

Python Programming

Section 1 – Introduction to Programming

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

1 – Introduction to Programming

Welcome to the first lesson of “Python Programming”. This is designed to give you an introduction to not only programming with the Python language, but a general introduction to computer programming.

Computer programming has advanced a great deal in the last few years. Where once it was almost exclusively centred around creating desktop applications, it now covers those, plus mobile applications for tablets and smartphones, back-end website programming, and even creating the software that runs electro-mechanical devices from microwaves to modern cars right up to interplanetary space probes.

Let’s have a quick look at what programming actually is:

- A computer cannot think (at least, not yet!), but it can follow instructions very quickly and accurately.
- A set of instructions for a computer to process is called a **program**.
- **Software** is the collective term used to describe the programs that run on a computer system.
- Software is stored on EEPROM (“Flash”) devices (e.g.: USB sticks, SSD hard drives), magnetic disks (e.g.: hard disk drives), optical disks (e.g.: CD-ROM, Blu-Ray), and even magnetic tape (though this is less common today).
- The process of creating software is programming. The writer of programs is a programmer.

Computer Systems

A computer system basically consists of three parts:

- User – a person needed to start and control the computer system
- Hardware – the physical components of a computer system.
 - Input devices (keyboard, mouse, touchpad, etc.)
 - Central processing unit (CPU)
 - Random access memory (RAM) – sometimes called “main memory” or “primary storage”
 - Secondary storage – hard drives, optical drives, USB drives, etc.
 - Output devices (monitor, printer, speakers, etc.)
- Software – the programs that run on the computer (see below)

A word about precision

It is *extremely* important, when working in software development, that you take special care to maintain absolute precision in your work.

There are two parts to this:

Computer code must be *absolutely precise*. The most common cause of errors in a program is a typo, or (in Python) a line of code that is incorrectly indented. Make sure you take careful note of messages displayed when you try to run your code and run into an error. We will look at debugging a little further into the course.

The second potential problem with a lack of precision is **naming conventions**. You must follow naming instructions *exactly*. It is a good habit to get into, because when working as a programmer, you will be working as a member of a team. Each member will be responsible for different parts of the project, and if you do not follow the instructions on naming files, variables, classes, etc., the code you write will not work with that of others.

This is even further exacerbated in Java (for example), where the file name has to exactly match the class name within the code (you’ll see this if you continue to study Java programming in Semester 2).

Software

There are three general categories of software:

- System software
 - Programs which control the hardware
 - Manage and direct the computer resources during the execution of programs
 - Coordinate input / processing / output activities required by a program
 - Examples: operating systems such as Windows, Linux, MacOS.
- Application software
 - Programs which perform specific functions
 - Examples: banking systems, financial software such as MYOB, word processing and spreadsheet software (sometimes called “office software”)
- Utility software
 - Programs which are designed to make the computer more useful, and easier to use
 - They perform many “house-keeping” tasks
 - Examples: text / hex editors, compilers, Windows explorer

Programming Languages

All programs are written in a programming language of some kind. There are three broad levels of language:

- Machine code – this is the only language the computer itself understands, and is just a huge batch of 0s and 1s.
- Assembly language – directly addresses the memory registers in the CPU. Easier to understand than machine code, but still complicated in comparison to high level languages.
- High Level languages – easiest to code in. English-like instructions. (e.g.: Python, Java, C#, C, Visual Basic, Pascal, COBOL, C++, etc.)

Before a program can be run on a computer it must be translated into machine language (binary). The process can be complicated, and some languages are referred to as “interpreted” because they do this at run-time rather than at design time. Python is first “interpreted” into something called “byte-code” before it is compiled into binary, so it is a bit of both “interpreted” and **compiled**.

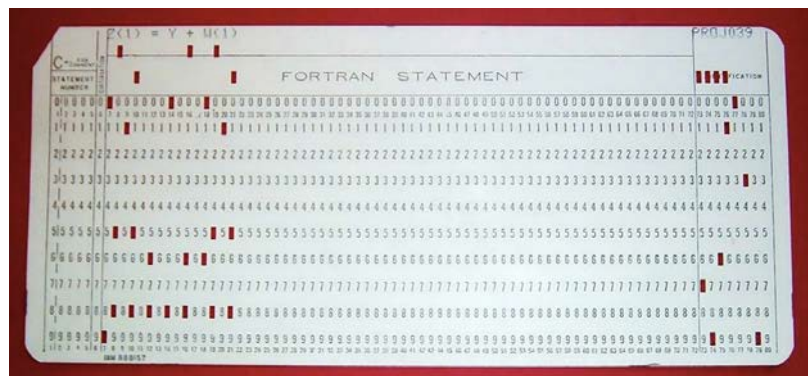


Figure 1 - FORTRAN programming punch card, c. late 60s - early 70s.

A compiler is a bit of utility software which makes a programmer's job much easier, because they do not have to write out the ones and zeroes of machine language as they once had to. In "the old days", computer programs were written in ones and zeroes by punching out sections of cardboard which were then fed into the computer. A mistake meant the programmer had to throw out the card and start a new one. Writing machine code was extremely laborious and time-consuming.



Figure 2 - Grace Hopper (centre right) at the UNIVAC 1, c. 1960

The first compiler was written in 1952 and referred to as "A-0", by an American programmer named Grace Hopper. Hopper had served in the US Navy during World War II as a computer scientists, pioneering some of the earliest military computers, and went on to become the "mother" of modern computer programming, helping to create the COBOL programming language and the UNIVAC 1, the world's first commercially-available computer.

She has a US Navy destroyer (DDG-70 *USS Hopper*) named after her, and its unofficial nickname is "Amazing Grace".

She is also known by the moniker "Granny COBOL".

Introducing Python

While Python itself is a programming language, to use it you will need to install some software. Some large-scale professional programming suites (such as Visual Studio 2017) have a Python option, using them for simply Python programs is overkill.

The most basic thing you will need to install is the actual Python language itself. At the time of writing, the latest version of Python is version 3.6.4, but this is updated quite often, so you may see a slight difference the version numbers.

If you go to <https://www.python.org/downloads/> you can click on the button to download the latest version, which will look something like *Figure 3*.



Figure 3

Version 2 or 3?


There are two broad versions of Python; 2 and 3. Each is similar to the other, but there are enough differences to make most programs written in one fail if you attempt to run it on the other.

Version 2 is the older one. We will be using Version 3 in this course, and all the handouts, sample files, and reference materials are coded in 3, so make sure that's the one you download.

