Python Tutorial Part 1

From

```python
# obligatory 'Hello World' when learning a new language
print("Hello World")

# Tell the user to press Enter to quit
# Only needed if running in a console to stop it closing, NOT in IDE
input("Press Enter to quit")
```

To

```python
def main():
      player = Player()                # create the player
      player = update_player(player)   # ask user to input name/character
      display_player(player)           # display player properties
      items = create_items()           # create game items
      locations = create_locations()   # create locations
      current_location = "Classroom"   # set current location
      play(player, locations, current_location, items)

      input("Enter to quit")

main()
```

Python Tutorial – Part 1

Note code has been copy/pasted from Visual Studio 2022 to obtain coloured text and the colour highlighting does NOT match Wing Personal IDE.

You will be following this tutorial as one of the following:

1. I have never done any text-based coding.
2. I have used Lua.
3. I have used C#, Java or other languages.
4. I am an expert in everything: Show me something new.

This tutorial has been designed to cover all the above, so references to Lua/C# and other language used for comparison will be made throughout.

Python code can be written using Notepad or any text editor, but in school environments running the scripts can be difficult, as you do not have access to the command line. Sometimes an association with the .py file extension will run the Python console, but may instead open the dreadful Idle built-in IDE.

It is better to use a proper IDE (Integrated Development Environment).
One of the best is a free application called Wing Personal which can be downloaded from https://wingware.com/downloads/wing-personal

The first thing to get used to when coding in any language is to **stop double-clicking** on your files to launch the editor. This may work for Word, Excel or simple text files, but will not work for coding. At best this will run the file, at worst you will get Windows asking you how it should handle the file type.

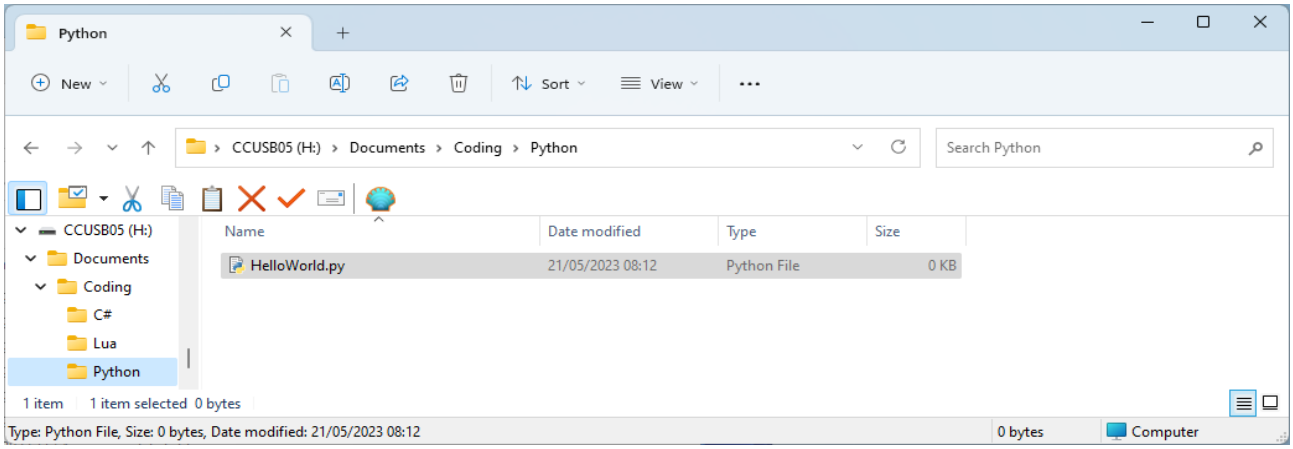
The second thing to get used to is organising your coding files properly. This makes finding them easier, and most IDEs in most languages allow you to select a folder and edit or run any of the files within it.
As you progress with coding skills, you will find you are using 'projects' or 'solutions' which consist of multiple files and folders, so it is a good idea to start with that principle.

Organising your files and folders

Suggested scheme to get started:

1. Create a new folder in Documents called ‘Coding’
2. Create a new folder inside Coding called ‘Python’
3. Optional (for use in part 2):
4. Create a new folder inside Python called ‘Pygame’

These folders will be empty initially, until you write new files from the IDE. The “HelloWorld.py” file seen below is an example of this.



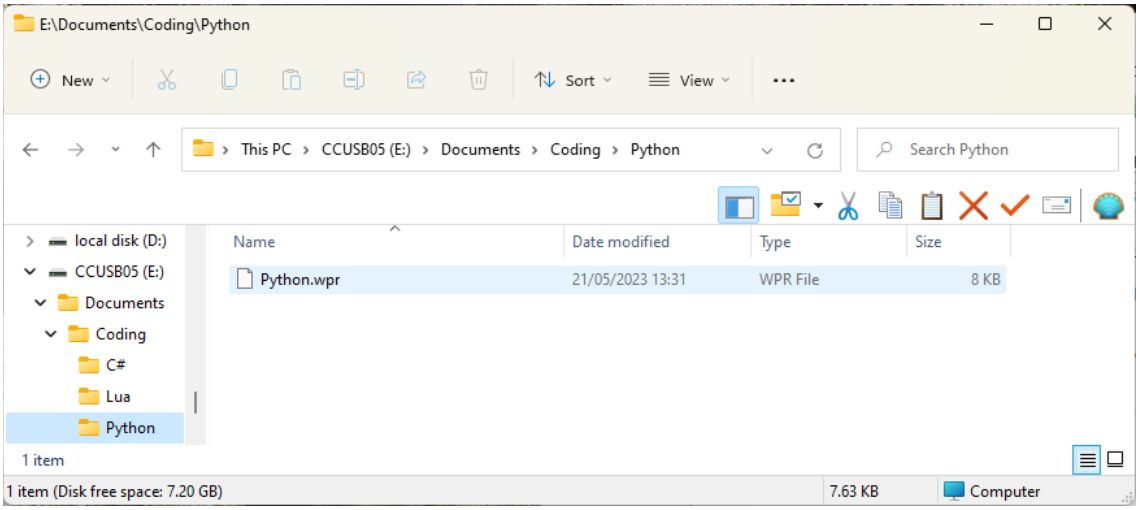
This tutorial has a number of files associated with it, which can be found at:

https://github.com/Inksaver/PythonForSchools

These files have links when they are referred to in this tutorial.

Setup Wing Personal as described in the tutorial at the end of this document.

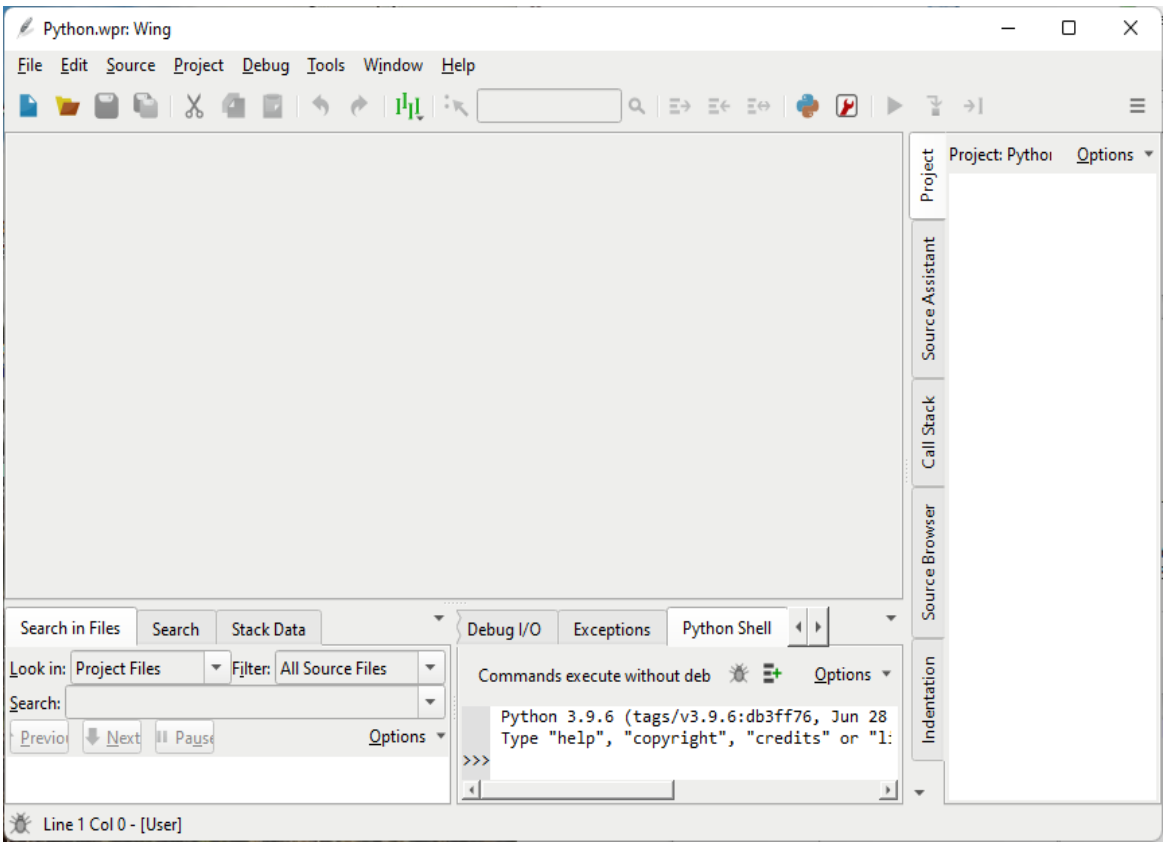
You should have a Wing project file in your Python folder: eg Python.wpr



Start Wing Personal

Refer to the Wing tutorial at the end of this document for setup details.

Use the Menu Project → Open Project and find your Wing .wpr project file. If the folder is empty it will look like this:



Direct Python Commands

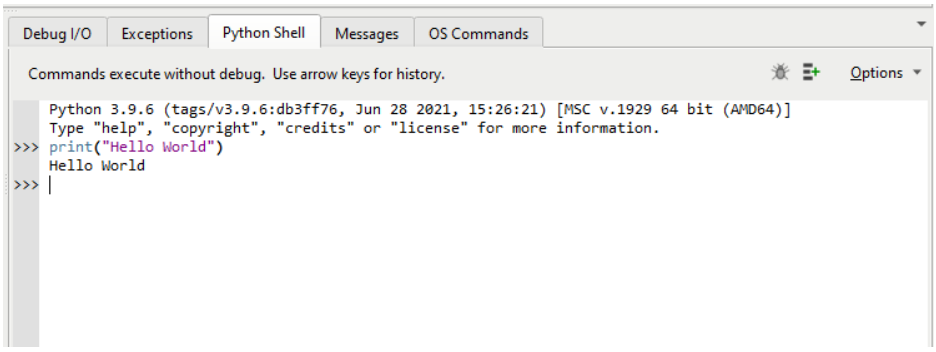
If you have used Python before, you will be familiar with the Idle 'Shell' which allows you to type commands in one at a time.

If you have used Lua with Zerobrane Studio, you will probably have used the Lua shell. It works the same way with Python

The same can be done within Wing.
Click on the 'Python Shell' tab.

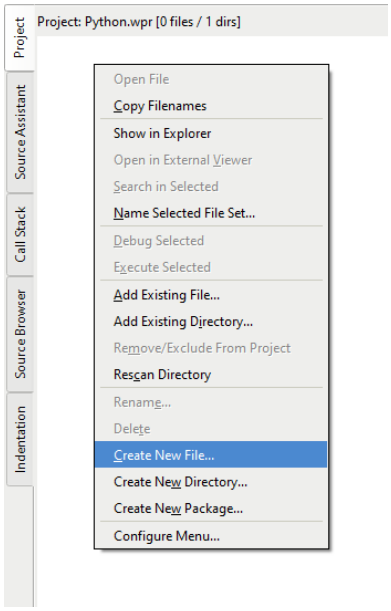
Type: `print("Hello World")`

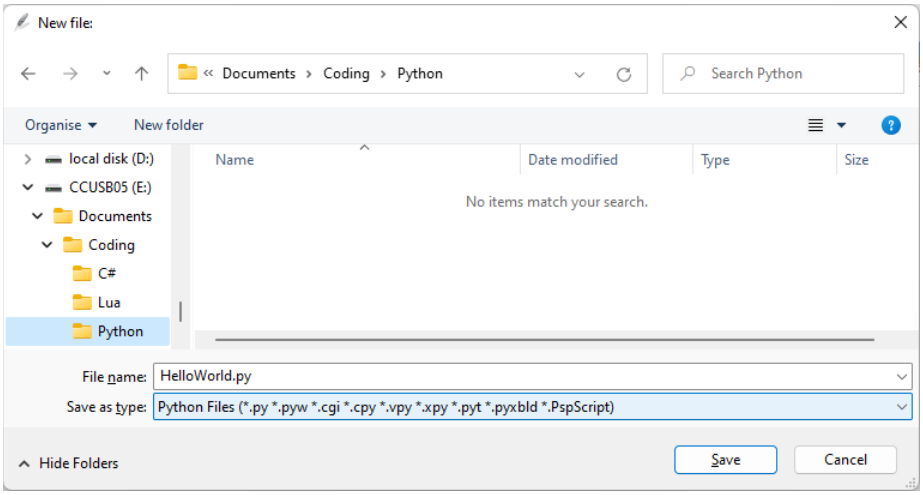
Press enter. The words “Hello World” appear in the console:



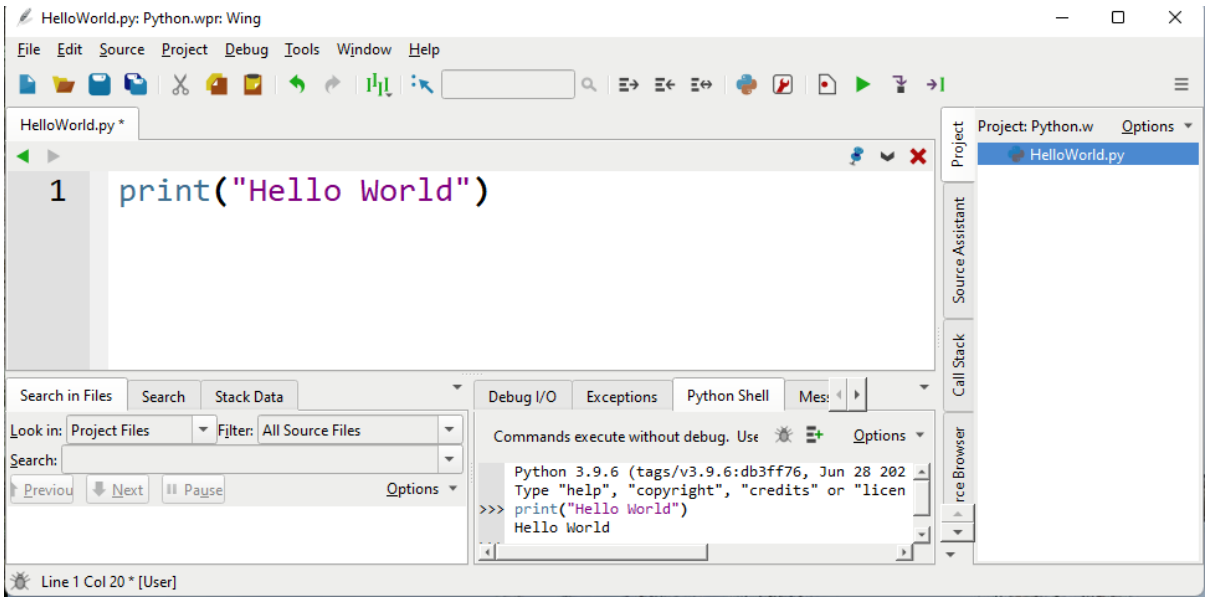
The line is temporarily saved in memory, and can be recalled using the up arrow on the keyboard, but to save it permanently it needs to be saved to a file, exactly the same as Lua.

