (then show the values of) one string and two integer variables:

```
Wise Owl
42
12
>>>
```

## Deleting Variables

Python will delete any variables that you've created when the program containing them finishes, but sometimes you may want to pre-empt this. You can do this using the **del** command:

```python
# a string variable to hold a company name
company_name = "Wise Owl Training"

# an integer variable to hold a number
answer = 42

# delete both variables
del answer, company_name

# print out their values (this will crash)
print(company_name)
print(answer)
```

Once you've deleted a variable, not surprisingly you can no longer refer to it. Running this program would give the following error:

```
NameError: name 'company_name' is not defined
>>>
```

## 2.3 Rules of Arithmetic

Python follows the same order of arithmetic operation as most other computer packages (taught in schools as *BODMAS*, standing for *Brackets Of Division Multiplication Addition Subtraction*).

```
# create 3 variables
first = 3
second = 5
third = 2

# perform two calculations
answer_using_bodmas = first + second * third
answer_using_brackets = (first + second) * third

# show answers
print(answer_using_bodmas)
print(answer_using_brackets)
```

You can use parentheses to change the default order of operation. This code would give the following output:

```
13
16
>>>
```

For the second calculation, Python sums the first two numbers before multiplying the result by the third one.

In addition to the standard operators of `+` , `-` , `*` and `/` you can also use these:

| Operation | Operator | |
|---|---|---|
| *Raising to the power* | `**` | The following code excerpt would return $2^{10}$, or 1024:<br>`# base number and power`<br>`base_number = 2`<br>`power = 10`<br><br>`# get answer`<br>`answer = base_number ** power`<br><br>`# show result`<br>`print(answer)` |
| *Taking the remainder or modulus of a number* | `%` | The following code would return 1 (the remainder when you divide 22 by 7):<br>`# base numbers`<br>`start_number = 22`<br>`divisor = 7`<br><br>`# get answer`<br>`answer = start_number % 7`<br><br>`# show result`<br>`print(answer)` |

## 2.4 Basic Strings

A later chapter will give much more details on the tricks and functions you can use when working with strings of text; this page just shows a few basic ones.

### New Lines and Tabs

You can use the escape characters `\n` and `\t` to include new lines and tabs in your output:

```
# include tabs and a line break in name
signature = "Company\t\tWise Owl\nBusiness\tTraining"
print(signature)
```

This program would produce the following output:

```
Company         Wise Owl
Business        Training
>>> |
```

**Wise Owl's Hint**

*Tabs are an unreliable way to align output, as the example above shows (the first line includes two tabs, but the second only one). A better way to align text is to use the* ***ljust***, ***rjust*** *and/or* ***center*** *functions (covered in a later chapter)*

### Quotation Marks

To create a string which includes quotation marks, either use an escape character or switch from double to single quotation marks (the second way seems easier):

```
# two ways to include quotes
owl_name_1 = "My name is 'Owl'"
owl_name_2 = "My name is \"Owl\""

# show the results
print(owl_name_1)
print(owl_name_2)
```

Both of these ways would embed quotation marks in the relevant strings of text, to give this output:

```
My name is 'Owl'
My name is "Owl"
>>> |
```

### Backslash Characters

Since the escape character is a `\` , how can you include this in a string of text? The answer is to repeat it:

```
# show the file path
print("The file path is C:\\wiseowl\\python\\")
```

This program would give the following output:

```
The file path is C:\wiseowl\python\
>>>
```

## Concatenating Text

Use the + symbol to join bits of text together:

This program would join the two variables together with a space between them, to give this:

```
Wise Owl
>>> |
```

```
# two string variables
start = "Wise"
end = "Owl"

# show company name
print(start + ' ' + end)
```

## Converting Numbers to Text

You can not join a string with a number; instead, you must first convert the number to a string using the **str** function.

```
# set name and age of someone ...
your_name = "Bob"
your_age = 42

# ... and print this out (will crash)
print (your_name + ", you are " + \
your_age + " years old")
```

This program will crash because it is trying to join a string of text (**your_name**) with an integer (**your_age**):

```
    print (your_name + ", you are " + \
TypeError: can only concatenate str (not "int") to str
>>>
```

Note the use of a backslash at the end of this line to act as a continuation character, allowing a single programming command to span multiple lines.

Here's a working version of the code above:

You must use the **str** function to convert numbers to text before joining them with other bits of text. This would give:

```
Bob, you are 42 years old
>>> |
```

```
# set name and age of someone ...
your_name = "Bob"
your_age = 42

# ... and print this out
print (your_name + ", you are " + \
str(your_age) + " years old")
```

## Getting Inputs from Users

You can pause a program to ask a user to input values using the **input** function - for example:

```
# get someone's name
your_name = input("Enter your name ==> ")
your_age = input("Enter your age ==> ")

print (your_name + ", you are " + \
your_age + " years old")
```

Running this program (and inputting the values **Bob** and **42** at the prompts) would give this output:

```
Enter your name ==> Bob
Enter your age ==> 42
Bob, you are 42 years old
>>> |
```

**Wise Owl's Hint**

*Note that the **input** function always gives a string of text, so there's no need to convert the 42 above to a string before concatenating it with the user's name.*