Installing Python

When you run the installation file you just downloaded, it will install into a folder on your computer in a location *something like* the following (the details may vary from one computer to another):

"C:\Program Files (x86)\Python36-32"

You can already start programming just with this. Clicking on your Windows button  will let you navigate to the “IDLE (Python 3.6 *n*-bit)” (where “*n*” is either 32 or 64, depending on your installation, but most likely 32-bit).

This will open up “IDLE”, Python’s “plain vanilla” editor. You can program from this, but it lacks some of the features we’ll be using, so you are well-advised to follow the instructions below to install “PyCharm Edu”, a free Python **integrated design environment** (IDE) which makes managing, creating, and using your Python programs much simpler.

Getting and Installing PyCharm Edu

You can download PyCharm Edu from: <https://www.jetbrains.com/pycharm-edu/download/#section=windows>

The latest version at the time of writing is 4.0.2, but again, this updates frequently. Don’t worry if the version isn’t exactly the same as the instructions below – the process is still the same.

Installation is as simple as double-clicking the **.exe** file you downloaded and following the wizard’s instructions.

Once it is downloaded, you will need to do some configuration, which is outlined at the end of this document.

A bit about Python

Guido van Rossum created Python in the early 1990s to be a highly readable and easy-to-learn language with a great deal of flexibility. The philosophy of its design is to keep the core language simple, with complexity added only as needed (using modules, which we’ll address later on).

And yes, as a fan of the comedy group, Rossum named the language after Monty Python.

Python is especially suited to gathering and analysing data, and because of its robustness and ease-of-learning, it is often used in scientific fields, where scientists (who are not programmers by trade) often have to create programs to achieve or analyse things within their disciplines.

Two notable scientific organisations that make extensive use of Python are CERN and NASA.

2 – Algorithms

An **algorithm** is a set of step-by-step instructions that are designed to solve a problem. Some common real-world examples of an algorithm are knitting patterns, recipes, and furniture assembly instructions.

To solve a problem on a computer, you need to devise an algorithm as a possible solution. A difficult problem can be solved more easily by dividing it up into smaller sub-problems and then working out a series of steps to solve each one.

When building a house, for example, a builder doesn't try to put everything together at once. The construction will be divided into sub-problems. For example:

- Lay the concrete slab
- Construct the wall frames
- Construct the roof frame
- Tile the roof
- Put in the windows
- Clad the wall frame (bricks, timber, etc.)
- Fit out the interior

Each of these steps has its own series of problems to solve. For example, laying the concrete slab would require:

- Measure out the area of the slab
- Calculate the amount of concrete required
- Order the concrete
- Build the timber formwork
- Lay down the steel reinforcement
- Pour the concrete
- Smooth the surface of the concrete
- Etc.

So rather than trying to figure out everything at once, the builder will organise the tasks into a logical sequence, and break each task into a group of sub-tasks to be completed.

Representing Algorithms

We can represent algorithms in a few different ways:

1. Use a formula:
$$\text{net} = (\text{gross} - (\text{gross} * 0.25)) - (\text{deduction1} + \text{deduction2} + \text{deduction3})$$
2. Using English instructions
 - a. Take 25% of gross wage from gross wage
 - b. Add all the deductions together
 - c. Subtract the result of b) from the result of a) to give the net wage

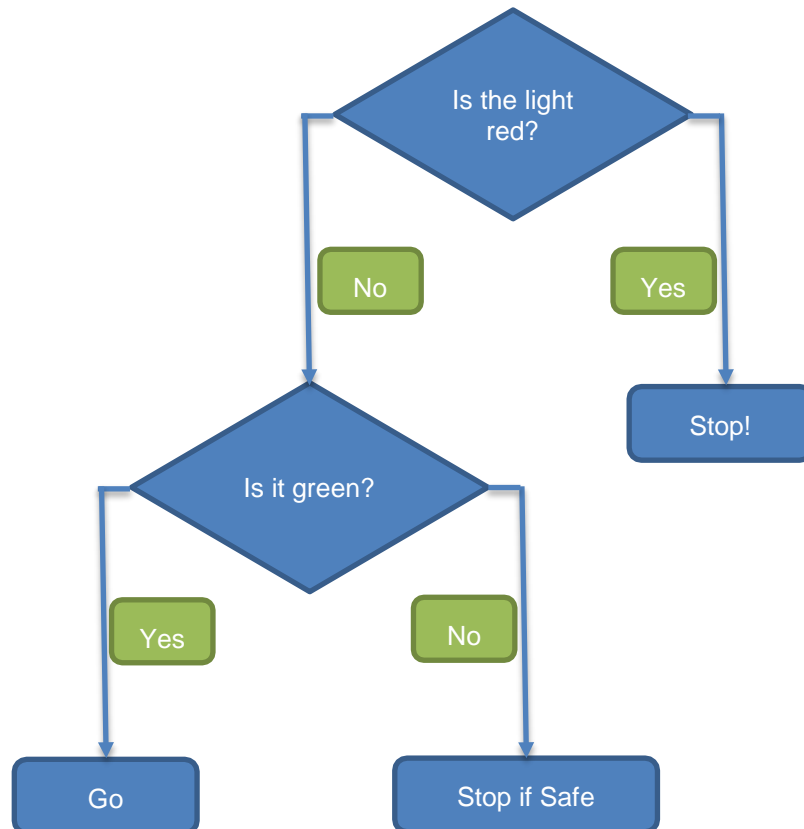
3. Using pseudocode, which is often used for designing computer programs. It is a combination of English and language-specific control words.

deductions = deduction1 + deduction2 + deduction3

tax = gross * 0.25

net = gross – tax – deductions

4. Using diagrams (e.g.: flowchart, Nassi-Shneidermann diagrams, Hierarchy structure charts, etc.). The best known (and most intuitive) is the Flow Chart



In programming, we have three main actions to use when we want to solve a problem. Briefly, they are:

- SEQUENCE
- SELECTION
- REPETITION (or “iteration”)

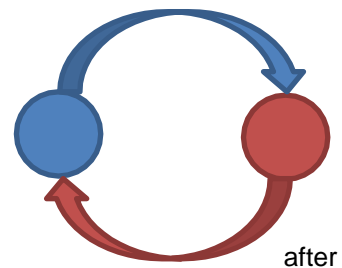
All computer programs use these three categories of action to solve problems. We'll study them all in much more detail as the course progresses.

SEQUENCE

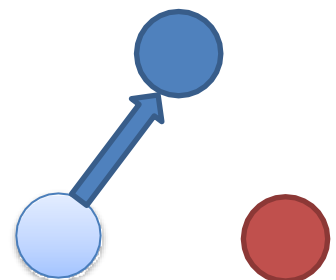
The most basic of the action categories, sequence simply means doing one thing after another. If you had a blue ball and a red ball, and you wanted to swap their positions, we instinctively know that you just *swap* them!