Right-Click anywhere in the Project tab and select ‘Create New File’. Type “HelloWorld.py” into the empty box and press ‘Enter’:





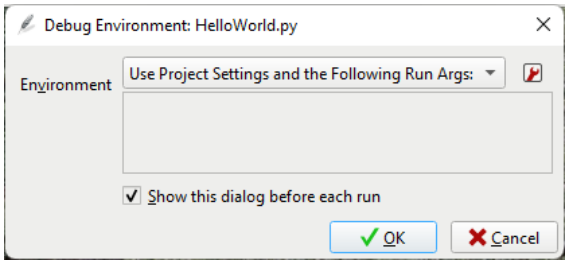
Type: `print("Hello World")` into the Editor window:



Click the green triangle on the toolbar, or press F5. This will automatically save the code and run it.

If you get this:

Un-tick the box and click OK



If you followed the setup, it will open in an external Python console, but...
You will see a black console window flash briefly and disappear.

Well done. You have run your first script. It is only one line, but a script can be hundreds or even thousands of lines. In order to see the results, you have to stop the console closing

See Section 1 below to fix the disappearing console

Python (and Lua) are interpreted languages. They read your script line by line, and carry out the instructions you wrote. In this case it 'print' s the text 'Hello World' to the console (screen).

Section 1

https://github.com/Inksaver/PythonForSchools/tree/main/Section1

01-HelloWorld1.py:

https://github.com/Inksaver/PythonForSchools/blob/main/Section1/01-Hello%20World1.py

```
# obligatory 'Hello World' when learning a new language
print("Hello World")

# Tell the user to press Enter to quit
# Only needed if running in a console to stop it closing, NOT in IDE
input("Press Enter to quit")
```

02-HelloWorld2.py:

https://github.com/Inksaver/PythonForSchools/blob/main/Section1/02-Hello%20World2.py

```
# demonstrates use of a function called main() to copy C, C++, C# and Java
# functions are ignored until 'called'

def main():
      print("Hello World Version2")

main() # 'Call' function main()

input("Press Enter to quit")
```

It works exactly the same as before, so why bother?
Procedural programming is much more efficient and can save you a lot of typing.

As your script grows in size, using functions (and procedures) makes it easier to maintain. As the only difference in practical terms between a function and a procedure is that a function returns at least one value, this tutorial will use the word function to cover both.

More advanced languages such as C# and Java start with a function/procedure called main() so it is a good idea to start using a similar function in Python (and Lua).

When the interpreter reads your modified script this time, it ignores any functions, but makes a mental note where they are, so when your script uses or 'calls' them, it knows where they are.

Python uses 'def' to def(ine) a function.

Lua uses 'function'

When it reaches line 5 (main()), this is a call to the main() function found at line 1, so the script jumps from line 5, which is the starting point, to line 1, the main() function.

The main() function has only one line of code: `print("Hello World")`, which it executes, then control is returned back to line 5, which happens to be the end of the program.

All further examples and exercises will use a main() function.

Printing to the screen

Displaying text either to a panel within the IDE, or to an external console is usually achieved with the `print()` procedure, as you have already used.

The `print()` procedure automatically inserts a "newline" character which forces the output cursor down to the next line, ready to `print()` again.

Python's input() function can take a second parameter, which cotrols whether the newline character is used. If you over-ride it with an empty string end="" then the cursor stays at the end of the last output to the screen.

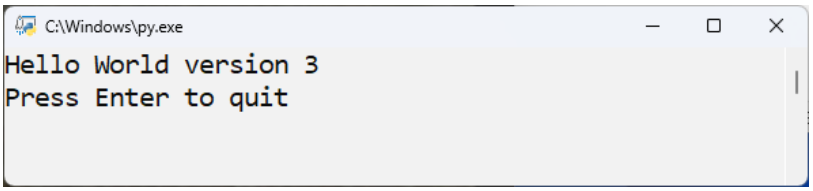
The next 3 files demonstrate this, along with the use of "escape characters" which help with the formatting of output.

03-HelloWorld3.py:

https://github.com/Inksaver/PythonForSchools/blob/main/Section1/03-Hello%20World3.py

```
# demonstrates the Python equivalent of Lua's io.write(). All output is on 1 line
# use F7 to step through in IDE

def main():
      print("Hello ", end = "")
      print("World ", end = "")
      print("version 3")

main()

input("Press Enter to quit")
```

Console output:
Note how all text is on one line

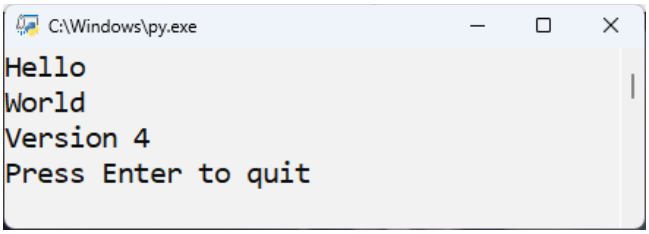


04-HelloWorld4.py:

https://github.com/Inksaver/PythonForSchools/blob/main/Section1/04-Hello%20World4.py

```
# demonstrates print() with \n newline character
def main():
        print("Hello \nWorld \nVersion 4")

main()

input("Press Enter to quit")
```

Console output:

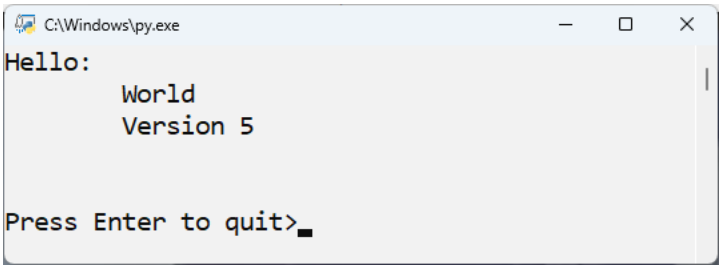


05-HelloWorld5.py:

https://github.com/Inksaver/PythonForSchools/blob/main/Section1/05-Hello%20World5.py

```
# demonstrates print()
# with \n newline and \t Tab

def main():
        print("Hello:")
        print("\tWorld ")
        print("\tVersion 5")

main()

input("\n\nPress Enter to quit>") # note 2 newlines
```

Console Output:



The backslash before n represents the newline character. Similar hidden characters you can play with are:

```
1. \'    single quote:    embed a single quote in the output string
2. \"    double quote:    embed a double quote in the output string
3. \\    backslash:       embed a backslash in the output string
4. \n    new line:        move cursor to a new line
5. \r    carriage return: move cursor back to the beginning of the line
6. \t    tab:             move cursor forward (usually) 4 spaces
7. \b    backspace:       move cursor back one space
8. \f    form feed:       move cursor down to next line
```

Many of these 'Escape Characters' are based on the actions of mechanical typewriters.

The typewriter had a tab key, which moved the 'carriage' (the roller that held the paper) to the left by 4 spaces, so the next character position was effectively moved 4 spaces to the right.

There was also a 'backspace' key which moved the carriage one space to the right, so you could then insert a thin piece of white tape to over-print the incorrect character, and allow you to type the correct one. The resulting mess was acceptable at the time.
The console backspace works in the same way: it moves the cursor back one space, but does NOT delete the character in front of it.

There was a prominent lever which did two things:

1. Carriage Return: which pushed the carriage fully to the right, so the left edge of the paper was under the print head

2. Line Feed (Form feed) which rotated the roller to move the paper upwards, so the print head was on a new line.

The newline character \n is effectively a combination of 'carriage return' (cr) and 'line feed' (lf) so it moves the virtual carriage to the left edge of the console, then moves it down by one line.

Windows uses both cr and lf to move to the next line, Unix, Linux and others use just use one of them, so 'newline' covers all operating systems.

UTF8 characters

Available from https://www.w3schools.com/charsets/ref_utf_box.asp

These characters can improve your console based projects in Lua, Python, C# and many other languages
They cannot be typed in, so are best copy/pasted into your code.

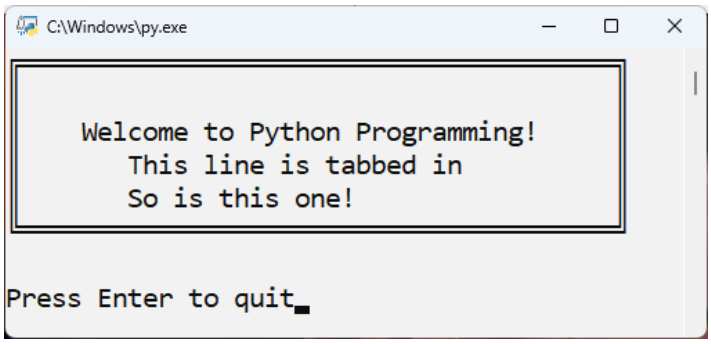
```
┌ ┬ ┐ ─ ╓ ╥ ╖ ═
├ ┼ ┤ │ ╟ ╫ ╢ ║
└ ┴ ┘   ╙ ╨ ╜
```

06-Assignment 1.py

https://github.com/Inksaver/PythonForSchools/blob/main/Section1/06-Assignment%201.py

```
'''
https://www.w3schools.com/charsets/ref_utf_box.asp
utf8 box characters stored here for easy copy/paste:

╒ ╥ ╕ ═ ╓ ╦ ╖ ═
╞ ╪ ╡ │ ╟ ╫ ╢ ║
╘ ╩ ╛   ╙ ╨ ╜

Use the template below to create an output that looks like one of these:
Note the triple single quotes used to surround multi-line comments

                                        Pre Windows 10 (1909) will not display UTF8
╔═══════════════════════════════════╗   ==========================================
╟                                   ╢   |                                        |
╟    Welcome to Python programming! ╢   |    Welcome to Python programming!      |
╟        This line is tabbed in x2  ╢   |        This line is tabbed in x2       |
╟        So is this one!            ╢   |        So is this one!                 |
╚═══════════════════════════════════╝   ++++++++++++++++++++++++++++++++++++++++++
'''

# Use print()
# use for example: print('=' * 8) to print '========'
# Can also use \t to line up the text
# The comment above is 5 lines of (42 characters)

def main():
       # your code goes here
       pass

main()

input("Press Enter to quit")
```

Use the print() function and escape characters to get this output. This one using UTF8:



You can use the UTF8 characters for fancy box construction:

https://www.w3schools.com/charsets/ref_utf_box.asp

Copy / paste the characters into your code.

They will NOT display in Windows console below version 10 (1909) except in ConEmu emulator.

Variables

Variables are memory locations reserved to hold some kind of data, and given a label to identify them. They must begin with a letter or an underscore character, and must not contain any spaces.

```
Examples:

myName = “Fred”
myAge = 15
isWorking = false
```

```
These DO NOT WORK!:

1myName (starts with a number)
!myName (starts with a !)
my Name (contains a space)
```

The first one ‘myName’ is the label, and it has been given the word ‘Fred’ as the data stored in it. Variables containing a string of letters or words are called ‘string’ variables

myAge contains a whole number. It is called an ‘integer’ variable

isWorking contains either true or false. It is called a ‘boolean’ variable

Lua and Python have a fairly flexible approach to variables. You can change the data stored in them from string to integer or boolean without any problem. (This is not the case with C# and Java)

Comments

As your script gets bigger, it is helpful to write in some comments to help others understand what you are doing, or to remind yourself if you come back to it after some time.

Single-line Python comments start with a #, C# and Java use //
(Lua comments are started with two hyphens: - - (no space between them)

Comments are ignored by the interpreter.

Multi-line comments start and end with ‘’’ or “””
(Lua uses --[[ ]]--at the beginning and end, C# and Java use /* at the start and */ at the end)

```
‘’’ This is a multi-Line comment.
     It allows plenty of room to make notes’’’

print(“Hello World”) # this line prints ‘Hello World’
```

User Input

Most programs need some sort of input from the user, either from the keyboard, mouse or other device such as a joystick.

Lua and Python cannot handle anything except the keyboard without using other libraries (pre-written code modules), so the next part of the tutorial is based on keyboard input.

When moving on to the Pygame library, then the mouse can be used. (Pygame or Tkinter in Python)
Python users will be aware of the input() function to get keyboard data.
It is usual to assign the keyboard input to a variable:

```
typedInText = input("Ask the user to type something")
```

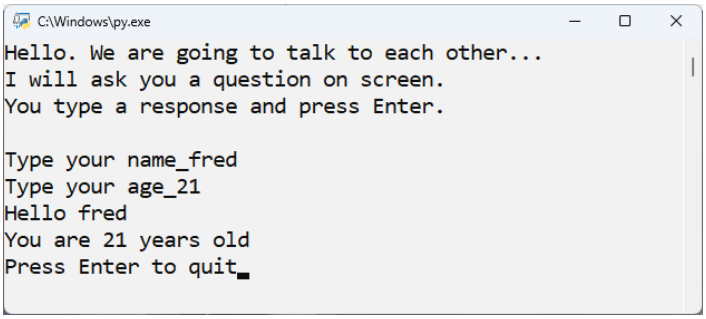
07-Input1.py:

https://github.com/Inksaver/PythonForSchools/blob/main/Section1/07-Input1.py

```
# introducing variables
# string concatenation (nerd word meaning join) using + (addition operator)

def main():
      print("Hello. We are going to talk to each other...")     # Spooky!!!
      print("I will ask you a question on screen.")
      print("You type a response and press Enter.")
      print()
      # Print a blank line

      name = input("Type your name_")  # input() does NOT move to the next line
      age = input("Type your age_")    # input is a function: it returns a value which
is stored in the variable age

      '''
      You can see we are using the input() built-in Python function
      var = input() # var is any variable such as name, age, height etc
      '''
      print("Hello " + name)
      print("You are " + age + " years old")              # the + joins words (strings)
together

main()

input("Press Enter to quit") # function used as a procedure (return value is discarded)
```

Note joining 2 strings uses the **+** operator.
This can cause problems as the same operator is used to add numbers.

Lua's double dot system is far better.



A much better way of concatenating strings is to use string interpolation. This is also known as the 'f string' The next file demonstrates this by putting 'f' in front of the opening single or double quotes used to surround a string.

You can then enter variables, numbers or even expressions inside the string surrounded by curly braces {}

08-Input2.lua:

https://github.com/Inksaver/PythonForSchools/blob/main/Section1/08-Input2.py

```
# does exactly the same as 07-Input1.py
# demonstrates use of string interpolation (f string)

def main():
      print("Hello. We are going to talk to each other...")
      print("I will ask you a question on screen.")
      print("You type a response and press Enter.")
      print()

      name = input("Type your name_")
      age = input("Type your age_")

      print(f"Hello {name}")                 # note the f in front of opening quotes
      print(f"You are {age} years old")      # note the {} braces containing numbers,
                                             variables or even expressions eg {var * 5}

main()

input("Press Enter to quit")
```

The only problem here is there is no validation of what the user is typing.
You might want a number, but that is not checked, or you might want a string of a particular length.

This can be addressed with input validation, which will be dealt with shortly.