## 2.5 Testing Conditions

### Simple Conditions

You can use the **if** statement to test conditions in Python, but you must follow it with a colon `:` and indentation:

In most languages you would indent your code at this point by pressing `Tab` to make it more readable. In Python this space is a vital part of your code, and without it you'll get a run-time error.

```
# get the user's age
age = input("Type your age ==>")

# warn user if too young (must convert
# string returned from input to int first)
if int(age) <= 18:
    print("Sorry you are too young")
```

### Multiple Conditions

If you want to test whether a condition is true or false, use **else**:

```
# get the user's age
age = input("Type your age ==>")

# convert this to an integer
# (note that this will crash if it isn't)
age_as_int = int(age)

# test age to see whether under 18 or not
if age_as_int < 18:
    print("Sorry you are too young")
else:
    print("Welcome!")
```

This program will display a different message for someone under 18 than for an adult. Here's a typical output from running the program:

```
Type your age ==>21
Welcome!
>>>
```

For multiple conditions, use as many **elif** statements as you need:

Here the program tests (in this order):

- Whether the age is less than 18
- Whether the age is less than 40 (knowing that it can't be under 18, otherwise it would have passed the first test)
- Whether the age is more than 60

The program only prints out a welcome message for people who don't meet any of these conditions. A typical output might be:

```
Type your age ==>65
Sorry you are too old
>>> |
```

```
# get the user's age
age = input("Type your age ==>")

# convert this to an integer
# (note that this will crash if it isn't)
age_as_int = int(age)

# test age to see what to do
if age_as_int < 18:
    print("Sorry you are too young")
elif age_as_int < 40:
    print("Sorry you are too middle-aged")
elif age_as_int > 60:
    print("Sorry you are too old")
else:
    print("Welcome!")
```

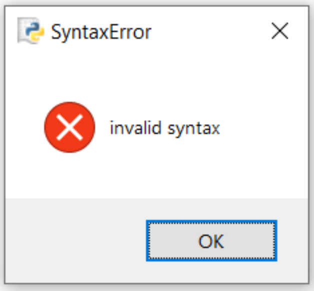
**Wise Owl's Hint**

*From version 3.10 Python will have a powerful **match** statement giving the equivalent of the C#/JavaScript **switch** statement, the VB **SELECT CASE** statement or the SQL **CASE WHEN** statement, reducing the need for multiple **elif** statement blocks like the one shown above.*

## Testing for Equality

When you are testing if two values are equal in Python, you must use two = signs in a row.

```
# get two people's ages
first_age = 42
second_age = 44

# are they the same?
if first_age = second_age:
    print("Same ages")
else:
    print("Different ages")
```


SyntaxError

invalid syntax

This can be a very disconcerting error message to see – what could you possibly have done wrong? The answer is that because you're testing a condition you need to put this:

```
# are they the same?
if first_age == second_age:
    print("Same ages")
else:
    print("Different ages")
```

**Wise Owl's Hint**

*If you're not used to it, this Python feature (and the fact that everything is case-sensitive) will probably account for about 90% of the bugs that you create!*

## Combining and Negating Conditions

You can use the **and**, **or** and **not** keywords to test different combinations of conditions:

```
# a (hopefully) nostalgic look back ...
if_vaccinated = True
if_masked = True
if_coughing = False

# a few places just require you to look well
if not if_coughing:
    print("Welcome (level 0)")

# can shop if either vaccinated or masked
if if_masked or if_vaccinated:
    print("Welcome (level 1)")

# some places require both
if if_masked and if_vaccinated:
    print("Welcome (level 2)")
```

This program would output this::

```
Welcome (level 0)
Welcome (level 1)
Welcome (level 2)
>>>
```

This is because:

- This person is NOT coughing
- The person is either vaccinated OR masked (in fact, they're both)
- The person is masked AND vaccinated

Python also treats the following two statements as identical:

```
if age >= 18 and age < 65:

if 18 <= age < 65:
```

The second test also checks that 18 is less than or equal to the value of the variable **age** and that the value of the variable **age** is less than 65.

## What we do!

| | | Basic training | Advanced training | Systems / consultancy |
|---|---|---|---|---|
| Office | Microsoft Excel | ✓ | ✓ | ✓ |
| | VBA macros | ✓ | ✓ | ✓ |
| | Office Scripts | ✓ | | |
| | Microsoft Access | | | ✓ |
| Power BI, etc | Power BI and DAX | ✓ | ✓ | ✓ |
| | Power Apps | ✓ | | |
| | Power Automate (both) | ✓ | ✓ | |
| SQL Server | SQL | ✓ | ✓ | ✓ |
| | Reporting Services | ✓ | ✓ | ✓ |
| | Report Builder | ✓ | ✓ | ✓ |
| | Integration Services | ✓ | ✓ | ✓ |
| | Analysis Services | ✓ | | |
| Coding | Visual C# | ✓ | ✓ | ✓ |
| | VB programming | | | ✓ |
| | MySQL | ✓ | | |
| | Python | ✓ | ✓ | ✓ |

Wise Owl Training