A computer, however, can't think – it can only follow explicit, step-by-step instructions. So you have to tell it to execute three instructions, one the other (**sequence**) to swap the balls.

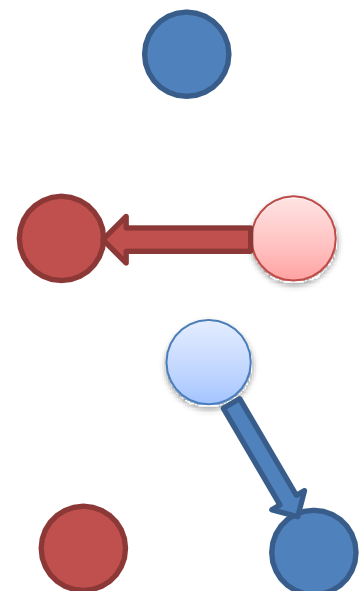
1. Move the blue ball to a third place



2. Move the red ball to the blue ball's old place



3. Move the blue ball to the red ball's old place.



This may seem complex, but remember that computers can't *think*. They can only follow instructions exactly as you tell them.

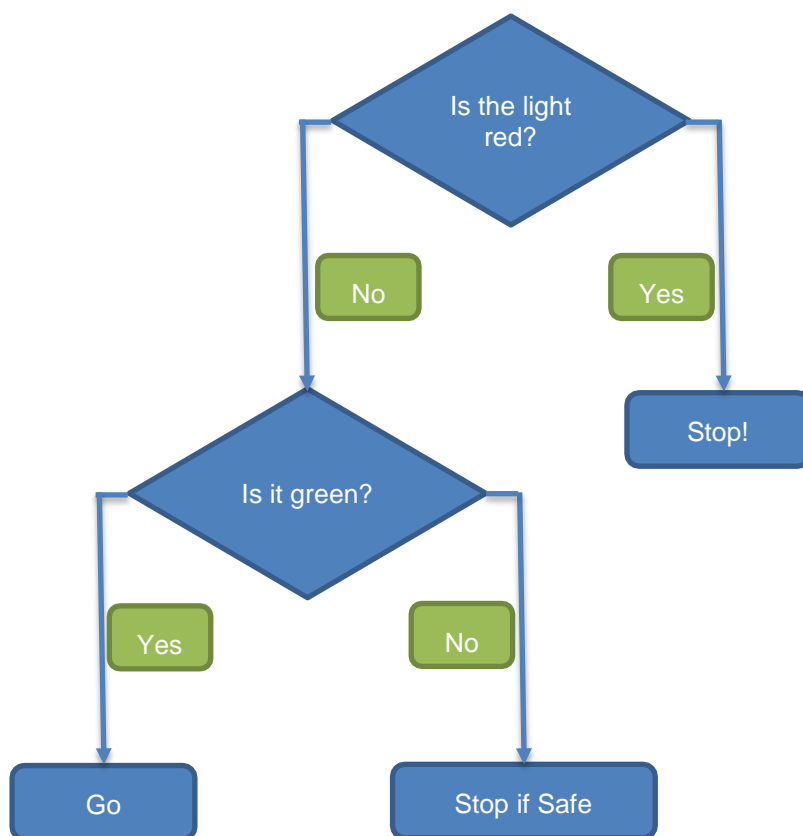
SELECTION

One of the most basic actions a computer can take is to **evaluate** a condition and return a **true or false** (boolean) response. For example, if you are playing a computer game and you shoot your gun at an opponent, the action boils down to the program (the game) asking: “Did the player hit the target?”.

If the player did (“True”), then something happens; the enemy falls down, an animation of some blood might appear, etc.

If the player missed (“False”), then something else happens; the enemy keeps going, a bullet hole appears on the wall behind them, etc.

One of the simplest ways to represent this is with a flow chart. Below is a simple flow chart that represents the “conditional test” and potential results for a traffic light.

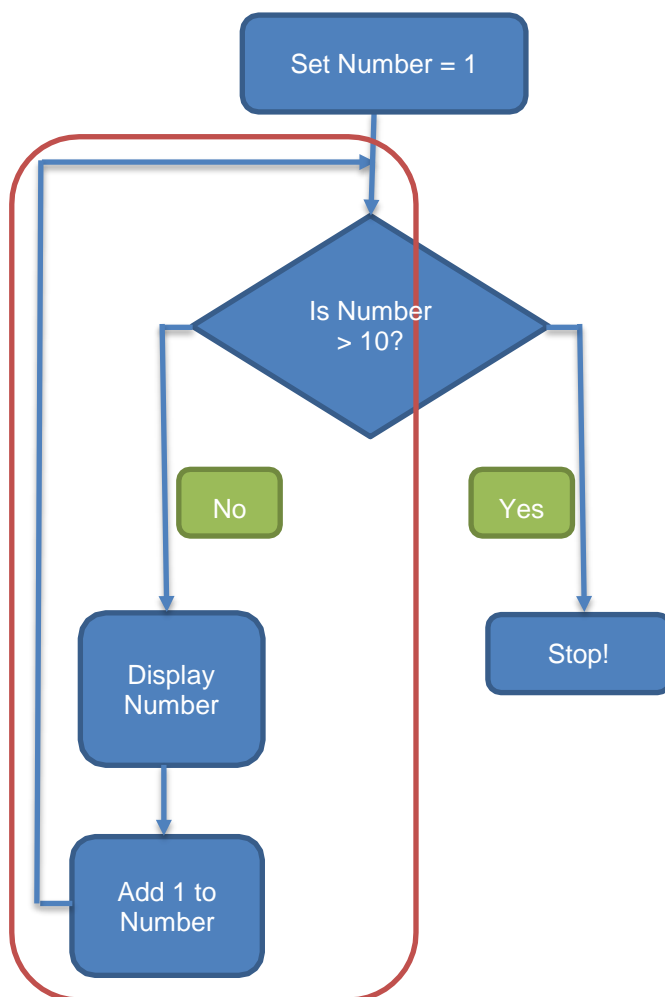


REPETITION

Also called “**Iteration**”, the third of the broad actions a computer can take is to repeat something very quickly until a condition is met. Being able to rapidly execute instructions is what makes a computer *seem* smart.

In programming, you’ll often see this referred to as “looping”, because it involves the computer testing a condition, performing some action, and then going back and retesting the condition again.