09-Assignment 2.lua

https://github.com/Inksaver/PythonForSchools/blob/main/Section1/09-Assignment%202.py

Use this template code to produce the output below. You can also use UTF8 characters:

```
'''
https://www.w3schools.com/charsets/ref_utf_box.asp
utf8 box characters stored here for easy copy/paste:

┌ ┬ ┐ ─ ╒ ╤ ╕ ═
├ ┼ ┤ │ ╞ ╪ ╡ ║
└ ┴ ┘   ╘ ╧ ╛

Use the template below to create an output that looks like one of these: (either UTF8
or ASCII)

╔═══════════════════════════════════╗     ------------------------------------
║    Personal Data Collection App    ╕     |    Personal Data Collection App    |
╚═══════════════════════════════════╝     ------------------------------------
Please type your first name_              Please type your first name_
How old are you?_                         How old are you?_
What month is your birthday?_             What month is your birthday?_

═══════════════════════════════════     ------------------------------------

Thank you for sharing your data.          Thank you for sharing your data.
It will now be sold to scam websites.     It will now be sold to scam websites.
This is what we know about you:           This is what we know about you:

╔═══════════════════════════════════╗     ++++++++++++++++++++++++++++++++++++
║    Name:                          ╕     +    Name:                          +
║    Age:                           ╕     +    Age:                           +
║    Birth Month:                   ╕     +    Birth Month:                   +
╚═══════════════════════════════════╝     ++++++++++++++++++++++++++++++++++++

'''


# create 3 variables called name, age, month to store the data
# use print(), input()
# create a decorative output similar to the examples

def main():
      pass # your code starts here


main() # program starts here

input("Press Enter to quit>")
```

```
C:\Windows\py.exe

╔═══════════════════════════════╗
║    Personal Data Collection App    ╕
╚═══════════════════════════════╝

Please type your first name_fred
How old are you?_21
What month is your birthday?_december

═══════════════════════════════════
Thank you for sharing your data.
It will now be sold to scam websites.
This is what we know about you:

╔═══════════════════════════╗
║  Name:        fred         ╕
║  Age:         21           ╕
║  Birth Month: december     ╕
╚═══════════════════════════╝

Press Enter to quit>
```

Section 2

Validating input

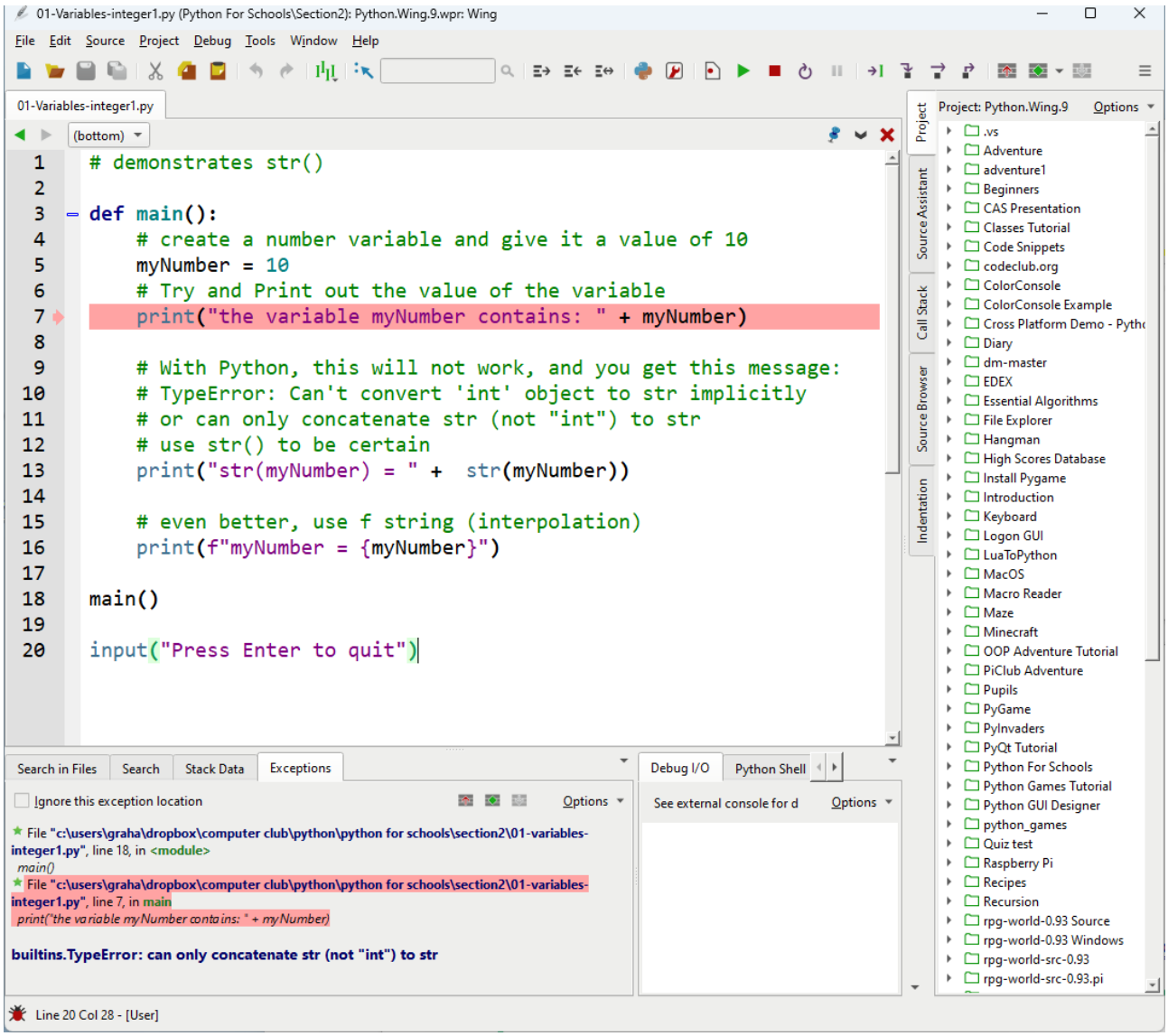
The next 3 files explain how to handle numbers and strings when input from a user, but there is still no checking. Errors will occur if the user types characters that cannot be converted to a number.

01-Variables-integer1.py

https://github.com/Inksaver/PythonForSchools/blob/main/Section2/01-Variables-integer1.py

```
# demonstrates str()

def main():
	# create a number variable and give it a value of 10
	myNumber = 10
	# Try and Print out the value of the variable
	print("the variable myNumber contains: " + myNumber)

	# With Python, this will not work, and you get this message:
	# TypeError: Can't convert 'int' object to str implicitly
	# or can only concatenate str (not "int") to str
	# use str() to be certain
	print("str(myNumber) = " +  str(myNumber))

	# even better, use f string (interpolation)
	print(f"myNumber = {myNumber}")

main()

input("Press Enter to quit")
```

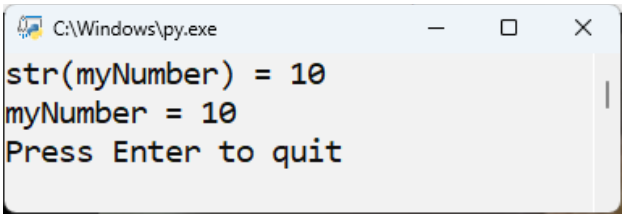
If you run the above code you get the error messages shown in the screenshot below:



Note the error message in the 'Exceptions' tab

You do not get this in Lua as it has a string concatenation symbol instead of using + there is no problem joining a string with a number. When compiled, the number is converted to a string automatically.

If you comment out line 7 and try again:



02-Variables-integer2.py

https://github.com/Inksaver/PythonForSchools/blob/main/Section2/02-Variables-integer2.py

This file demonstrates the use of `int(value)` as well as `str(value)`.
If you type in 'ten' when asked for a number, the result of `int("ten")` throws an exception (error)

Using `str(userInput)` when outputting the final print() ensures there is no error.

```
# int(), str()

def main():
	# Anything typed in is a string, INCLUDING numbers!
	# Ask the user to type something, and store what they typed.
	userInput = input("Type in any letters, numbers or symbols and press Enter")
	print("You typed in the characters " + userInput)
	# Ask the user to enter only numbers
	userInput = input("Type in any number, and press Enter")

	# If they typed in 3, it is the character "3"
	# You have to convert it to a number
	userInput = int(userInput) # if it cannot be converted it errors
	# ValueError: invalid literal for int() with base 10
	print("The variable userInput now contains:  " + str(userInput))

main()

input("Press Enter to quit")
```

03. Variables-float.py

https://github.com/Inksaver/PythonForSchools/blob/main/Section2/03-Variables-float.py

Similar to above, but using real numbers.

```
# demonstrate float and use of tostring() and tonumber()

def main():
	# Float variables
	myFloat  = 1.5 # create a float variable
	# Note the use of the tostring() function
	print("The variable myFloat contains: " + str(myFloat))

	# get a number from the user
	userInput = input("Type a number from 1 to 100 (program will crash if not a number)_")
	# convert userInput to number
	userInput = float(userInput) # if it cannot be converted it errors

	# ValueError: could not convert string to float:
	print("Your number: " + str(userInput) + " multiplied by " + str(myFloat) + " = " +
str(userInput * myFloat))
	# or use f string
	print(f"Your number: {userInput} multiplied by {myFloat} = {userInput * myFloat}")

main()

input("Press Enter to quit")
```

Conditional Statements

These are an essential part of any programming languages. The examples in Blue textboxes are NOT on Github!

The first keyword is `'if'`

You will be asked to use something called **pseudocode** during your studies. This is a code-like approach, but not in the syntax of a specific language. Here is an example:

```
if some condition is true then // pseudocode or Lua
    do some code
end
```

This code block will only 'do some code' if some condition checked in the line containing 'if' is true.

A more realistic pseudocode block is:

```
if userInput == "your name" then // pseudocode or Lua
    print("That is correct!")
end
```

The double == means 'Is the variable `userInput` equal to'

(The statement: userInput = would assign a value to the variable.)

Lua is the closest programming language to pseudocode. The block above is actually correct Lua syntax!

The second keyword is `'else'`.

This allows an alternative block of code to run if the original condition is not true.

You can check for an alternative condition by using `'elif'` (Lua uses `'elseif'`, C# uses `'else if'`)

```
if userInput == "your name":  # Python code
    print("That is correct!")

elif userInput == "name":
    print("That is half correct")
```

04-try.py – use of try: except:

https://github.com/Inksaver/PythonForSchools/blob/main/Section2/04-try.py

This file uses an if statement to check if the user entered a value that cannot be converted to a number.

If they did enter a number then the calculation is performed and output.

```
# demonstrate if statement to prevent crash

def main():
	# Float variables
	myFloat  = 1.5 # create a float variable
	# Note the use of the str() function
	print("The variable myFloat contains: " + str(myFloat))

	# get a number from the user
	userInput = input("Type a number from 1 to 100_")
	# convert userInput to number
	try:
		userInput = float(userInput) # if it cannot be converted it errors
		print(f"Your number: {userInput} multiplied by {myFloat} = {userInput *
myFloat}")
	except:
		print("You did not type a number")

main()

input("Press Enter to quit")
```

05-IfElseifElse.py

https://github.com/Inksaver/PythonForSchools/blob/main/Section2/05-IfElseifElse.py

```
# demonstrates boolean variables and if elif else
def main():
	# Boolean variables can only be either true or false
	# Most languages associate true = 1, false = 0
	# You can also think of yes = true, no = false

	choice = False # variable called 'choice' is given the default value False
	userInput = input("Do you like Python? (y/n)_")
	# user SHOULD have typed a 'y' or 'n'
	if userInput == "":                          # Enter only
		print("You only pressed the Enter key")
	elif userInput == 'y':                       # 'y' typed in
		print("Great! variable 'choice' is now True")
		choice = True             # set choice to true as the user typed 'y'
	elif userInput == 'n':                       # 'n' typed in
		print("Oh. That is disappointing")
	else:                                        # some other characters typed in
		print(f"You typed {userInput} I can't translate that to True/False")

	print(f"\nThe value of the boolean variable 'choice' is: {choice}")

main()

input("Press Enter to quit")
```

String Operations

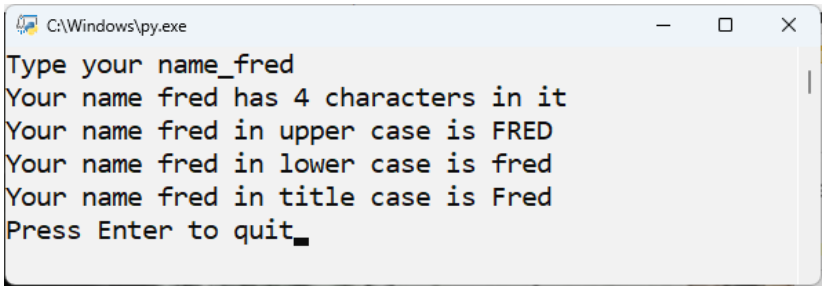
06-Strings.py

https://github.com/Inksaver/PythonForSchools/blob/main/Section2/06-Strings.py

This file demonstrates the string operators:

1. `len()` - returns the length of the string
2. `upper()` - returns the string in UPPER CASE
3. `lower()` - returns the string in lower case.
4. `title()` - returns the string in Title Case

```
# demonstrates  len, .upper(), .lower(). title()

def main():
        userInput = input("Type your name_")
        # len returns the length of the string inside the brackets as an INTEGER
        numberOfChars = len(userInput)
        print(f"Your name {userInput} has {numberOfChars} characters in it")

        # string.upper() converts all characters to UPPER CASE
        print(f"Your name {userInput} in upper case is {userInput.upper()}")

        # .lower() converts all characters to lower case
        print(f"Your name {userInput} in lower case is {userInput.lower()}")

        # .title() converts all characters to Title Case
        print(f"Your name {userInput} in title case is {userInput.title()}")

main()

input("Press Enter to quit")
```



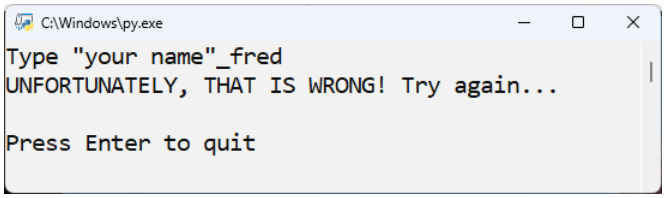
07-Strings2.py

https://github.com/Inksaver/PythonForSchools/blob/main/Section2/07-Strings2.py

The start of a very silly game, where the user has to type “your name” instead of their real name

```
def main():
	userInput = input('Type "your name"_')

	if userInput == "your name":	# user typed in 'your name' as instructed!
		print("That is correct!")
	else:	# user typed in their real name
		# use of f string with upper(), lower() or title() by using single
quotes
		print(f"{'Unfortunately, that is wrong!'.upper()} Try again...\n") #
shout at the user! (UPPER CASE)

main()

input("Press Enter to quit")
```

The downside is you only get one try. If you mess up you have to run it again.



To fix that, there needs to be a way of looping round and try again until the correct input is entered.

The next 4 files deal with loops.

Loops

08-Loop1.py Infinite while loop

https://github.com/Inksaver/PythonForSchools/blob/main/Section2/08-Loop1.py

```
# demonstrates use of the while loop
'''
 In the previous script, it only runs once. If you type in the
 words "your name" you get a brownie point. Otherwise you are wrong.
 But you can only do it once, then you have to re-start.
 Use a loop to allow another chance:
'''

def main():
    while True: # True is always True, so this is an infinite loop
        userInput = input('Type "your name"_')
        if userInput == "your name":   # user typed in 'your name' as instructed!
             print("That is correct!")
             break                     # break out of the loop
        else:                          # user typed in their real name
            print(f"{'Unfortunately, that is wrong!'.upper()} Try again...\n") # shout
at the user! (UPPER CASE)

    input("Press Enter to quit")

main()
```

The line `while True:` will repeat all the following lines up to the corresponding whitespace representing the end of the code block continuously, or until a break statement is encountered.

This is called an infinite loop because the while condition cannot change. True is always true.