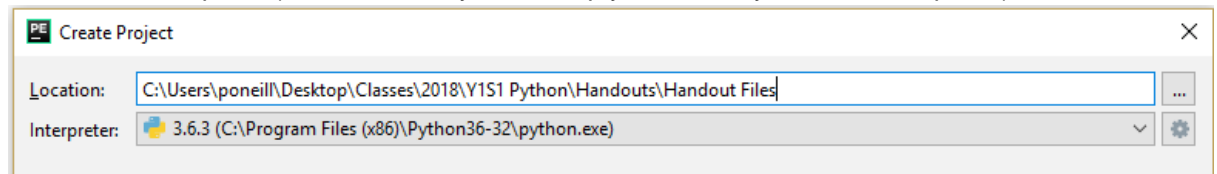
Below is a flow chart which shows a loop that tests to see if a number is below ten – if it is, the computer will display it; if not, it will end the program. So it will display the numbers 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 on the screen. It is a very simple loop, and it relies on a selection test to begin (as described above). The actual **iteration** section of the flow chart is outlined in red.



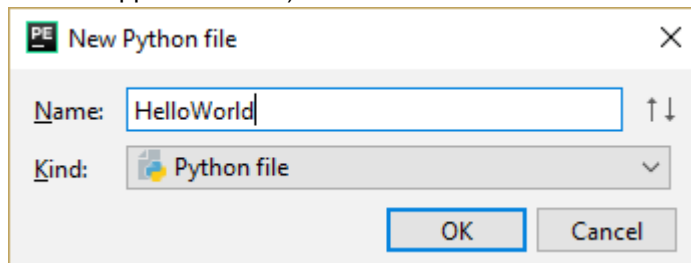
3 – Writing a Simple Python Program

There is a bit of tradition with learning a programming language, where the first program you right will display the words “Hello World!” to the screen. It’s commonly called a “Hello World” program, so we’ll continue it and create a simple Python program to do just that.

1. Open PyCharm Edu and select “New Project” (either from the opening dialog or from the “File” menu, depending on where your program starts)
2. Create a folder to hold your Python projects and call it “**Handout Files**”. Your removable hard drive is the best place (but make sure you back up your files to your home computer!)

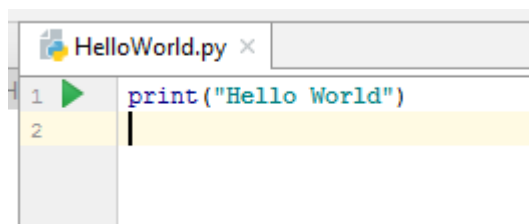


3. Click “Create”
4. Right-click on the project file and select **New --> Python File**
5. Enter the name “HelloWorld” in the project name (note the lack of space between “o” and “W”, and the upper-case “W”)



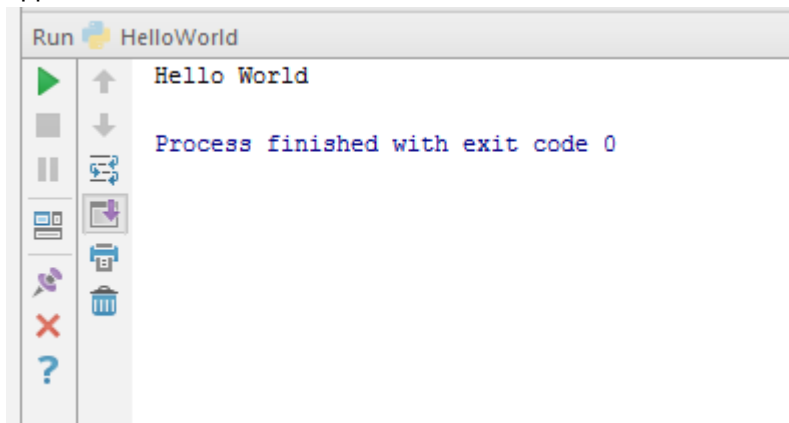
6. Click “OK” and you will get an empty program called “HelloWorld.py” (.py is the file extension for Python files, like “.docx” is a Word document or “.exe” is for an executable program)
7. Enter the following code *exactly* as written into the editor (the editor will highlight the words in different colours – don’t worry about that for now)

```
print("Hello World!")
```



8. Click on the green “play” arrow, and you should get a “Run” window open at the bottom of the screen which displays the output of the program, along with an “exit code”. Again, don’t worry

about the exit code for the moment – the important thing is that the words “Hello World!” appear in the window.



9. If you get an error message, go back and make sure you’ve entered the code **exactly** as shown, with no spelling errors or extra spaces. Make sure the parentheses are closed, as well as the opening and closing quotation marks (“”). If you still have an error, ask your teacher for help.

Okay – you’ve written your first Python program. It’s not quite going to run the next Mars rover, but it’s a start. It’s also a good spot to look at some of the structures within Python that you’ll be using all the time.

The print() function

A lot of programming relies on **functions**. We’ll be using them all the time – and later on even writing our own – but for the moment, the most important thing to know about them is that a function *does something*. We don’t need to know all the ins-and-outs of *how* it does it; we just need to know that when we **call** it (or **invoke** it), it will perform some sort of work for us.

In this case, there is a function built into Python called `print()` (note the parentheses after the function name. Even if there is nothing in there, we still always put the empty parentheses there to tell everyone that this is a function).

The `print()` function is pretty complex. It takes whatever is between the parentheses, talks to the computer’s operating system and tells it to take that information, send it to the graphics processing unit (the GPU, or video card), and then tells the GPU to display that information on the computer’s monitor. That’s all pretty complicated – but we don’t need to worry about *how* it does this. We just need to know that if we call it and give it the information we want (“Hello World!”) that it will display this on the screen.

By leaving the parentheses empty, we can print a blank line to the screen.

Literals

In this sense, we are using what programmers call a **literal**. Whatever we put between quotation marks inside the `print()` function will be *literally* displayed on the screen. A **literal** is often used in a program as a “placeholder”, and is often a **character string** (such as “Hello World!”); literally, a string of characters, one after the other.

We can also use numbers as a literal, though we don't put the quotations around a number.

Go back to your "HelloWorld" program and change the contents of the `print()` function so that it reads:

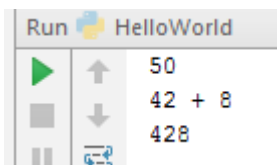
```
print(42)
```

Now run the program. What was the program's output? The number 42, right?

Okay, nothing that surprising there. But here's the interesting bit. Try running the following two bits of code and see what you get as a result:

```
print(42 + 8)
print("42 + 8")
print("42" + "8")
```

Your output should be:



The reason is that the first line of code was treating the numbers as *values*, so when you put the "+" sign in there, the program assumed you wanted them added together. The second line, however, treated the entire section between the parentheses as a **string literal** and displayed it.

The third line is where it gets *really* interesting. Here, the program saw not the values 42 and 8, but rather the **character strings** "42" and "8", and instead of adding the values, it **concatenated** the strings together to give you "428".

This can be useful when you want to combine different things together, as we'll do in some of the exercises.

New Lines

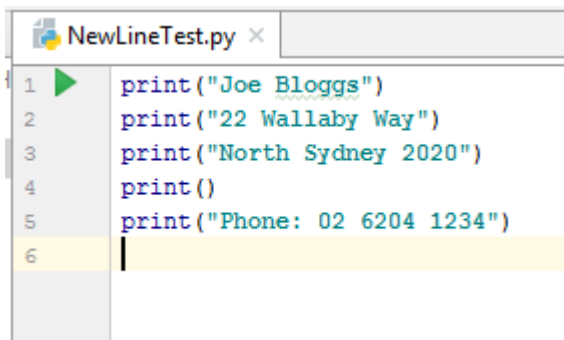
There's not much point printing lines to the screen if we can't put a new line in. Up above, we've seen one way of doing that, by using an empty `print()` function.

Let's create a new Python program to test a few things.

Create a new Python program called "NewLineTest". Go to **Step 3 – Writing a Simple Python Program** above if you can't remember how.

Let's create a person's name and address, with some space between some of the lines of text, all using the `print()` function.

Enter the following code *exactly* as shown:



```
1 print("Joe Bloggs")
2 print("22 Wallaby Way")
3 print("North Sydney 2020")
4 print()
5 print("Phone: 02 6204 1234")
6
```

Your output should be:

```
Joe Bloggs
22 Wallaby Way
North Sydney 2020

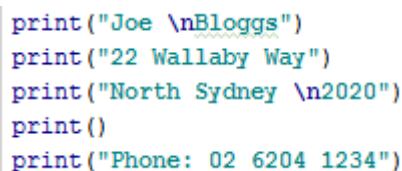
Phone: 02 6204 1234
```

Note that each `print()` gives us a new line, and that the empty `print()` gives us a blank line between the town and phone number.

We don't *have* to use empty `print()` to give a new line, however. Sometimes it's handy to be able to create a line break *inside* a string literal. To do that, we use a special character, "n", which stands for "new line". The problem is that if we just put an "n" inside a `print()` function, Python will just display the "n" character. So we use another special character, called an **escape character** ("\\") to tell the program that the character *immediately following it* is not a normal part of the character string.

If we wanted to use the same program as above, but we wanted to put "Joe" and "Bloggs" on two different lines, and split the town and post-code, we would use the escape character ("\\") immediately followed by "n" to tell it where to break.

Try it:



```
print("Joe \\nBloggs")
print("22 Wallaby Way")
print("North Sydney \\n2020")
print()
print("Phone: 02 6204 1234")
```

(note how PyCharm has changed the escape character and newline character to blue? That's called "syntax highlighting" and makes programming a little easier)

You should get the following output when you run this program:

Joe
Bloggs
22 Wallaby Way
North Sydney
2020

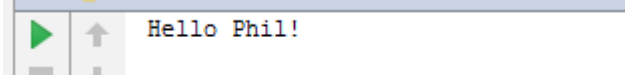
Phone: 02 6204 1234

There is no hard-and-fast rule on when to use an empty `print()` function or a `\n` escape character – it will depend on your program and how you have constructed it.

Now we're going to run through some basic exercises in Python. You will need to create the ".py" files as directed in your "Handout Files" project.

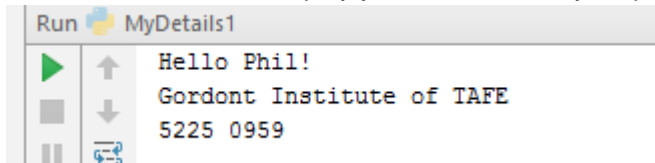
4 – Simple Python Exercises (expected results are shown)

1. Create **MyName.py**, basing it on the “HelloWorld” program.
Change the message from “Hello World!” to your name.



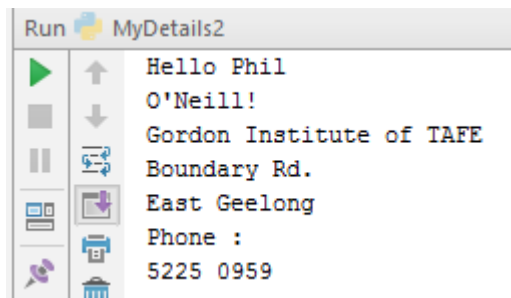
```
▶ ↑ Hello Phil!
```

2. Create **MyDetails1.py**
Add two more lines to display your address and your phone number.



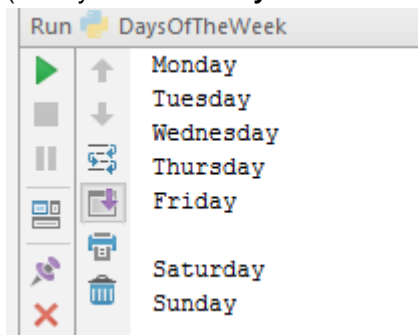
```
Run MyDetails1
▶ ↑ Hello Phil!
  ↓ Gordont Institute of TAFE
  ↓ 5225 0959
```

3. Create **MyDetails2.py**
Add a new line character so that your first and second names show on separate lines.
Add a new line character so that your address appears on multiple lines
Add a new line (with `print()`) so that your phone number appears on a separate line from the word “Phone”.



```
Run MyDetails2
▶ ↑ Hello Phil
  ↓ O'Neill!
  ↓ Gordon Institute of TAFE
  ↓ Boundary Rd.
  ↓ East Geelong
  ↓ Phone :
  ↓ 5225 0959
```