09-Loop2.py Improved while loop

https://github.com/Inksaver/PythonForSchools/blob/main/Section2/09-Loop2.py

A more specific while loop could be used, where the condition being checked is what userInput contains:

```
# demonstrates while loop without break

def main():
    userInput = ""
    while userInput != "your name":    # != means 'is NOT equal to' -> while
                                         userInput is not equal to 'your name'
        userInput = input('Type "your name"_')
        if userInput == "your name":   # user typed in 'your name' as instructed!
            print("That is correct!")
            #break                     # break no longer needed. the loop will not
                                         run again as userInput is now = 'your name'
        else:                          # user typed in their real name
            print(f"{'Unfortunately, that is wrong!'.upper()} Try again...\n") #
shout at the user! (UPPER CASE)

    input("Press Enter to quit")

main()
```

The symbol `!=` means 'not equal'

The line

```
while userInput != "your name":
```

translates to:

while `userInput` is **not** equal to '**your name**' do

As it was set to an empty string when the loop started, this condition is true, so the loop runs at least once.

If the user types in 'your name', the message `"That is correct!"` is printed out, but the loop exits because it's condition is no longer true. UserInput IS equal to 'your name'.

A variation on the while loop is a repeat until loop. Unfortunately Python does not support this type of loop.

Another loop called a 'for' loop will be covered shortly.

10 – MagicBall.py multiple if/elif/else

https://github.com/Inksaver/PythonForSchools/blob/main/Section2/10-MagicBall.py

```
# demonstrates while loop and random
import random

def main():
	question = ""
	while question != "q":
		# get an input from the user but do absolutely nothing with it except check
for quit!
		question = input("Ask the magic 8 ball a question: (q to quit)_")
		answer = random.randint(0,8)  #returns a random whole number between 0 and 7
		if answer == 0:
			print("It is certain")
		elif answer == 1:
			print ("Outlook good")
		elif answer == 2:
			print ("You may rely on it")
		elif answer == 3:
			print ("Ask again later")
		elif answer == 4:
			print ("Concentrate and ask again")
		elif answer == 5:
			print ("Reply hazy, try again")
		elif answer == 6:
			print ("My reply is no")
		elif answer == 7:
			print ("My sources say no")

	input("Press Enter to quit")

main()
```

Section 3

01-ForLoops.py Drawing triangles with ASCII

https://github.com/Inksaver/PythonForSchools/blob/main/Section3/01-ForLoops.py

```
# demonstration of for loops and repeat string

def main():
    numberOfRows = input("Type a number between 5 and 20_")

    numberOfRows = int(numberOfRows)
    if numberOfRows > 1:

        # draw a triangle
        # for variable = start, finish, step do
        for i in range(numberOfRows): # eg start at 1 step to 6: 1, 2, 3, 4, 5, 6
            # i starts at 0, then steps 1, 2, 3, 4, 5, etc -> numberOfRows
            lineOfChars = "*" * i # multiply string by i (Can be used for >1 character)
            print(lineOfChars)
            # reverse the triangle by starting with a high number and using -1 for the
step: 6, 5, 4, 3, 2, 1
            for i in range(numberOfRows, 0, -1):
                # alternative repeat characters
                lineOfChars = "".ljust(i, "*") # -> left justify an empty string to length
i using "*"
                print(lineOfChars)

    input("Press Enter to quit")

main() # program starts here
```

For loops differ from while loops because the number of times they iterate is limited and fixed.

A typical for loop such as:

```
for i in range(1, 5, 1):
```

runs 5 times only.

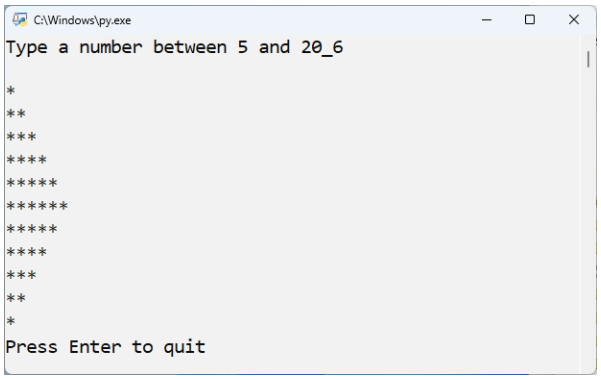
The variable **i** is the loop counter, and its value changes by 1 every time the loop runs.

When the value of `i` reaches 5 the loop runs one last time then stops. The value of `i` can be used by the code within the loop itself, as in this code where it is used to create a string of "*" characters of the length defined by the value of `i`.

```
lineOfChars = string.rep("*", i)
```

Note the second for loop uses -1

```
for i in range(numberOfRows, 0, -1):
```



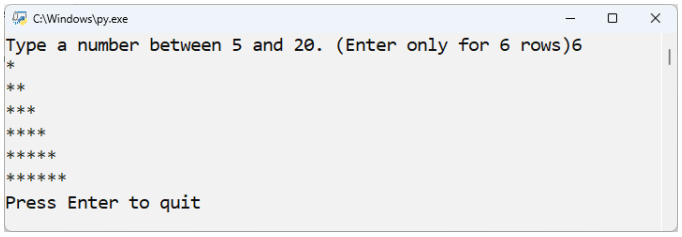
this runs the loop in reverse. The counter starts at the highest value, and drops by -1 each iteration until it reaches 0.
The “step” is -1 but can be set to any integer value.
General ‘for loop’ construction:

```
for counter in range(startValue, endValue, step):
```

02-ForLoopsAssignment.py

https://github.com/Inksaver/PythonForSchools/blob/main/Section3/02-ForLoops%20Assignment.py

Using the template below which draws the left side of a triangle, produce the following output:




```
# Change the code in main() to draw the following:
'''

     *
    ***
   *****
  *******
 *********
***********

'''

'''
        This example is 6 rows deep. You could do this in a number of ways:
        1. Lazy: write 6 string variables with the correct layout, then print them: 1 mark!

        2. Better: create 6 string variables using code to fill them, then print them: 2 marks!
           (you could use a combination of string.rep() and concatenation -> .. )
           eg row1 = string.rep(" ", 4) .. "*" .. string.rep(" ", 4)

        3. Use a for loop to draw both halves of the tree one row at a time similar to 2.
           (join the string.rep(# ,#) combinations in the for loop)
           but able to cope with as many rows as the user chooses: 10 marks!
'''

def main():
        numberOfRows = input("Type a number between 5 and 20. (Enter only for 6 rows)")
        if numberOfRows == "":
                numberOfRows = 6
        else:
                numberOfRows = int(numberOfRows)
        # draw a triangle
        # for variable = start, finish, step do
        for i in range(1, numberOfRows + 1):
                # string.rep() repeats the character(s) given by the number supplied
                # eg string.rep("*", 4) returns "****"
                lineOfChars = "*" * i
                print(lineOfChars)

main() # program starts here

input("Press Enter to quit")
```

Hints:

This example is 6 rows deep. You could do this in a number of ways:

1. Lazy: write chosen number string variables with the correct layout, then print them: 1 mark!

2. Better: create chosen number string variables using code to fill them, then print them: 2 marks!
   (you could use a combination of string.rep() and concatenation -> ..)
   eg row1 = string.rep(" ", 4) .. "*" .. string.rep(" ", 4)

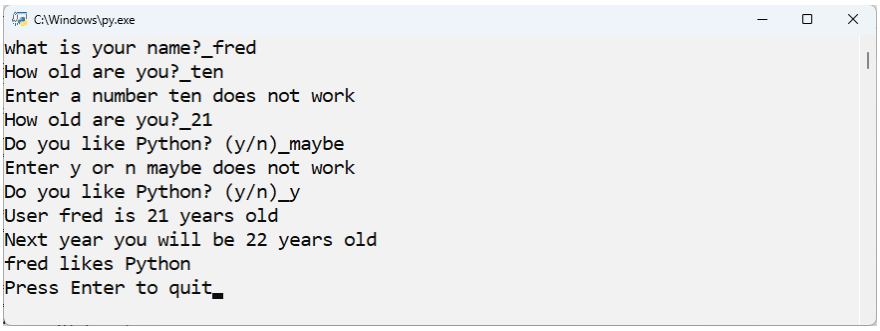
3. Use a for loop to draw both halves of the tree one row at a time similar to 2.
   (join the string.rep(# ,#) combinations in the for loop)
   but able to cope with as many rows as the user chooses: 10 marks!

Validating User Input

03-InputValidation.lua

https://github.com/Inksaver/PythonForSchools/blob/main/Section3/03-InputValidation.py

```
def input_x(prompt, dataType = "string"):     # get input from user
      # if not supplied then give default value
      while True:                             # break not required as return used instead
            userInput = input(prompt+ "_")
            if dataType == "string":
                  return userInput
            elif dataType == "int":
                  try:
                        return int(userInput)
                  except:
                        print(f"Enter a number {userInput} does not work")
            elif dataType == "float":
                  try:
                        return float(userInput)
                  except:
                        print(f"Enter a number {userInput} does not work")
            elif dataType == "bool":
                  if userInput.lower()[0] == "y":
                        return True
                  elif userInput.lower()[0] == "n":
                        return False
                  else:
                        print(f"Enter y or n {userInput} does not work")

def main():
      name = input_x("what is your name?")
      age = input_x("How old are you?", "int")
      likesPython = input_x("Do you like Python? (y/n)", "bool")

      print(f"User {name} is {age} years old")
      print(f"Next year you will be {age + 1} years old")
      if likesPython:
            print(f"{name} likes Python")
      else:
            print(f"{name} does not like Python")

main() # program starts here
input("Press Enter to quit")
```



A new function called input_x() has been written to expand on the basic input() function.

Apart from the prompt, there is a parameter descrbing the dataType of the return value requested

This allows the function to check whether the input is numerical (if a number is required), or a yes/no answer if a boolean (true / false) is needed.

To use it, supply either “string”, “int”, “float” or “bool” after the prompt text.
If nothing is supplied the line `dataType = "string"` will use the default value “string”

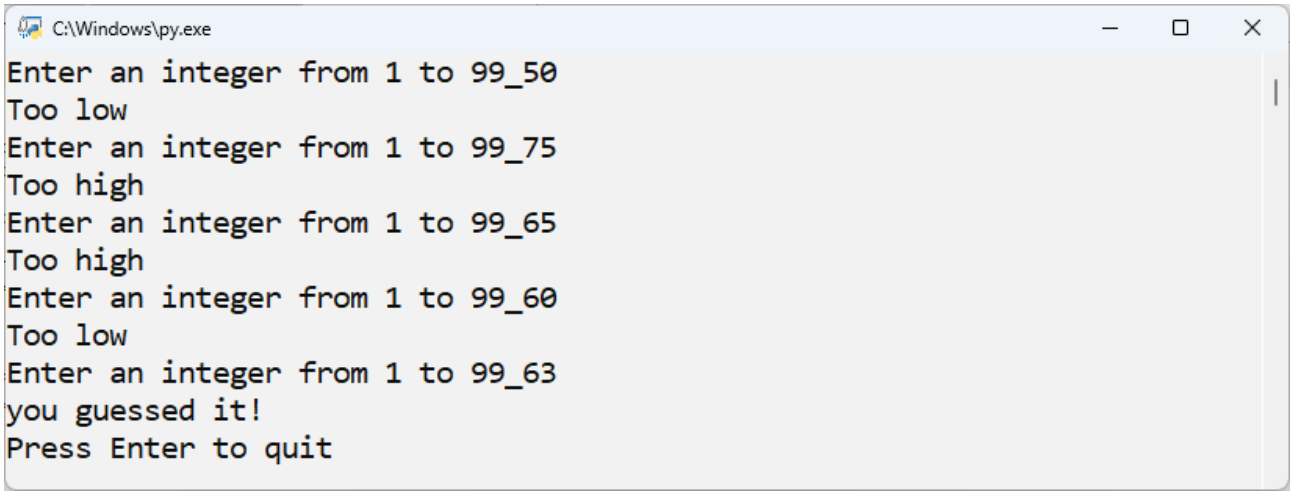
This is an improvement, but cannot distinguish between integer or real numbers, and will return an empty string if the user simply presses the enter key.

The input function now has an infinite while loop.
User input is checked to see if it is valid. If so the value is returned
If not the loop continues.

The dataType of the returned value is either string, integer, float or boolean, so can be safely be used directly in the calling code.

04-GuessTheNumber.py

https://github.com/Inksaver/PythonForSchools/blob/main/Section3/04-GuessTheNumber.py



Random Numbers

Games often use random numbers, and Python has a method of generating them, using it's random library

If you want a random number between 1 and 99 use this:

```
randomNumber = random.randint(1, 99)
print(randomNumber)
```

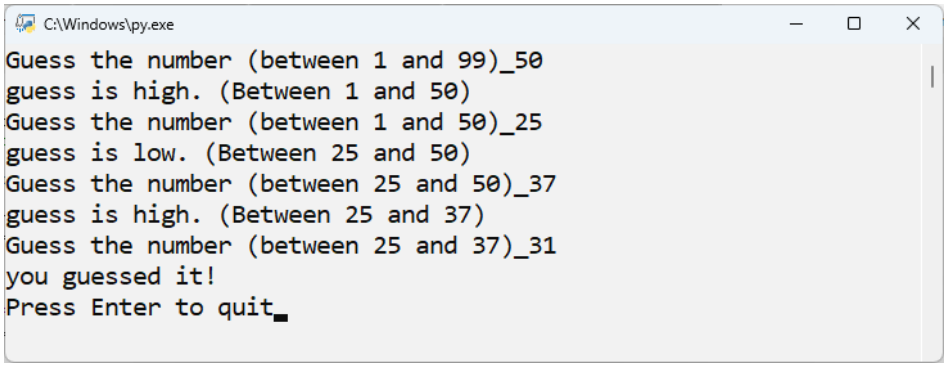
This simple game uses the random number function to create a number between 1 and 99
The user is asked to guess the number using the input_x() function developed earlier inside a while loop.

```
import random

def input_x(prompt, dataType = "string"):     # get input from user
      # if not supplied then give default value
      while True:                             # break not required as return used instead
            userInput = input(prompt+ "_")
            if dataType == "string":
                  return userInput
            elif dataType == "int":
                  try:
                        return int(userInput)
                  except:
                        print(f"Enter a number {userInput} does not work")
            elif dataType == "float":
                  try:
                        return float(userInput)
                  except:
                        print(f"Enter a number {userInput} does not work")
            elif dataType == "bool":
                  if userInput.lower()[0] == "y":
                        return True
                  elif userInput.lower()[0] == "n":
                        return False
                  else:
                        print(f"Enter y or n {userInput} does not work")

def main():
      n = random.randint(1, 99) # pick a number between 1 and 99

      guess = -1
      while guess != n:
            # no need to convert guess to a number, as return type is guaranteed
            guess = input_x("Enter an integer from 1 to 99", "int")
            if guess < n:
                  print("Too low")
            elif guess > n:
                  print("Too high")
            else:
                  print("you guessed it!")

      input("Press Enter to quit")

Main()
```

05-Assignment-Improve GuessTheNumber.py

https://github.com/Inksaver/PythonForSchools/blob/main/Section3/05-GuessTheNumber%20Assignment.py

The version above only tells you if you are too low or too high. It would be great if you were given some help remembering what you have already tried:



Modify the code above to give the helpful guidance shown in the screenshot → the range of numbers remaining.

Hints:

Create 2 variables to hold the largest and smallest numbers guessed so far.

Re-assign these variables as guesses are made

Output an appropriate message after each guess, telling the user how they have fared, and the range they now need to use.

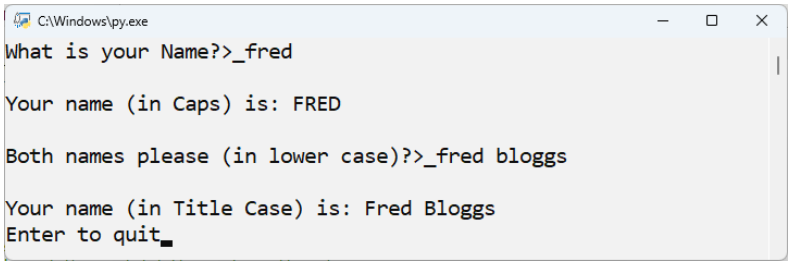
Section 4 Keyboard Input Library

The next few files create a re-useable library to get validated keybpord content. The first file uses a function called get_input() which adds a character(s) after the prompt, and returns the user input. There is no validation here.