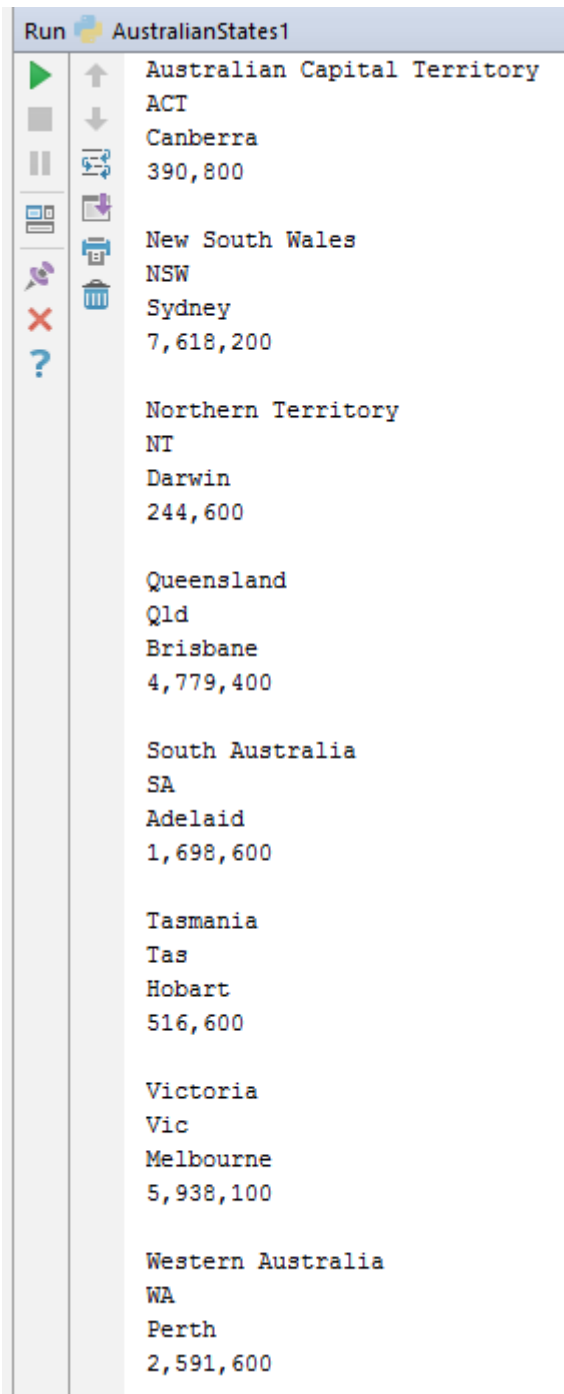
4. Create **DaysOfTheWeek.py**
This program should display the weekdays in one `print()` line, and the weekend days in a second `print()` line. New line characters should ensure that the days are on separate lines.
(Hint: you should **only** have two lines of code)



```
Run DaysOfTheWeek
▶ ↑ Monday
  ↓ Tuesday
  ↓ Wednesday
  ↓ Thursday
  ↓ Friday
  ↓ Saturday
  ↓ Sunday
```


5. Create **AustralianStates1.py**

This program should have a separate `print()` for each state, showing the State name in full, the short name, the capital city, and the population (get the population from the screenshot, below). Place the details within each state on separate lines of output using the `\n` character.



```
Run AustralianStates1

Australian Capital Territory
ACT
Canberra
390,800

New South Wales
NSW
Sydney
7,618,200

Northern Territory
NT
Darwin
244,600

Queensland
Qld
Brisbane
4,779,400

South Australia
SA
Adelaide
1,698,600

Tasmania
Tas
Hobart
516,600

Victoria
Vic
Melbourne
5,938,100

Western Australia
WA
Perth
2,591,600
```

Debug Exercises

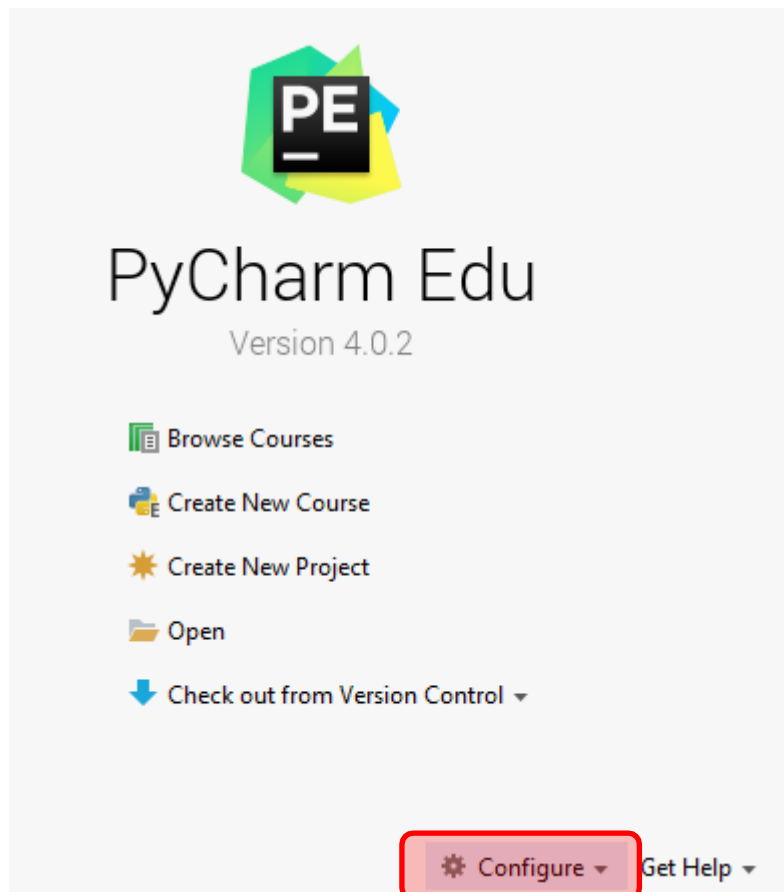
There is a file on the Gordon Online site for “Section 1” called “**DebugExercises01.zip**”. Unzip the files in it into your “Handout Files” project folder and open them in PyCharm. Have a look at the code and try to run them, then see if you can figure out what is wrong, correct the errors, and successfully run the programs (**FixErrors1.py** and **FixErrors2.py**)

Appendix

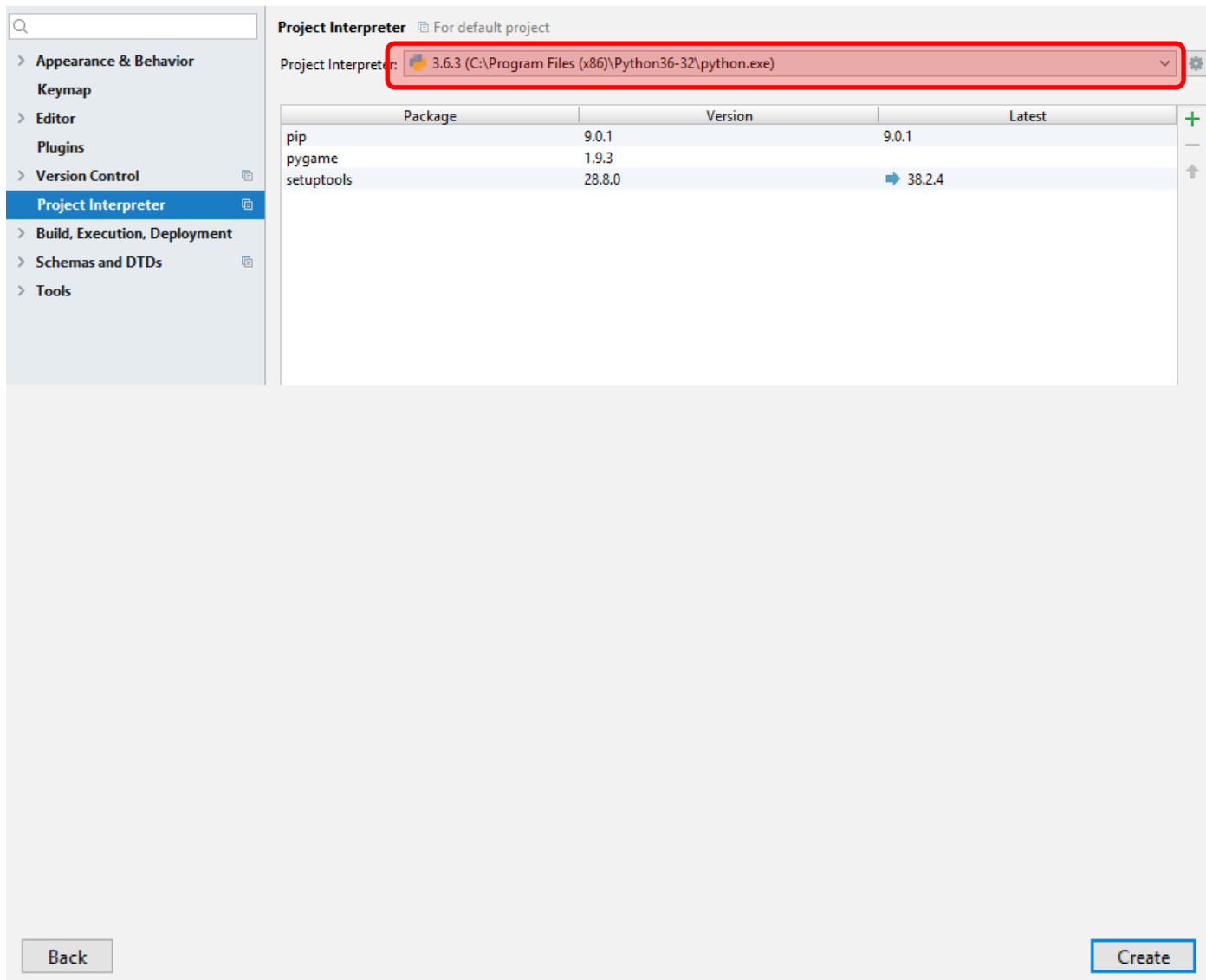
Configuring PyCharm Edu

When installing PyCharm Edu, you will need to link it to the Python interpreter (where you installed Python).

When PyCharm Edu is installed and started, you need to select “Configure --> Settings”

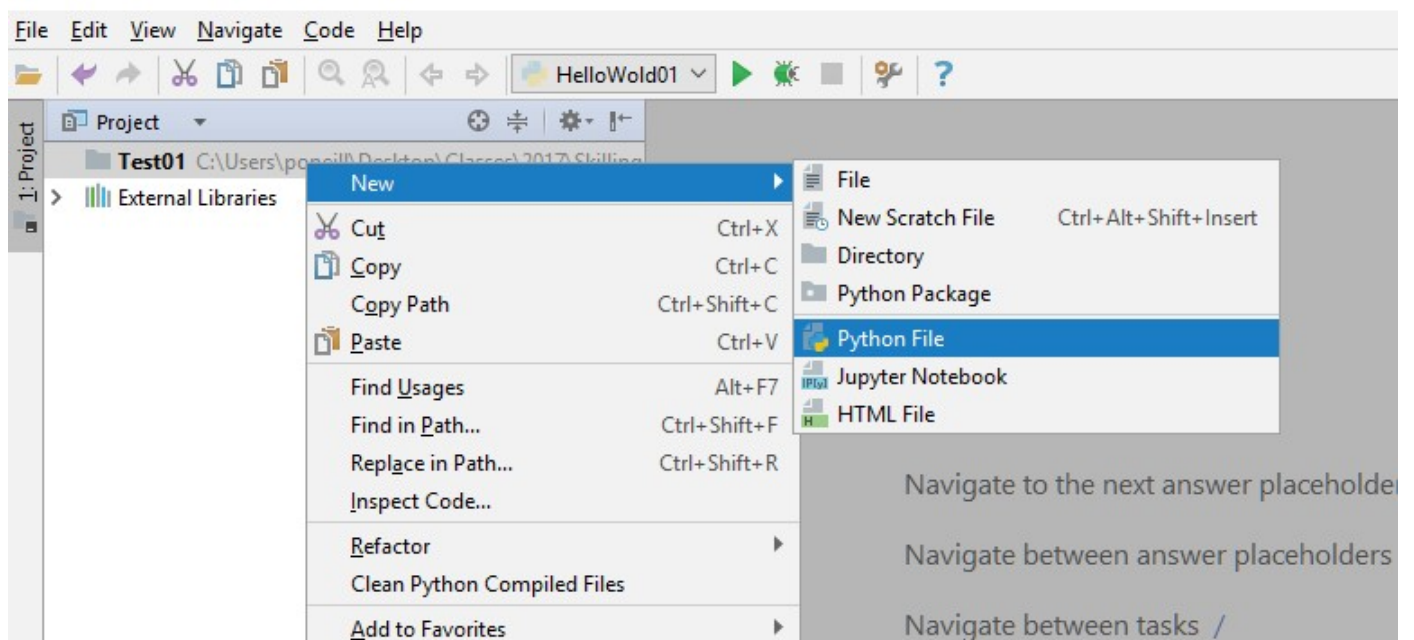


Select “Project Interpreter” on the left, then in the drop-down box, select the python.exe file in the folder you set up when installing Python, then click “OK”. This will pick up and attach the interpreter.