01-GetUserName.py

https://github.com/Inksaver/PythonForSchools/blob/main/Section4/01-GetUserName.py

```
def get_input(prompt):
      response = input(f"{prompt}>_")

      return response

def quit():
      input("Enter to quit")


# It is good coding practice to use a main() function
def main():
      # assume user is too lazy to use a capital letter
      # use .title() to sort the capital letters out on all words
      print(f"\nYour name (in Caps) is: {get_input('What is your Name?').upper()}\n")
      print(f"\nYour name (in Title Case) is: {get_input('Both names please (in lower
case)?').title()}")
      quit()


# Script runs from here: everything above is ignored until called
main() # call the function main()
```



02-GetUserNameValidated.py

https://github.com/Inksaver/PythonForSchools/blob/main/Section4/02-GetUserNameValidated.py

An improvement on this is to expand the get_input() function to validate the user input, and loop until a valid input is given:

```
def get_input(prompt, size = 1):
      response = ""
      while len(response.strip()) < size:
            response = input(f"{prompt}>_")

      return response
```

This version takes a size parameter, so will only accept an input that is >= to size.

The .strip() function removes leading and trailing whitespace to stop users spamming the spacebar to break the input check.

So what if you wanted to get a number, or boolean response?

The function can be made larger and include more checks, but eventually it should be written into it's own file, which can then be used in any project by simply importing it.

Keyboard Library demonstration

https://github.com/Inksaver/PythonForSchools/blob/main/Section4/03-kboard_demo.py

https://github.com/Inksaver/PythonForSchools/blob/main/Section4/kboard.py

Download or copy/paste BOTH files kboard.py and 03-kboard_demo.py and make sure they are both in the same directory.

The file kboard.py is a simple library to obtain reliable input data.

It does not run on its own.

The file 03-kboard_demo.py is the test file to make use of the kboard library:

```
C:\Windows\py.exe
Type your name_

Just pressing the Enter key or spacebar doesn't work...
Type your name_fr

Try entering text between 3 and 6 characters...
Type your name_fred
How old are you_ten

Try entering a number - ten does not cut it...
How old are you_5.6

Try entering a number - 5.6 does not cut it...
How old are you_21
How tall are you in metres?_1.65
Do you like Python? (y/n)_maybe

Only anything starting with y or n is accepted...
Do you like Python? (y/n)_y
User name : Fred
User age : 21
User height : 1.65
User likes Python : True

What do think of this utility?
        1) Brilliant
        2) Not bad
        3) Could do better
        4) Rubbish
Type the number of your choice (1 to 4)_-1

Try a number from 1 to 4...
Type the number of your choice (1 to 4)_1
User thinks this utility is: Brilliant
Press Enter to Quit_
```

If you follow this screenshot, you can see the responses are validated with an error message shown if incorrect

kboard.py can be used in any of your own projects by adding the file to the project folder and using

import kboard as kb at the top of your starting file.

This is how it is done in 03-kboard_demo.py

```
import kboard as kb # shortens the class name to kb

def main():
    user_name = kb.get_string("Type your name", True, 3, 6) # returns user input in Title Case
    user_age = kb.get_integer("How old are you", 5, 120)    # gets an integer between 5 and 120
    user_height = kb.get_float("How tall are you in metres?", 0.5, 2.5) # gets a float between
                                                                          0.5 and 2.5
    user_likes_python = kb.get_boolean("Do you like Python? (y/n)")  # returns True or False

    print(f"User name : {user_name}")
    print(f"User age : {user_age}")
    print(f"User height : {user_height}")
    print(f"User likes Python : {user_likes_python}")
    print()
    menu_list = ["Brilliant", "Not bad", "Could do better", "Rubbish"]
    user_choice = kb.menu("What do think of this utility?", menu_list)
    print(f"User thinks this utility is: {menu_list[user_choice]}")

    # if running from a console/terminal instead of an IDE to prevent closing.
    kb.get_string("Press Enter to Quit", False, 0, 20) # Used instead of input("Press Enter to
                                                          Quit")

main()
```

If you study the code above you will see something new:

```
menu_list = ["Brilliant", "Not bad", "Could do better", "Rubbish"]
```

This is a **Python list** enclosed with square brackets [] and in this case it is a list of strings. Lists can contain pretty much anything such as numbers, other lists or dataTypes.

The list is passed to the keyboard library to produce a numbered menu as shown in the screenshot.

The kboard library is known as a code module in Python and is comparable to the Lua table structure, or static classes in other languages.

You can use any of it's functions by calling them, and passing parameters to qualify what you want to be returned.

To guarantee getting a valid string with a length of between 3 and 6 characters, in Title Case use:

```
name = kb.get_string("What is your name?", True, 3, 6)
```

Demonstrated here is the ability to get a guaranteed return value of the correct dataType:

- string `kb.get_string(prompt, withTitle, minValue, maxValue)`
- integer `kb.get_integer(prompt, minValue, maxValue)`
- float `kb.get_float(prompt, minValue, maxValue)`
- boolean `kb.get_boolean(prompt)`

Also a Menu option which guarantees an integer index value from a list supplied to the library

```
options = ["Brilliant", "Not bad", "Could do better", "Rubbish"]
choice = kb.menu(title, options)
```

The variable choice will be a number which is the index of the item in the supplied list
If choice == 0 then the value of options[0] is "Brilliant"

04-GetUserNameWithLibrary.py

https://github.com/Inksaver/PythonForSchools/blob/main/Section4/04-GetUserNameWithLibrary.py

```
import kboard as kb

def quit():
    input("Enter to quit")

# It is good coding practice to use a main() function
def main():
    #assume user is too lazy to use a capital letter
    '''
    kb.getString(prompt, withTitle, minInt, maxInt)
        prompt = "What is your name?", withTitle = False, min length 2, max length 20
        prompt = "Both names please?", withTitle = True, min length 4, max length 30
    '''

    print(f"\nYour name (in Caps) is: {kb.get_string('What is your Name?', False, 2, 20).upper()}\n")
    print(f"\nYour name (in Title Case) is: {kb.get_string('Both names please (in lower case)?',
True, 4, 30)}")

    quit()

# Script runs from here: everything above is ignored until called
main() # call the function main()
```

The output from this is exactly the same as the 2 earlier files, but this time it is using the external library to get the validated input.

Section5 Adventure Game

Now you can use the kboard library to get user input and menus, it is time to create a text adventure game using the kboard library to guarantee user input, and make use of the menu system.

For a typical text based adventure game you will need the following:

A player, with variables to store Name, Character, Strength, HitPoints, Health, Inventory etc

Locations: Places in your game where the player can visit

Items: Objects the player can find or use including weapons

Enemies: characters that want to be mean to the player.

You could use a list of variables eg player_name, player_character, player_health etc. but it gets a bit cumbersome.

Python has a new data structure introduced in version 3.7: the dataclass

```
@dataclass
class Player:
      name = ""
      character = ""
      strength = 0
      health = 0
      mana = 0
      hitpoints = 100
      inventory = []
      characters = ["Wizard", "Priest", "Fighter", "Ninja"]
```

This is similar to the table structure used in Lua as you can read and change the values using dot notation

To create a player use:

```
player = Player()
```

to change a value, eg name use:

```
player.name = “Fred”
```

to read a value use:

```
print(player.name)
```

This is a great introduction to the concept of Object Oriented Programming (OOP) by using a data structure that contains many different dataTypes.

01-PlayerDataClass.py

https://github.com/Inksaver/PythonForSchools/blob/main/Section5/01-PlayerDataClass.py

```
import kboard as kb
from dataclasses import dataclass

'''
https://www.w3schools.com/charsets/ref_utf_box.asp
utf8 box characters stored here for easy copy/paste:

┌ ┬ ┐ ─ ╒ ╤ ╕ ═
├ ┼ ┤ │ ╞ ╪ ╡ ║
└ ┴ ┘   ╘ ╧ ╛
'''

@dataclass
class Player:
        name = ""
        character = ""
        strength = 0
        health = 0
        mana = 0
        hitpoints = 100
        inventory = []
        characters = ["Wizard", "Priest", "Fighter", "Ninja"]

def display_player(player):
        separator = "=" * 40                    # repeat '=' 40 x
        print(separator)
        print("\t\tPlayer Properties")
        print(f"    Name:     {player.name}")
        print(f"    Character:{player.character}")
        print(f"    Strength: {player.strength}")
        print(f"    Health:   {player.health}")
        print(f"    Mana:     {player.mana}")
        print(separator)
        print()

def update_player(player):
        player.name = kb.get_string("Type a name for your player (3-15 characters)", True, 3, 15)
        choice = kb.menu("Choose a character", player.characters)
        player.character = player.characters[choice]
        if choice == 0: # Wizard
                player.strength = 80
                player.health = 90
                player.mana = 70
        elif choice == 1: # Priest
                player.strength = 70
                player.health = 100
                player.mana = 60
        else:
                player.strength = 50
                player.health = 50
                player.mana = 0
        # complete this if block to change values for all characters

        return player

def main():
        player = Player()
        player = update_player(player)
        display_player(player)

        input("Enter to quit")

main()
```

The output from this file is:

```
C:\Windows\py.exe
Type a name for your player (3-15 characters)_fred
Choose a character
        1) Wizard
        2) Priest
        3) Fighter
        4) Ninja
Type the number of your choice (1 to 4)_2
═══════════════════════════════════════
                Player Properties
    Name:      Fred
    Character:Priest
    Strength: 70
    Health:   100
    Mana:     60
═══════════════════════════════════════

Enter to quit
```

Download or copy/paste the file, run it, then make some changes:

Some of the player properties can be deleted / modified / added to suit the game you want to write. For example the player could be in a Sci-Fi, space, holiday park, haunted house situation where completely different characteristics are required.

02-LocationsDataClass.py

https://github.com/Inksaver/PythonForSchools/blob/main/Section5/02-LocationDataClass.py

The dataclass does not allow 'methods' which are functions and procedures that are also stored in the class, as it is data only, but it can be instantiated, meaning you can create data objects.

Our adventure will need different locations, so another dataclass called Locations has been added and this time more than one location has been created:

```
C:\Windows\py.exe
Type a name for your player (3-15 characters)_fred
Choose a character
        1) Swot
        2) Nerd
        3) Dunce
        4) Bully
Type the number of your choice (1 to 4)_2
═══════════════════════════════════════════════════════════
                Player Properties
    Name:      Fred
    Character:Nerd
    Strength: 70
    Health:   100
    Mana:     60
═══════════════════════════════════════════════════════════

═══════════════════════════════════════════════════════════
You are in Classroom, The computer lab
furnished with steam driven PCs
═══════════════════════════════════════════════════════════

Which direction?
        1) Corridoor
        2) Quit
Type the number of your choice (1 to 2)_
```

```python
import kboard as kb
from dataclasses import dataclass

'''
https://www.w3schools.com/charsets/ref_utf_box.asp
utf8 box characters stored here for easy copy/paste:

┌ ┬ ┐ ─ ╒ ╤ ╕ ═
├ ┼ ┤ │ ╞ ╪ ╡ ╫
└ ┴ ┘   ╘ ╧ ╛
'''

@dataclass
class Player:
        name = ""
        character = ""
        strength = 0
        health = 0
        mana = 0
        hitpoints = 100
        inventory = []
        characters = ["Swot", "Nerd", "Dunce", "Bully"]

@dataclass
class Location:
        name = ""
        description = ""
        to_north = ""
        to_east = ""
        to_south = ""
        to_west = ""
        items = []

def create_locations():
        '''
        create locations. IMPORTANT ensure exits correspond:
        eg Classroom.toEast -> Corridoor therefore Corridoor.to_west = Classroom
        '''
        locations = []    #empty list

        locations.append(Location()) # add a new empty Location object to the list
        locations[0].name = "Classroom"    # set tne name property
        locations[0].description = "The computer lab\nfurnished with steam driven PCs" # set the description
property
        locations[0].to_east = "Corridoor" # set the exit(s) as appropriate

        locations.append(Location())
        locations[1].name = "Corridoor"
        locations[1].description = "narrow passage filled with pupils,\nscattered trainers and bags.\nIs that weed I
can smell?"
        locations[1].to_north = "Office"
        locations[1].to_south ="Outside"
        locations[1].to_west = "Classroom"

        locations.append(Location())
        locations[2].name = "Office"
        locations[2].description = "Head teachers office. \nA well used cane hanging from a hook. \nHalf empty
whisky bottle on the desk"
        locations[2].to_south = "Corridoor"

        locations.append(Location())
        locations[3].name = "Outside"
        locations[3].description = "the playground: scattered empty plastic bottles\nnotice boards covered in
grafitti"
        locations[3].to_north = "Corridoor"

        return locations

def display_location(locations, current_location):
        directions = []   # empty list
        location_index  = -1 # initialised to -1
        for i in range(len(locations)):
                if locations[i].name == current_location:
                        location_index = i
                        break
        if location_index > -1:    # must have found match as index  >= 0
                separator = "=" * 79                        # repeat '=' 79 x
                location = locations[location_index]
                print(separator)
```

```
                print(f"You are in {location.name}, {location.description}")
                if location.to_north != "":
                        directions.append(location.to_north)
                if location.to_east != "":
                        directions.append(location.to_east)
                if location.to_south != "":
                        directions.append(location.to_south)
                if location.to_west != "":
                        directions.append(location.to_west)
                print(separator)
                print()
        else: # coding error ? mis-spelt location.name
                print(f"Unable to find location '{current_location}'")

        return directions

def display_player(player):
        separator = "=" * 79      # repeat '=' 79 x
        print(separator)
        print("\t\tPlayer Properties")
        print(f"    Name:     {player.name}")
        print(f"    Character:{player.character}")
        print(f"    Strength: {player.strength}")
        print(f"    Health:   {player.health}")
        print(f"    Mana:     {player.mana}")
        print(separator)
        print()

def update_player(player):
        player.name = kb.get_string("Type a name for your player (3-15 characters)", True, 3, 15)
        choice = kb.menu("Choose a character", player.characters) # choice is an integer between 0 and the length of
list-1
        player.character = player.characters[choice] # get the name from the list of characters
        if choice == 0: # Swot
                player.strength = 80
                player.health = 90
                player.mana = 70
        elif choice == 1: # Nerd
                player.strength = 70
                player.health = 100
                player.mana = 60
        else: # Dunce, Bully or any others
                player.strength = 50
                player.health = 50
                player.mana = 0
        # complete this if block to change values for all characters

        return player

def play(player, locations, current_location):
        ''' game loop. Runs until user selects "Quit"  '''
        while current_location != "Quit":
                directions = display_location(locations, current_location)
                directions.append("Quit")
                choice = kb.menu("Which direction?", directions)
                current_location = directions[choice]

def main():
        player = Player()                   # create the player
        player = update_player(player)      # ask user to input name/character
        display_player(player)              # display player properties
        locations = create_locations()      # create locations
        current_location = "Classroom"      # set current location
        play(player, locations, current_location)

        input("Enter to quit")

main()
```

So how does this work?

Starting at the bottom, main() starts the program as usual.

```
player = Player() # create the player
```

This creates a player 'object' from the dataclass Player(). The default values are used.

```
player = update_player(player)    # ask user to input name/character
```

This calls the function update_player(), passing in the newly created player object as a parameter.

```
player.name = kb.get_string("Type a name for your player (3-15 characters)", True, 3, 15)
```

This asks the user for a name using the kboard library, so a string between 3-15 characters, in Title Case is guaranteed. This new value is assigned to player.name

```
choice = kb.menu("Choose a character", player.characters) # choice is an integer
between 0 and the length of list-1
player.character = player.characters[choice] # get the name from the list of
characters
```

This uses the kboard menu function to get a character from the pre-existing list which was provided in the class definition: `characters = ["Swot", "Nerd", "Dunce", "Bully"]`

It is also possible to replace this list with a new one if required.