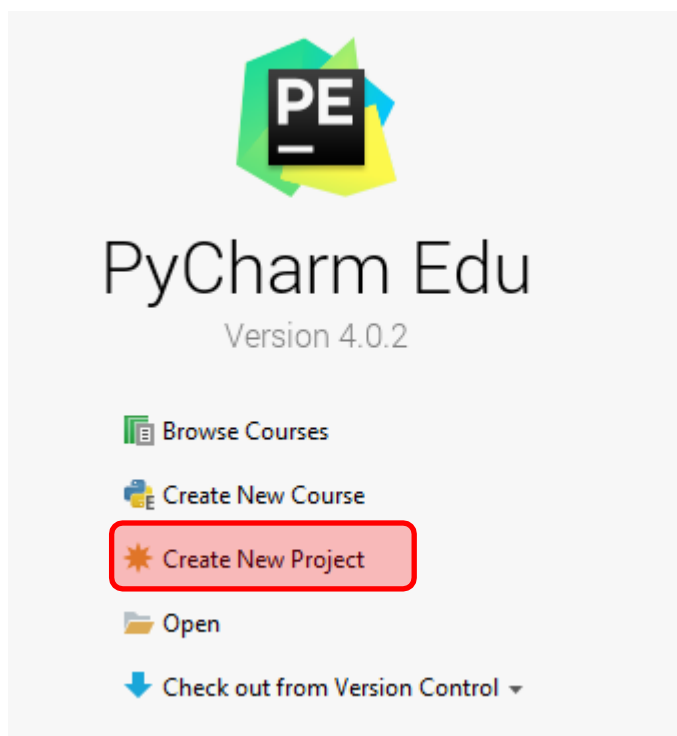


Click “Create”

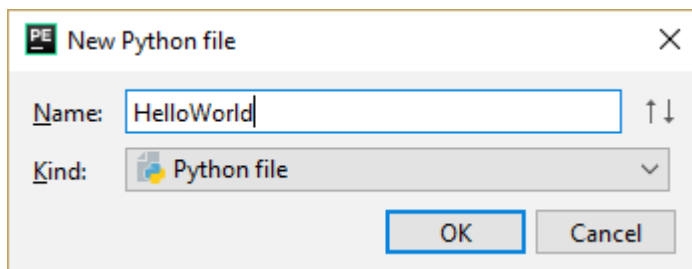
Right-click on the project name (“Test01” in this example) and select “New --> Python File” as shown



(or select “Create New Project” from the startup screen).



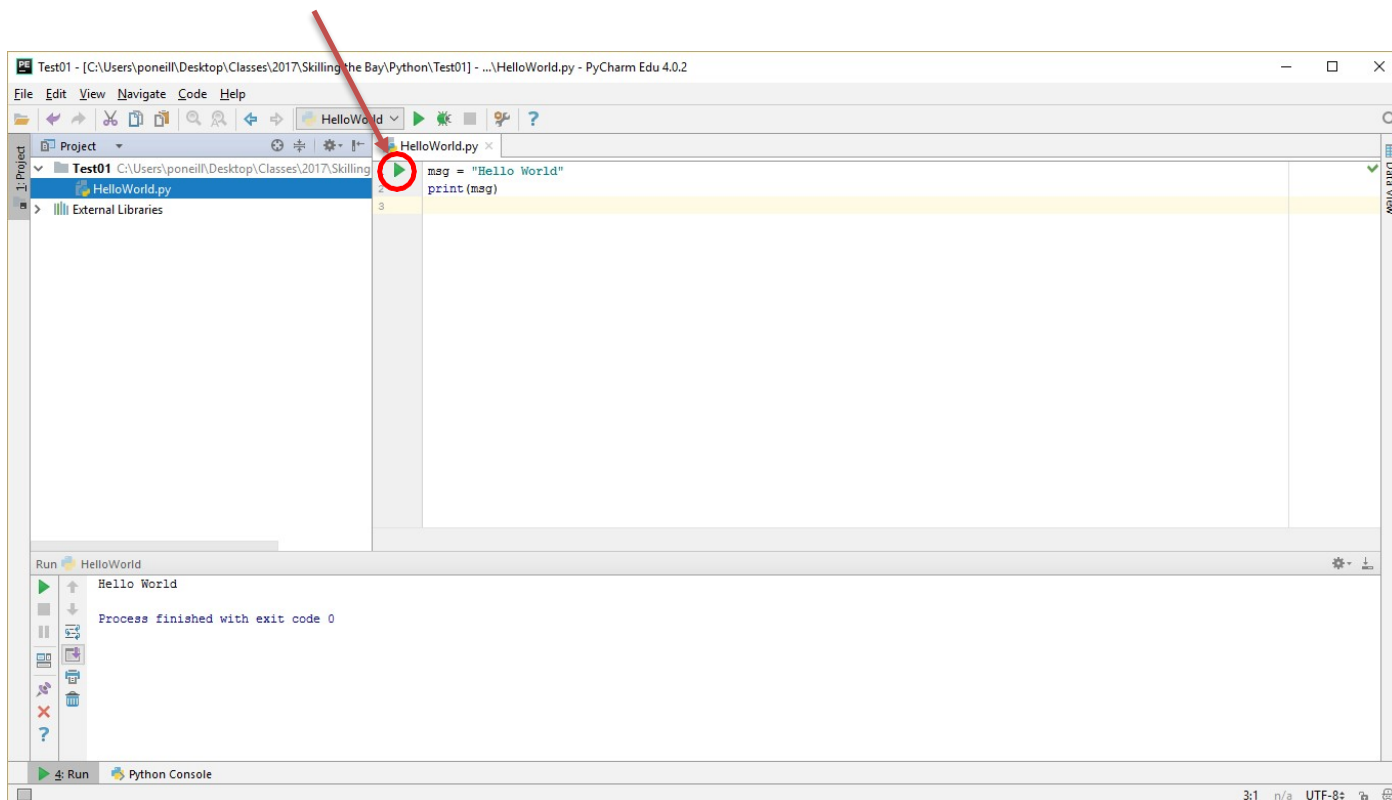
Name the file “HelloWorld”



Enter the following code into the editor

```
msg = "Hello World"
print(msg)
```

Click the little green “Play” button, and if the program executes (you should see “Hello World” appear in the “Run” toolbar at the bottom), the setup is complete and correct.



Copyright Information

Figure 1

Arnold Reinhold; <https://commons.wikimedia.org/wiki/File:FortranCardPROJ039.agr.jpg>

Figure 2

This image, which was originally posted to *Flickr.com*, was uploaded to Commons using [Flickr upload bot](#) on 18:21, 4 June 2012 (UTC) by [Jan Arkesteijn \(talk\)](#). On that date it was licensed under the license below.



This file is licensed under the [Creative Commons](#) Generic license.