Choice is the number chosen by the user in response to the menu. (The menu displays the choices numbered from 1, not 0, so the number returned is adjusted to compensate)

To get the value of the character, the list index is used, so choice 0 is "Swot"

A simple if else choice changes the values of the other properties. This block is incomplete.

Next the player's properties are displayed using the display_player() procedure

```
display_player(player)          # display player properties
```

The game locations are now created and stored in a list called 'locations'

```
locations = create_locations() # create locations
```

This time multiple Location objects are created, but they are not assigned to individual variables such as 'home', 'location1' etc. Instead they are simply added to the list, and have to be accessed by searching the list for the location name required.

```
locations.append(Location()) # add a new empty Location object to the list
locations[0].name = "Classroom"    # set tne name property
locations[0].description = "The computer lab\nfurnished with steam driven PCs"    # set the
description property
locations[0].to_east = "Corridoor" # set the exit(s) as appropriate
```

This is done in 2 stages:

1. append a new empty object to the list

2.Update the properties of that object

Next the current_location is chosen:

```
current_location = "Classroom" # set current location
```

finally the game loop play() function is called

```
def play(player, locations, current_location):
    ''' game loop. Runs until user selects "Quit"  '''
    while current_location != "Quit":
        directions = display_location(locations, current_location)
        directions.append("Quit")
        choice = kb.menu("Which direction?", directions)
        current_location = directions[choice]
```

This runs until the user chooses “Quit”

A list of possible exits is obtained from calling the function display_location(). This checks each possible exit and adds to the list if it is not an empty string.

This list is then passed to the kboard.menu function to get the user choice of which way to go.

The current_direction is updated and the loop continues.

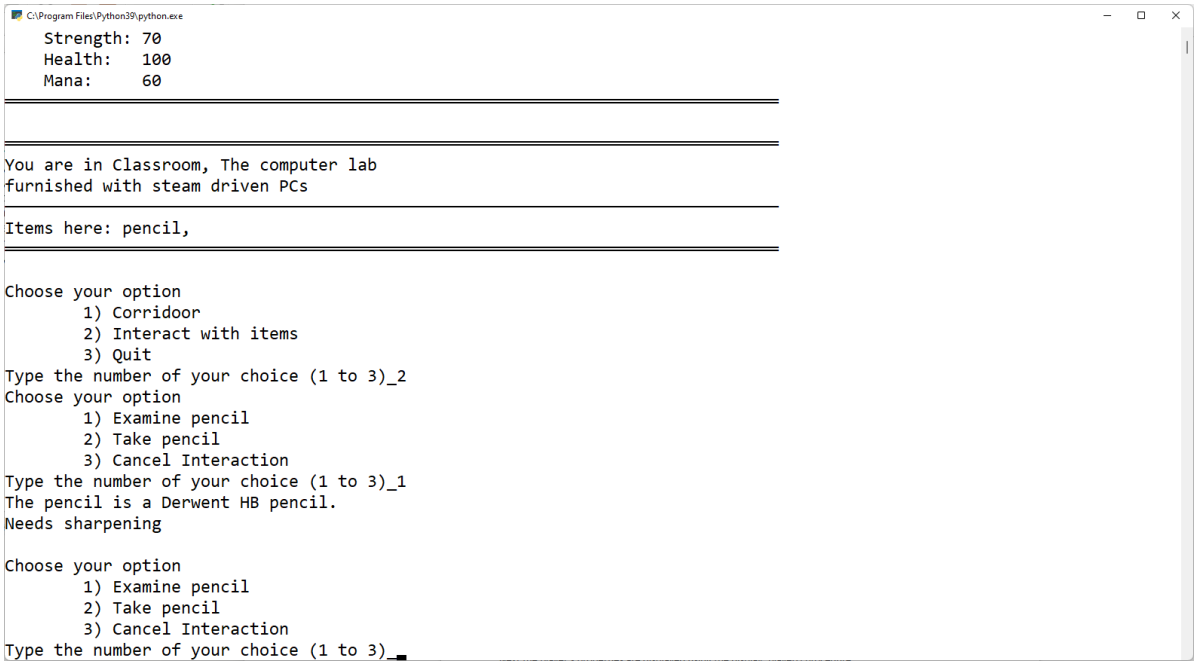
Assignment: Customise the Adventure Game

The file: 03-AdventueGame.py has a working adventure game with 4 locations, 2 items.

https://github.com/Inksaver/PythonForSchools/blob/main/Section5/03-AdventureGame.py

Use it to customise your version:

1. Change the player characteristics, add or remove properties, eg player.stamina
2. Add more items, keys, torches, weapons, clues
3. Add/Edit locations to suit the adventure style you are writing
4. Add an Enemy dataclass and add enemy(s) to the location properties



Keyboard.py library code

https://github.com/Inksaver/PythonForSchools/blob/main/Section4/kboard.py

The kboard.py code is shown here for interest only. There are parts that you may find difficult to understand at this stage, but you do not need to know in order to use it.

```
def process_input(prompt, min, max, data_type):
    ''' This function is not called directly from other files. Python does not have a 'private'
keyword instead the convention is to use a leading underscore _ '''
    valid_input = False
    while valid_input is False:
        print(prompt, end="_")
        user_input = input()    # Could use: user_input = input(f"{prompt}_"), but these 2 lines can
be used with other languages
        if len(user_input) == 0:
            print("\nJust pressing the Enter key doesn't work...")
        else:
            if data_type == "bool":
                if user_input[0].lower() == "y":
                    user_input = True
                    valid_input = True
                elif user_input[0].lower() == "n":
                    user_input = False
                    valid_input = True
                else:
                    print("\nOnly anything starting with y or n is accepted...")
            else:
                try:
                    if data_type == "int":
                        user_input = int(user_input)
                    elif data_type == "float":
                        user_input = float(user_input)

                    if user_input >= min and user_input <= max:
                        valid_input = True
                    else:
                        print(f"\nTry a number from {min} to {max}...")
                except:
                    print(f"\nTry entering a number - {user_input} does not cut it...")

    return user_input

def get_string(prompt, with_title = False, min = 1, max = 20): # with_title, min and max can be over-
ridden by calling code
    ''' Public method: Gets a string from the user, with options for Title Case, length of the string.
Set min to 0 to allow empty string return '''
    valid = False
    while not valid:
        user_input = input(prompt + "_").strip()      # change '_' for any preferred character eg '>'
        if len(user_input) == 0 and min > 0:
            print("\nJust pressing the Enter key or spacebar doesn't work...")
        else:
            if len(user_input) >= min and len(user_input) <= max:
                if with_title:
                    user_input = to_title(user_input) # Python has string.title() function. C#, Lua and
Java do not, so not used here
                valid = True
            else:
                print(f"\nTry entering text between {min} and {max} characters...")

    return user_input

def get_integer(prompt, min = 0, max = 65536): # min and max can be over-ridden by calling code
    ''' Public Method: gets an integer from the user '''
    return process_input(prompt, min, max, "int")

def get_float(prompt, min = 0.0, max = 1000000.0): # min and max can be over-ridden by calling code
```

```
    ''' Public Method: gets a float from the user '''
    return process_input(prompt, min, max, "float")

def get_boolean(prompt):
    ''' Public Method: gets a boolean (yes/no) type entries from the user '''
    return process_input(prompt, 1, 3, "bool")

def to_title(input_text):
    ''' Private Method: makes text into Title Case '''
    temp_list = list(input_text.lower())
    for index in range(len(temp_list)):
        if index == 0:
            temp_list[0] = temp_list[0].upper()
        elif temp_list[index - 1] == " ":
            temp_list[index] = temp_list[index].upper()

    return ''.join(temp_list)

def menu(title, menu_list):
    ''' displays a menu using the text in 'title', and a list of menu items (string) '''
    print(title)
    for i in range(1, len(menu_list) + 1):    # this range numbers the menu items starting at 1
        print(f"\t{i}) {menu_list[i - 1]}")   # -1 as the iterator starts at 1 instead of 0

    return get_integer(f"Type the number of your choice (1 to {len(menu_list)})", 1, len(menu_list)) -
1 # -1 to return correct list index
```

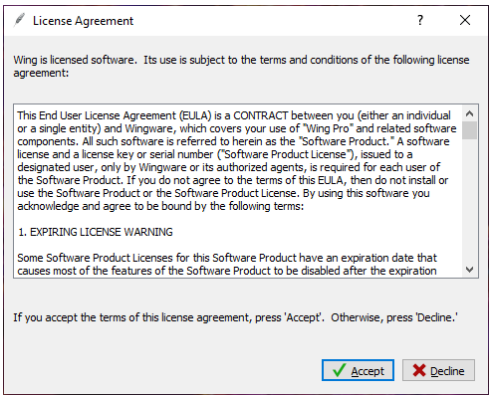
Wing IDE (Python) Tutorial

Wing is an IDE (Integrated Development Environment) and is packed full of features, but it does take some initial setting up.

Start Wing in the usual way.

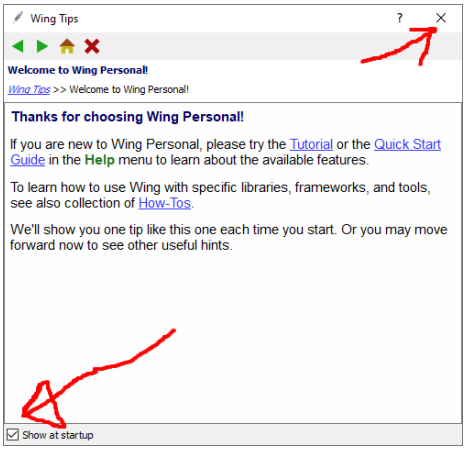
(Find it near the bottom of the Windows 10 designed-for-tablets interface

Or Search for 'Wing')



Click Accept on the License Agreement

Wing will open with this dialog

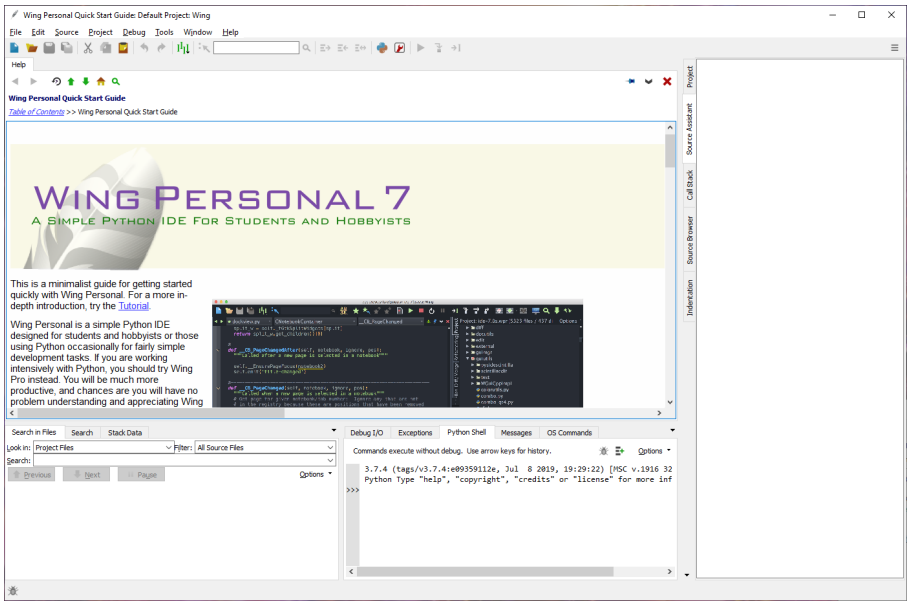


Un-tick 'Show at startup.

Close the dialog.

Initial startup window

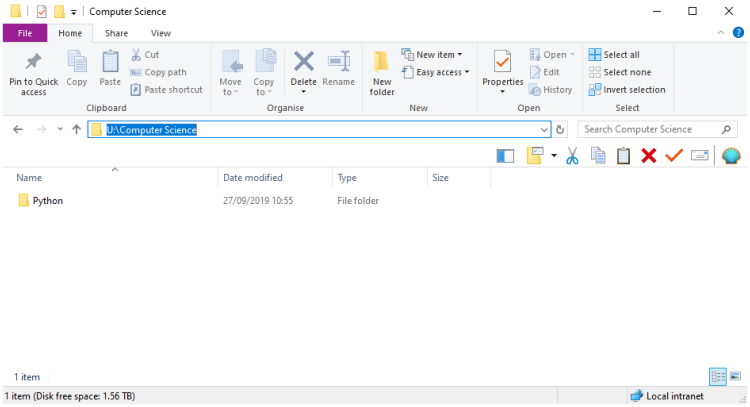
You can close the help page shown here by clicking the red cross.
(2/3 across the window, under the toolbar)



You will be using Wing to open your entire Python folder, not just individual files. If you have been subjected to the horrible 'Idle' experience, this will be a foreign concept to you.

If you have not already done so, create a Python folder somewhere in your user space. (You can minimise Wing to give you more desktop space to open Windows explorer and create your folder.)

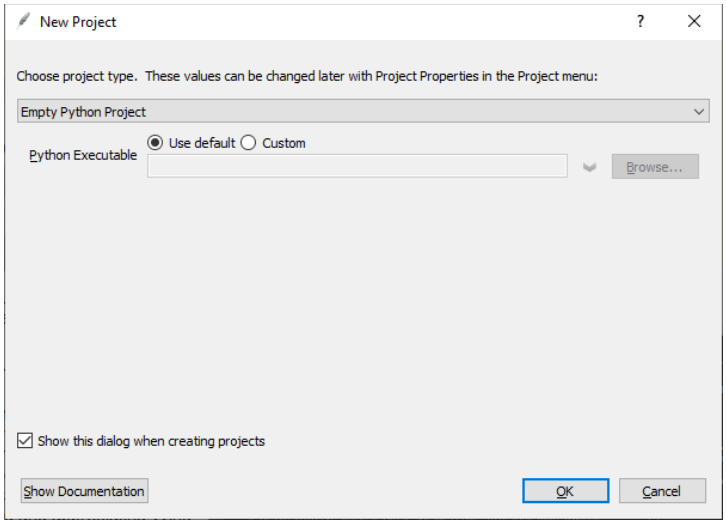
This screenshot is **not** from a school PC, so has been created to represent your 'U:' drive, a sub-folder called 'Computer Science' and a folder called 'Python' using Windows FileSystem Explorer



From the Wing window choose:

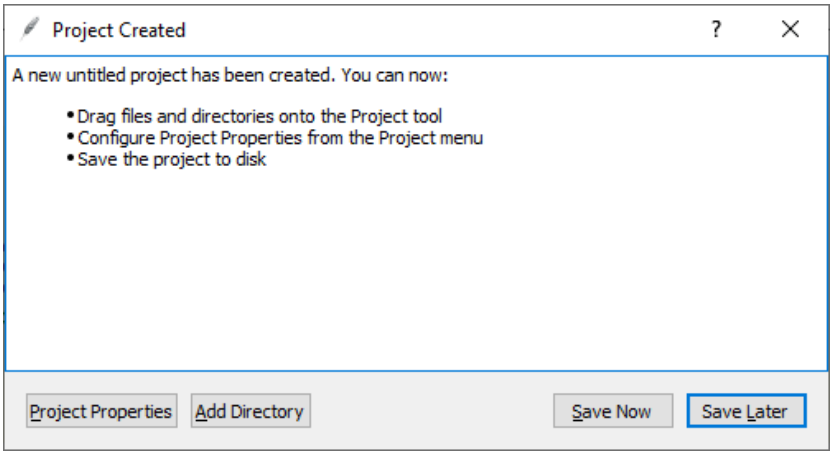
`Menu:Project → New Project`

Nothing needs changing here.



Click OK

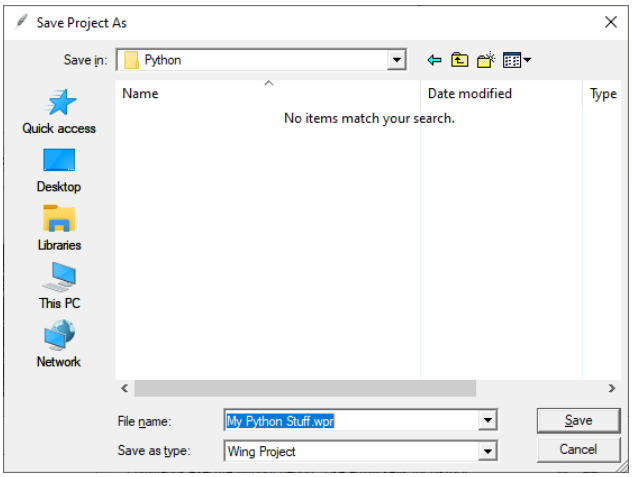
This dialog appears



Click the 'Save Now' button

If you get warnings about 'System', or 'Permissions' etc, etc with red warning boxes, just click OK.

The file browser dialog will open:

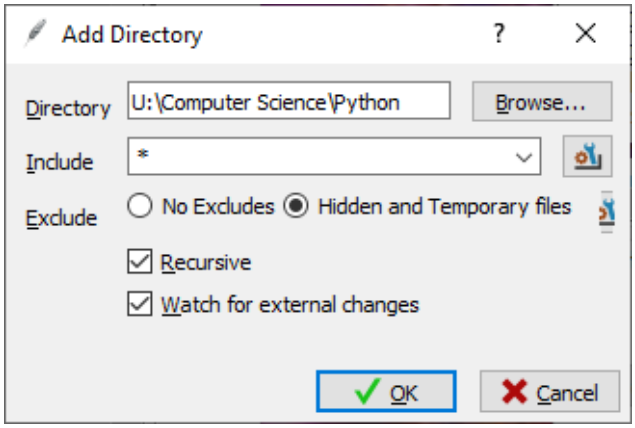


Find your 'Python' folder.

Give the project a name:
(Probably NOT 'My Python Stuff')

Click 'Save'

Menu: Project → Add Existing Directory

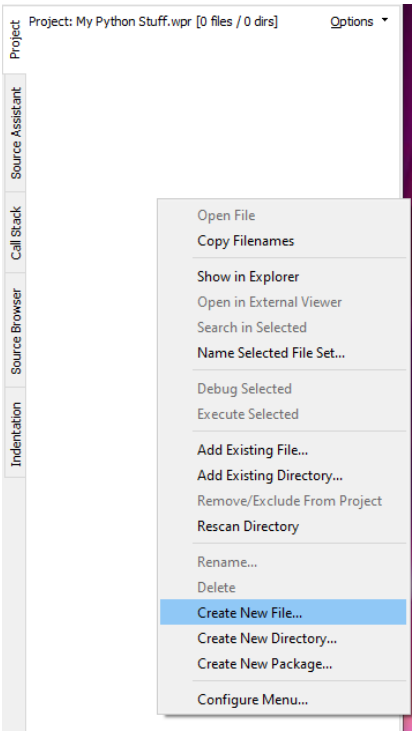


It should already be set to your Python folder
This will add any existing files or sub-folders
to your project

Click 'OK'

If you already have python files in that folder
they will show up on the right side in the Project tab.

(This one is empty.)

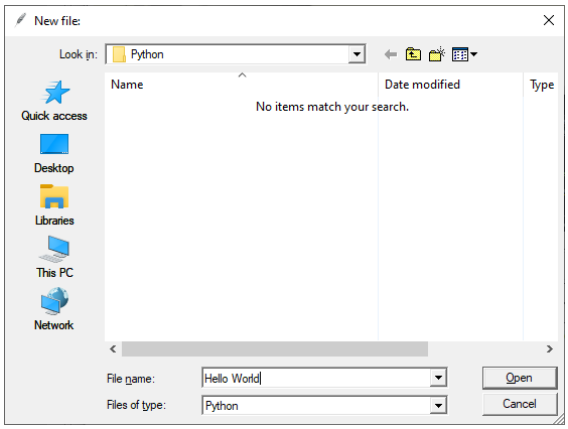


To create a new file right-click in the project tab
and select 'Create New File'

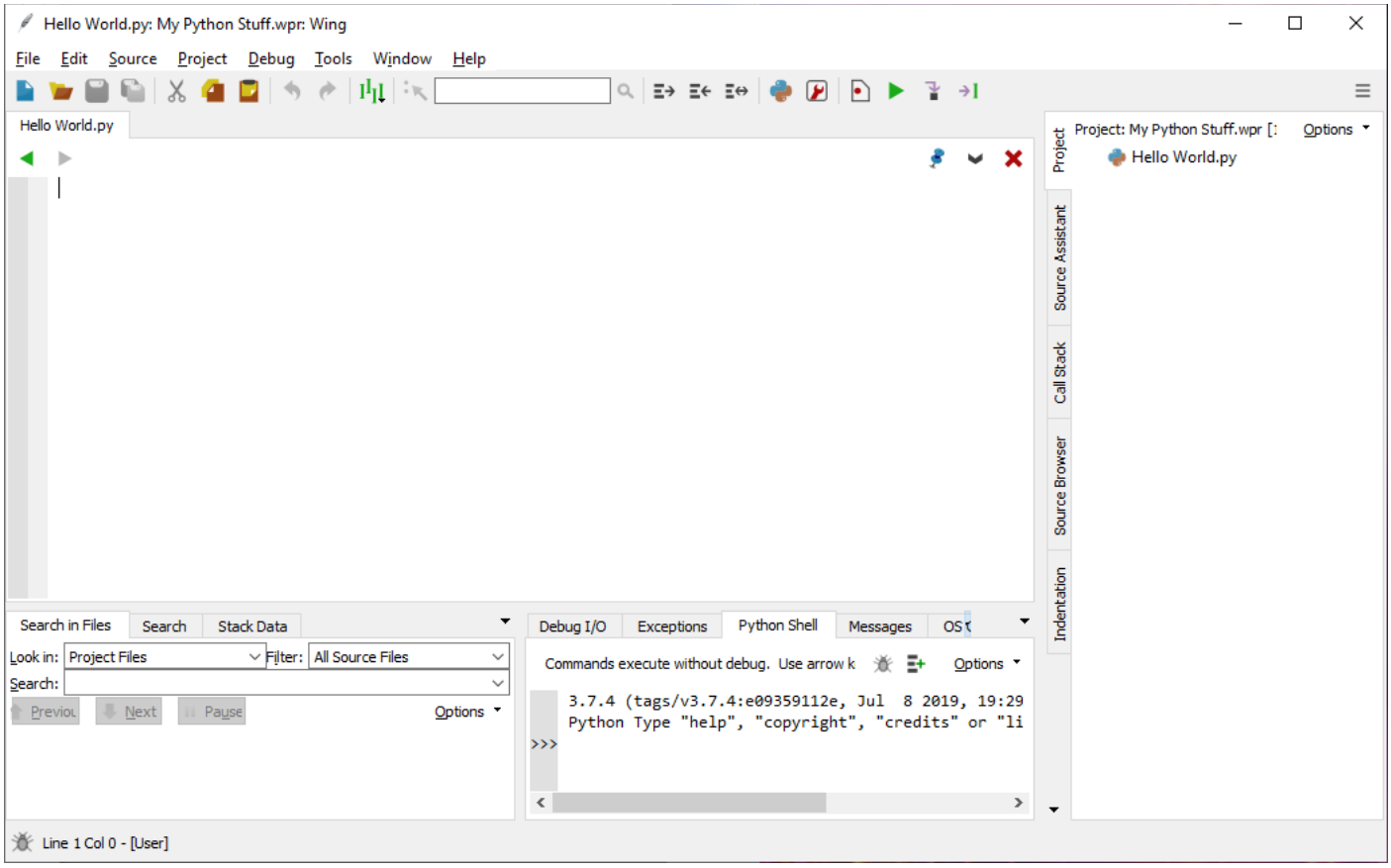
The File browser will appear

Type the name of your file:

(‘Hello World’ in this screenshot)



Click ‘Open’



Note the file showing in the Project tab on the right (Hello World.py)

Note the same file is open for editing with the cursor waiting on the left side.

You can add existing or new files / folders by right- clicking in the Project Tab.

Before starting to type your code, Change your Preferences:

1. Line numbers: **`Menu → Edit → Show Line Numbers`**

2. Indentation: This is the curse of Python in general, and Idle in particular:

`Menu → Edit → Preferences…`

Select Editor

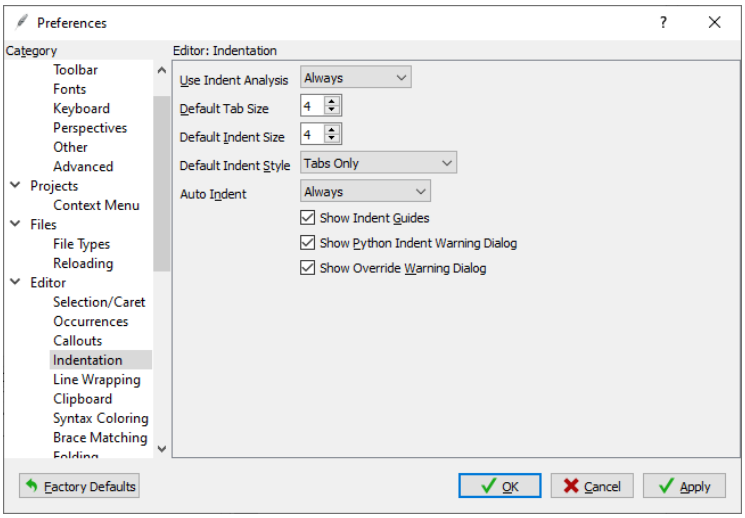
Select Indentation

Default tab size 4

Default Indent size 4

Default indent Style Tabs Only

Click: Show Indent Guides



This will give you the easiest options for indents, and give the least amount of grief.

When indenting your code always use the Tab key, not spaces

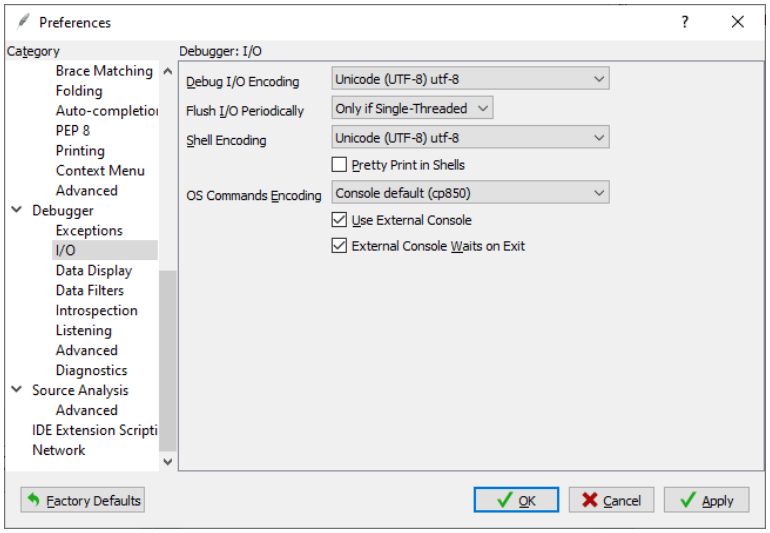
While preferences is open:

Select Debugger

Select IO

Click: Use External Console

Click: External Console Waits on Exit.



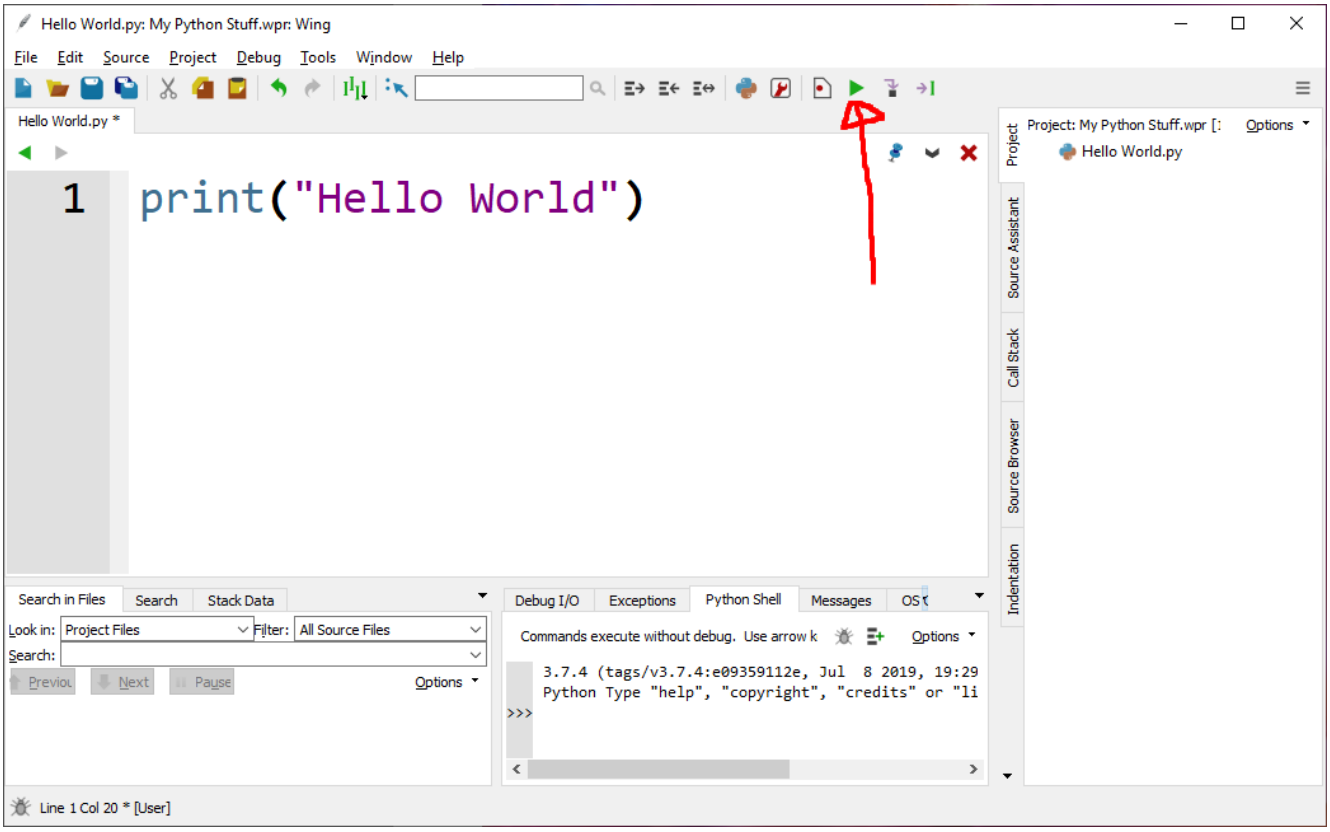
Click OK

You can now edit and run your Python scripts.

Use L-Ctrl key along with + or - keys to change the font size in the editor if required.

Write your epic script and **click the green triangle** to run it:

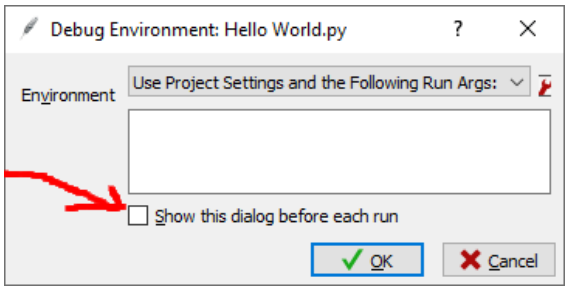
(The example shown here is obligatory when learning a new programming language)



This dialog is a nuisance.

Un-tick 'Show this dialog before each run'

It will not appear any more (but only for this file)

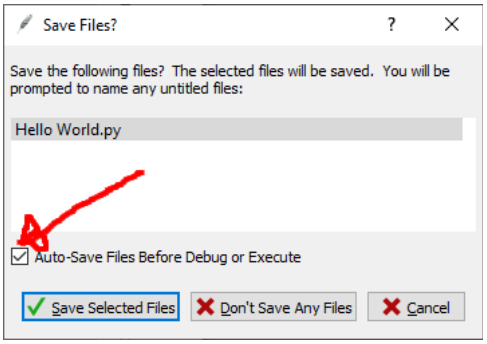


Click OK

This appears every time you run after an edit.
It is also a nuisance.

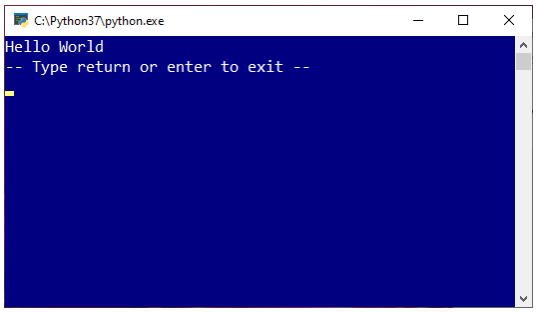
Tick the box Autosave Files

It will not appear any more (but only for this file)



Click 'Save selected Files"

Success



The problem is these settings are saved to:
`C:\Users\<username>\AppData\Roaming\Wing Personal 7`

which is a local folder on the computer you are currently using. If you fight with your classmates next visit and get to use the same one again, all your settings are preserved.

So why go to all this trouble? What is there to be gained from using Wing?

1. Multiple files can be edited, allowing copy/paste between them, and when you start using multi-file programs, you can switch between the files quickly and easily.

2. Instant indentation error fixing.

3. Auto-complete

4. Excellent debugging

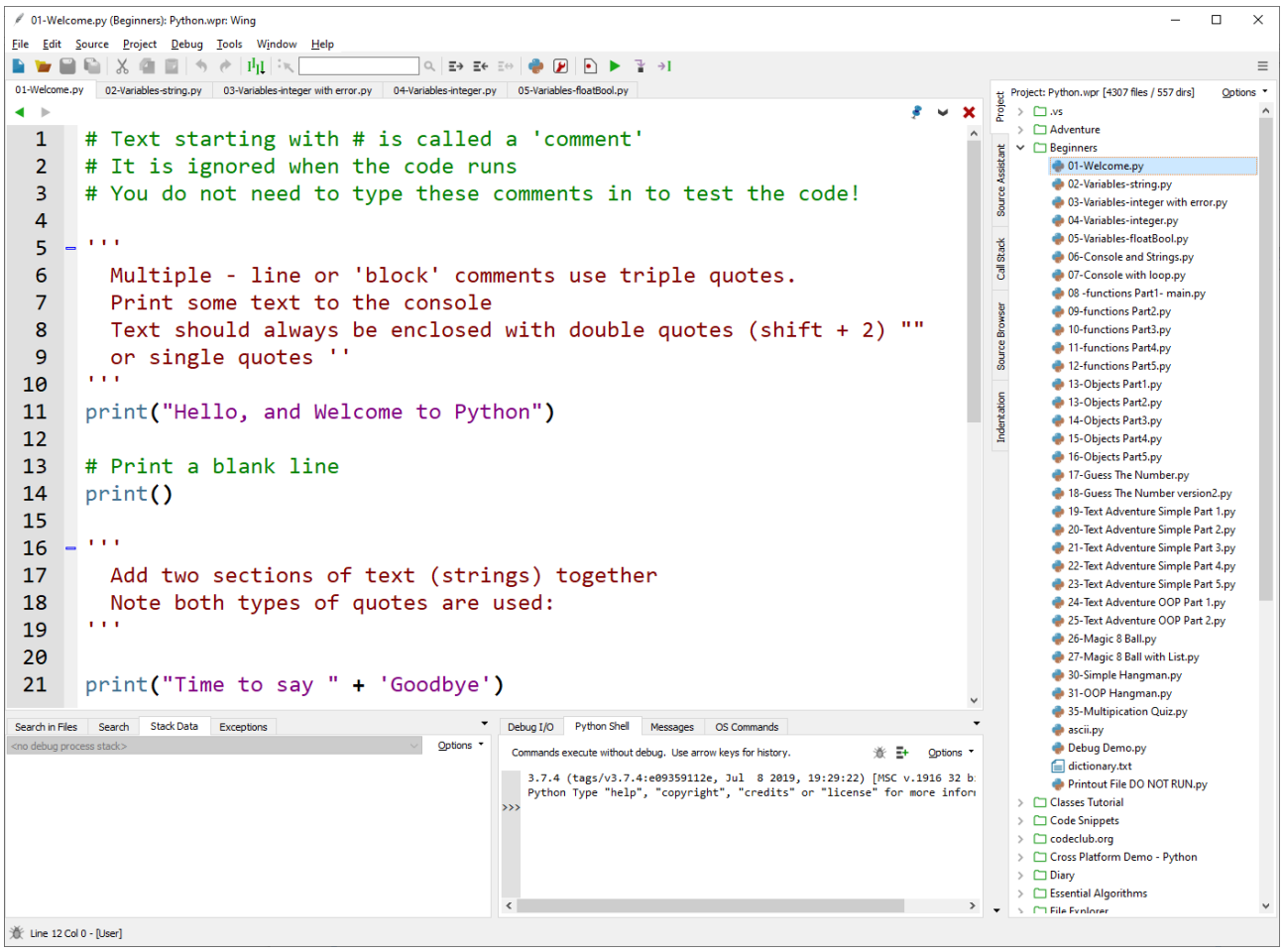
1. Multiple file editing

This screenshot shows multiple folders and files in the Project Tab

5 files have been opened by double-clicking them in the Project Tab

The first file '01-Welcome.py' has been selected for editing.

You can copy lines from any file and paste in another



2. Instant indentation error fixing

If you look carefully at the screenshot below, you can make out faint yellow lines under the text on lines 6 and 7

This is because the indentation is a mixture of 4 space characters and one tab character.

```
1  def add(num1, num2):
2      return num1 + num2
3
4  def main():
5      numberAsString = input("Type a number_")
6      number1 = int(numberAsString)
7      print("First Number = " + str(number1))
8      numberAsString = input("Type a second number_")
9      number2 = int(numberAsString)
10     print("Second Number = " + str(number2))
11     print("Total = " + str(add(number1, number2)))
12
13 main()
```

When you hover the mouse over the yellow line you get a tooltip:

```
numberAsString = input("Type a number_")
number1 = int(numberAsString)
print("Firs Warning: Inconsistent indentation (not in units of indent size) umber1))
numberAsString = input("Type a second number_")
```

Finally when you try to run the code you get the usual error Idle throws at you all the time:

```
1  def add(num1, num2):
2      return num1 + num2
3
4  def main():
5      numberAsString = input("Type a number_")
6      number1 = int(numberAsString)
7      print("First Number = " + str(number1))
8      numberAsString = input("Type a second number_")
9      number2 = int(numberAsString)
10     print("Second Number = " + str(number2))
11     print("Total = " + str(add(number1, number2)))
12

File "c:\Users\graha\Dropbox\Computer Club\Python\Beginners\Debug Demo.py", line 6, in ?
number1 = int(numberAsString)
builtins.IndentationError: unindent does not match any outer indentation level (Debug Demo.py, line 6)
```

Fixing the error:

1. Select the Indentation Tab

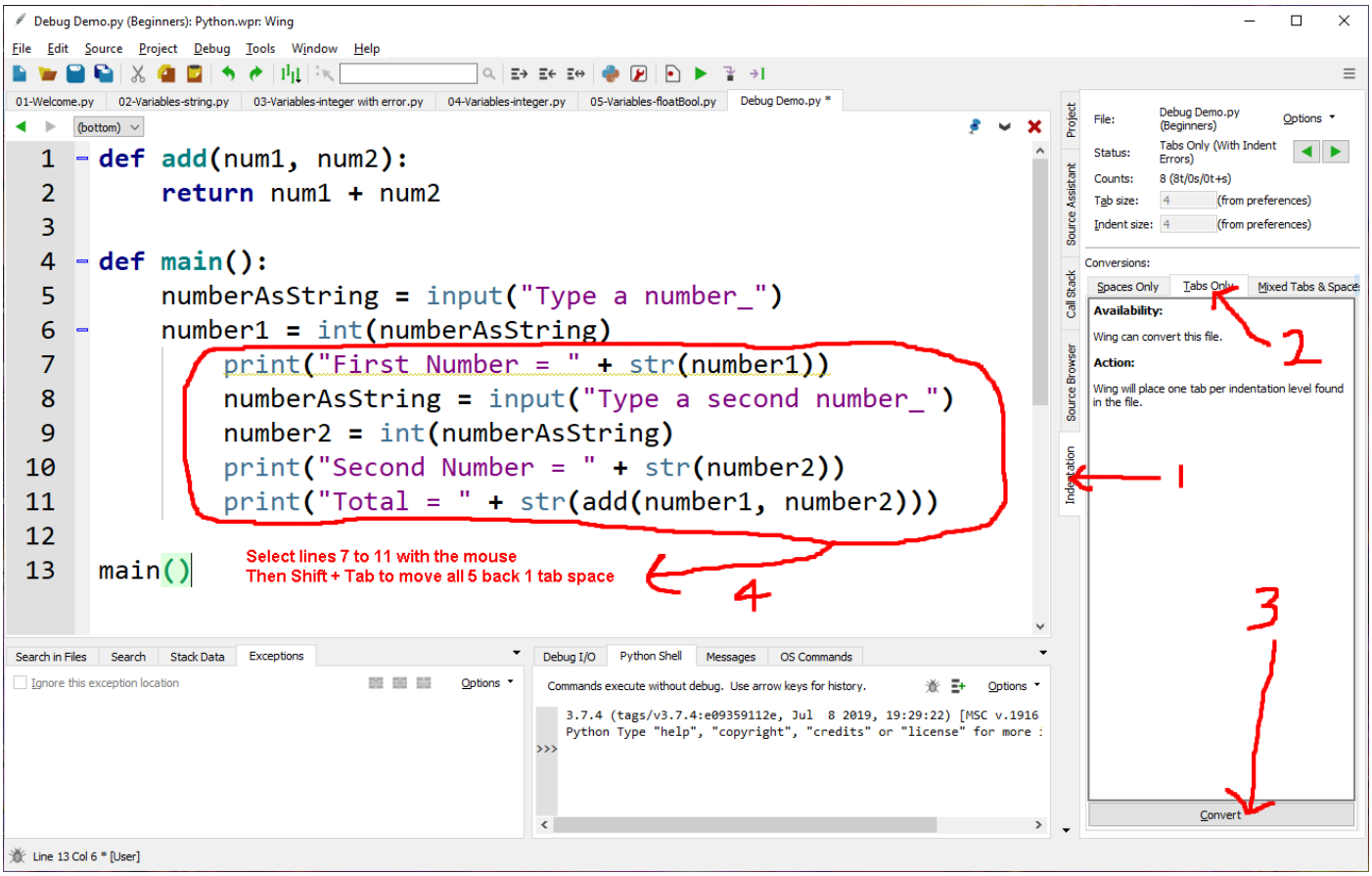
2. Select the Tabs Only Tab

3. Click 'Convert'

4. Depending how messed-up the original file was, some manual correcting may be needed, as here.
   Select all the lines out of place with the mouse.

   To move them to the right, hit the Tab key

   To move them to the left hit Shift + Tab

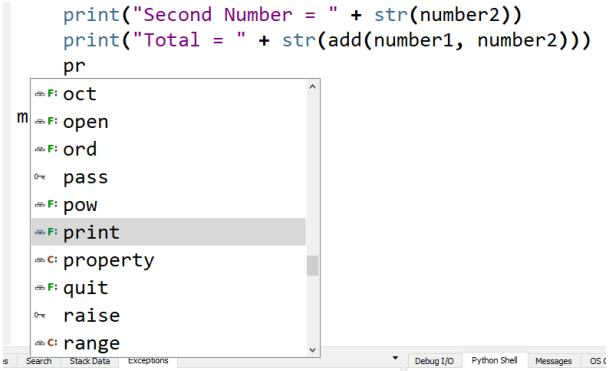


3. Auto-Complete (Intellisense)

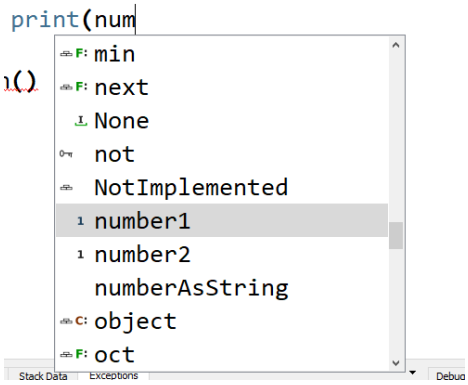
Using the above example, start a new line and type ‘p’ followed by ‘r’:

The intellisense has chosen ‘print’ as the most likely term required. Hit the Tab key to auto-complete

If it was not the correct word, use mouse or arrow keys to select the correct entry



The variable `numberAsString` has already been defined, so to complete the print statement, use an open left bracket, followed by ‘num’. The first 3 matches are returned:



Down arrow twice, Tab key and auto-complete is done.

Debugging

To stop the script running at a selected place, click in the grey margin next to a line number to insert a red dot (a break point).

Select the tab 'Stack Data' in the lower left pane

Select the 'Call Stack' tab on the right pane

If you start the program with F5 or the green triangle it will run until it reaches the break point.
For this example start by hitting F7 (or Menu → Debug) so it goes to the beginning (Line 1) and waits there:

```
1  def add(num1, num2):
2      return num1 + num2
3
```

Line 1 (as expected) Hit F7 again

```
1  def add(num1, num2):
2      return num1 + num2
3
4  def main():
5      numberAsString = input("Type a number_")
```

Line 4 NOT Line 2. The interpreter saw a function

definition, noted it's presence, and moved on. F7

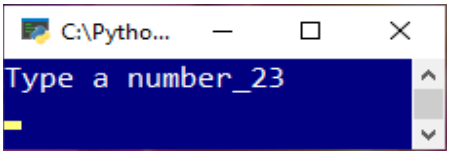
```
12
13 main()
14
```

Line 13, The end of the script. Not line 5. As before a definition has been noted and ignored. Hit f7

```
4  def main():
5      numberAsString = input("Type a number_")
```

Finally a line of code on line 13 'called' the function main() and was sent to the first line of code inside the function. Hit F7

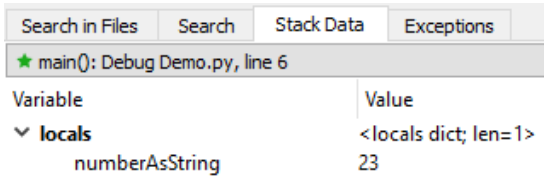
As this will run the code containing input(), you will need to interact with the console, type a number, eg 23 and hit return.




```
5      numberAsString = input("Type a number_")
6      number1 = int(numberAsString)
```

Line 6. The variable numberAsString has been given a value of 23. Look in the 'Stack Data' pane:

Hit F7 again to move to line 7. This will execute line 6



Line 6 converts the input string '23' into a number 23

Unfortunately the Stack Data does not distinguish between strings and numbers.

Hovering the mouse does: