

Software for Digital Innovation (CIS4044-N)

Tutorial 1: Introduction to Python and IDLE

Before You Start

Please take some time to familiarise yourself with the IDLE IDE.

The IDLE IDE documentation

<https://docs.python.org/3.6/library/idle.html>

Before you touch your keyboard, read through this brief to get an overview of the task at hand. You are not expected to complete the entire brief within the allotted two hours, but to make a start and continue outside of the class.

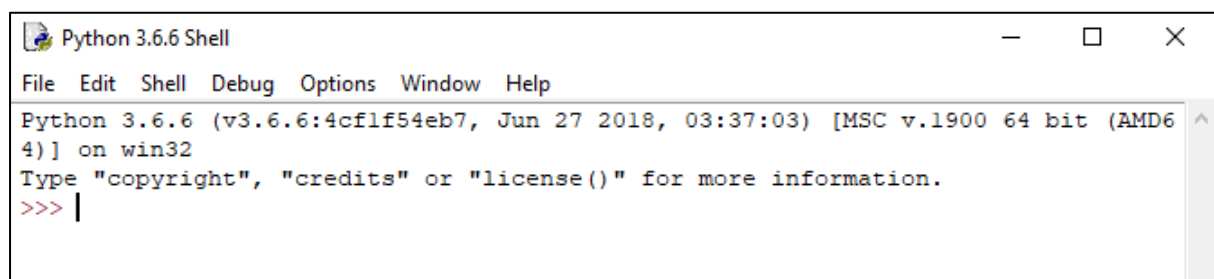
Attempt to complete this set of tasks before your next tutorial. Any issues seek help from your tutors.

Introduction

The learning aims of this session are to familiarise you with the IDLE environment and enable you to write simple Python modules using the `input()` and `print()` functions.

Launching IDLE

- **Linux** (MATE desktop) go to the Applications top menu | scroll to Programming | click IDLE.
- **MacOS** open Launchpad and search "IDLE". Click on the icon.
- **Windows** press the Windows key for the start menu and select All apps | scroll to the Python 36 folder | and click the IDLE application.



Briefly review the menu drop downs. Find out what version of IDLE is in use by selecting Help | About IDLE.

Accessing Help Documentation

Now select [Help | Python Docs](#).

This will access the IDLE shell help documentation for Python. This will be a useful resource that you should refer to as you develop your Python programming knowledge.

What is the short cut key for accessing this documentation?

Configuring IDLE

The IDLE environment can be customised to a limited extent through the configuration menu.

[Select Options | Configure IDLE](#)

Now see if you can:

- Reset the font size to 14pt
- Reset the window size to 100 by 60 (General tab)

You will need to re-launch IDLE before these take effect.

Keyboard Shortcuts

There are some useful short cut keys for managing the IDLE shell that you may wish to use as your experience with the environment grows:

[Under the Keys tab, investigate what default short cut keys are available.](#)

Note you can use <ALT> x to syntax check your code!

IDLE Interactive

The IDLE shell prompt is >>> Type the following commands without it. For each new line push Enter.

```
>>> import keyword
>>> keyword.kwlist
```

These are Python's reserved words (you cannot use these as identifiers or variable names).

Now type the following at the prompt:

```
>>> Hello
```

Note the red error message.

There are several pieces of information given to help identify the error including the line number and the nature of the error.

Always try to understand what the error message is trying to convey so that you can correct your code.

What was the problem in this case?

Now type the following at the prompt:

```
>>> print(Hello)
```

Why did that not work?

Now type the following at the prompt:

```
>>> print("Hello")
```

Why did that work?

Now type the following at the prompts, note the responses and discuss why.

```
>>> type(print)
>>> type(Hello)
>>> type("Hello")
```

IDLE Shell (Types)

Data structures are used to store data for programs to use. Some simple structures are illustrated below.

Type the following at the IDLE prompt:

```
>>> v = 10
>>> w = 2.7
>>> x = "Hello"
>>> y = chr(97)
```

Now predict the outcome from each of the following type statements before executing:

```
>>> type(v)
>>> type(w)
>>> type(x)
>>> type(y)
```

What is the output if you type the following?

```
>>> print(y)
```

Make sure you understand why the output results from each case (ask peers or tutors).

IDLE Shell (Operators)

Predict what you think will happen in each of the following cases if you type the following at the prompt (and then try it):

<pre>>>> x = 11 >>> x = x + 1 >>> print(x)</pre>	<pre>>>> y = 11 >>> y = y + 1</pre>	<pre>>>> z = "23" >>> z = "23" + "4" >>> print(z)</pre>
<pre>>>> x = 2 >>> y = 3 >>> z = y / x >>> print(z)</pre>	<pre>>>> x = 2 >>> y = 3 >>> z = y / 2 >>> print(z)</pre>	<pre>>>> x = 7 >>> y = x * x >>> print(y)</pre>
<pre>>>> x = 4 >>> y = x ** x >>> print(y)</pre>		

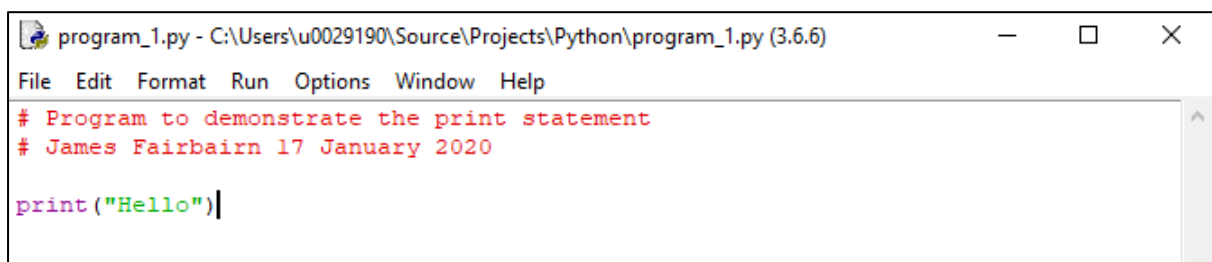
Modules: program_1.py

A module is simply a text file storing your program code.

To create a new module select **File | New File**.

This will open a text editor window into which you can type your program statements.

Type the following into the text editor:



```
program_1.py - C:\Users\u0029190\Source\Projects\Python\program_1.py (3.6.6)
File Edit Format Run Options Window Help
# Program to demonstrate the print statement
# James Fairbairn 17 January 2020

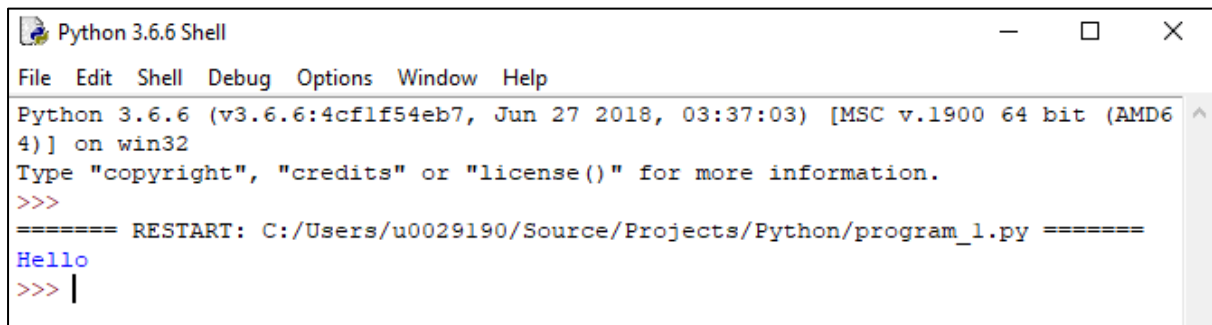
print("Hello")|
```

Save your module (program) by selecting **File | Save**.

Give it a suitable name (e.g. program_1.py).

Run your module (program) by selecting **Run | Run Module**.

Your module should now run and display the following output.



```
Python 3.6.6 Shell
File Edit Shell Debug Options Window Help
Python 3.6.6 (v3.6.6:4cflf54eb7, Jun 27 2018, 03:37:03) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/u0029190/Source/Projects/Python/program_1.py =====
Hello
>>> |
```

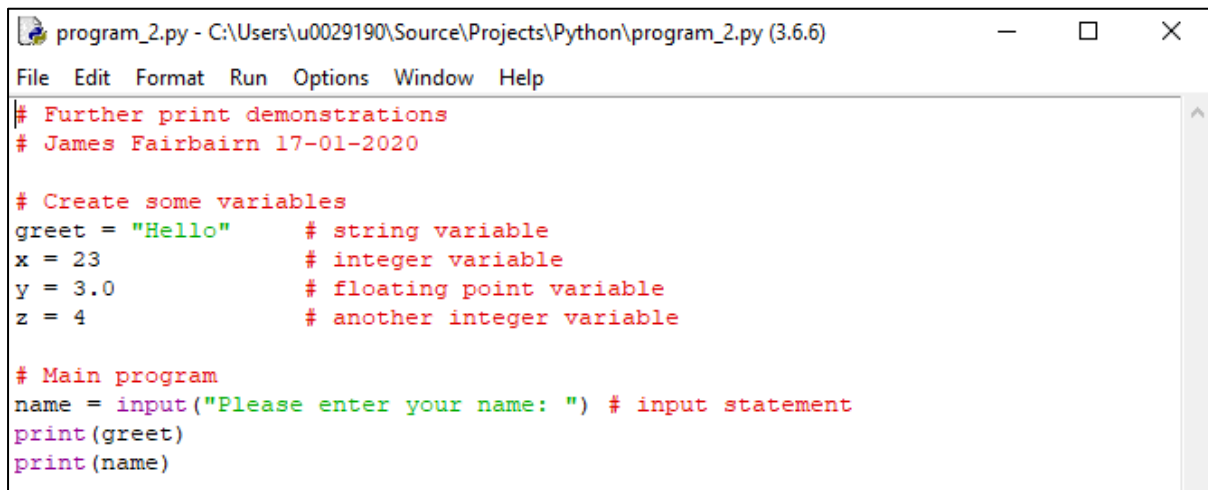
Note the shortcut key for Run Module (F5).

Note also, if you do not save your module, you will be prompted to save it.

You can access the module later by choosing **File | Open** in the IDLE shell and navigating to it.

Module Practice: program_2.py

Create a new module and type the following:

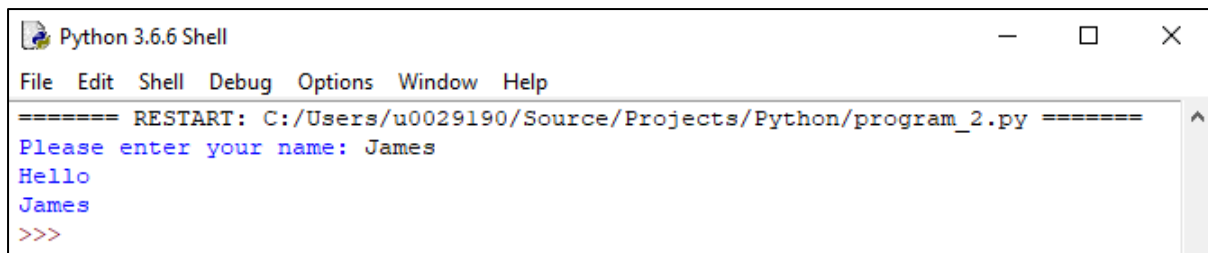


```
program_2.py - C:\Users\u0029190\Source\Projects\Python\program_2.py (3.6.6)
File Edit Format Run Options Window Help
# Further print demonstrations
# James Fairbairn 17-01-2020

# Create some variables
greet = "Hello"      # string variable
x = 23               # integer variable
y = 3.0              # floating point variable
z = 4                # another integer variable

# Main program
name = input("Please enter your name: ") # input statement
print(greet)
print(name)
```

Now save and run your module. It should produce the output below.



```
Python 3.6.6 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/Users/u0029190/Source/Projects/Python/program_2.py =====
Please enter your name: James
Hello
James
>>>
```

How can we modify the program to print the greeting and your name on the *same* line?

Modify the final two print statements too:

```
print(greet, name)
```

Now save and run your module. Check the greeting and your name is on the same line.

You can also achieve this output using the `end=""` argument:

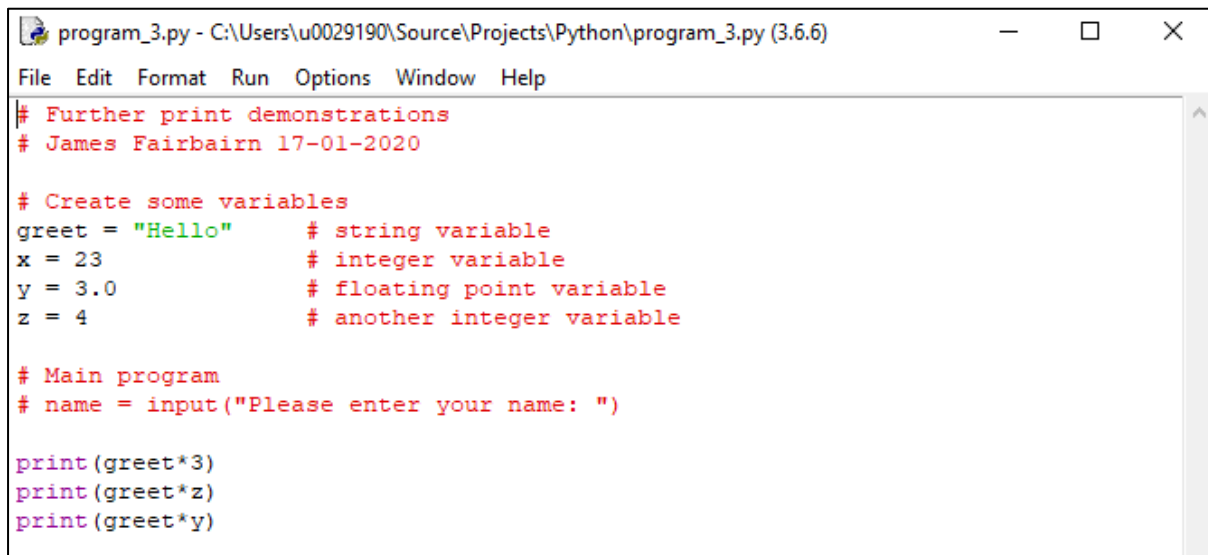
```
print(greet, end="")
print(name)
```

Further extend the program by adding these print statements to the bottom of the file:

```
print(x*y)                # multiply two numbers
print(greet*3)            # multi print a string
print("23 divide by 3.0 = ", x/y)    # divide two numbers
print("23 integer divide by 3.0 = ", x//y) # integer division
```

Module Practice: program_3.py

Save a copy of *program_2.py* as *program_3.py*. Modify *program_3.py* to the following:



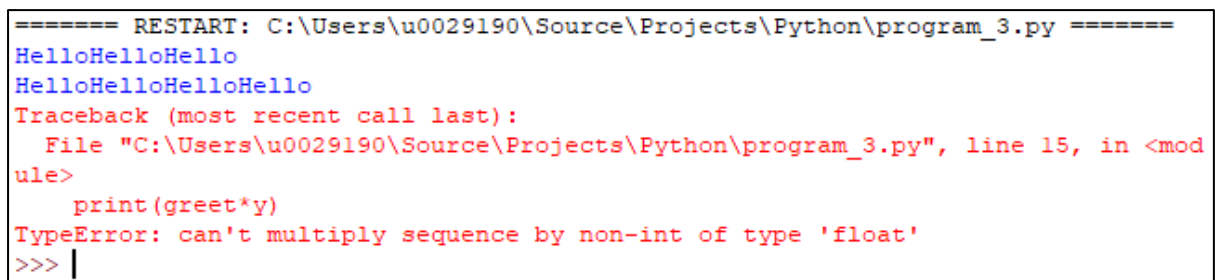
```
program_3.py - C:\Users\u0029190\Source\Projects\Python\program_3.py (3.6.6)
File Edit Format Run Options Window Help
# Further print demonstrations
# James Fairbairn 17-01-2020

# Create some variables
greet = "Hello"      # string variable
x = 23               # integer variable
y = 3.0              # floating point variable
z = 4                # another integer variable

# Main program
# name = input("Please enter your name: ")

print(greet*3)
print(greet*z)
print(greet*y)
```

Running the module will yield the following output:

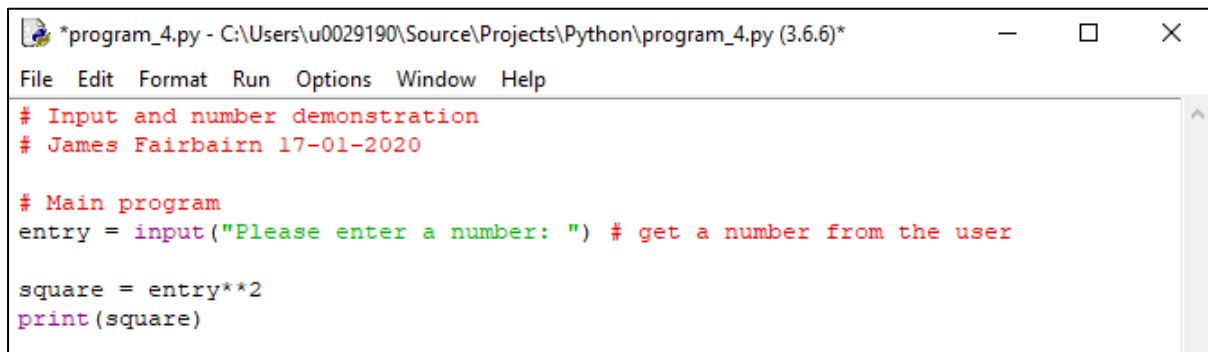


```
===== RESTART: C:\Users\u0029190\Source\Projects\Python\program_3.py =====
HelloHelloHello
HelloHelloHelloHello
Traceback (most recent call last):
  File "C:\Users\u0029190\Source\Projects\Python\program_3.py", line 15, in <mod
ule>
    print(greet*y)
TypeError: can't multiply sequence by non-int of type 'float'
>>> |
```

Which line of code caused the trace-back error? Why?

Module Practice: program_4.py

Create a new program_4.py module and type the following:

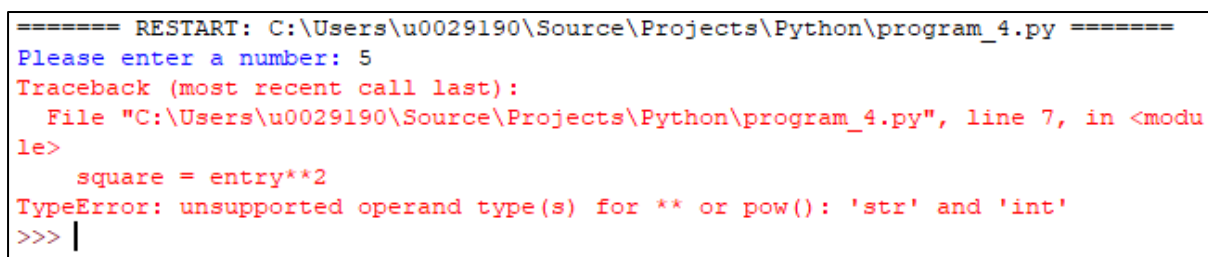


```
*program_4.py - C:\Users\u0029190\Source\Projects\Python\program_4.py (3.6.6)*
File Edit Format Run Options Window Help
# Input and number demonstration
# James Fairbairn 17-01-2020

# Main program
entry = input("Please enter a number: ") # get a number from the user

square = entry**2
print(square)
```

Run your module and you should receive the following output:



```
===== RESTART: C:\Users\u0029190\Source\Projects\Python\program_4.py =====
Please enter a number: 5
Traceback (most recent call last):
  File "C:\Users\u0029190\Source\Projects\Python\program_4.py", line 7, in <module>
    square = entry**2
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
>>> |
```

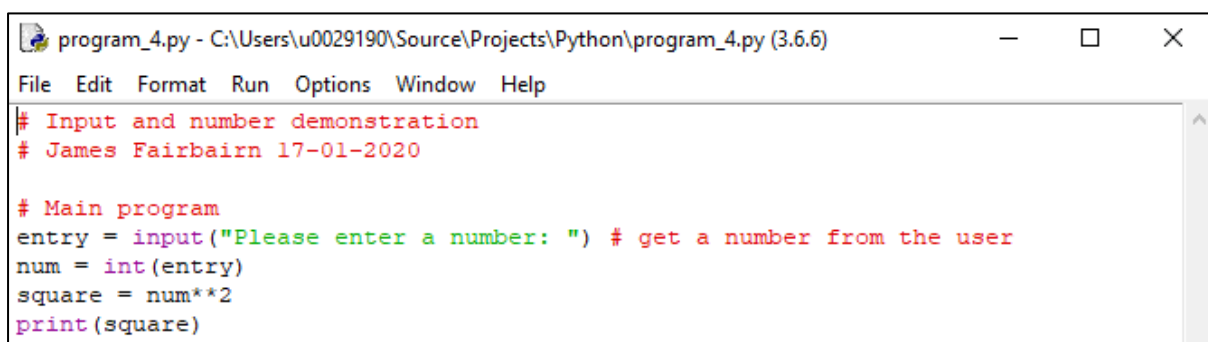
Why has your program failed?

Explanation

When using the `input()` statement to acquire data from a user, Python treats the `input()` literally as a set of characters (a character string) from the keyboard.

This can be corrected by *converting* the type from a string to an integer by casting.

Amend your code as shown below and try again:



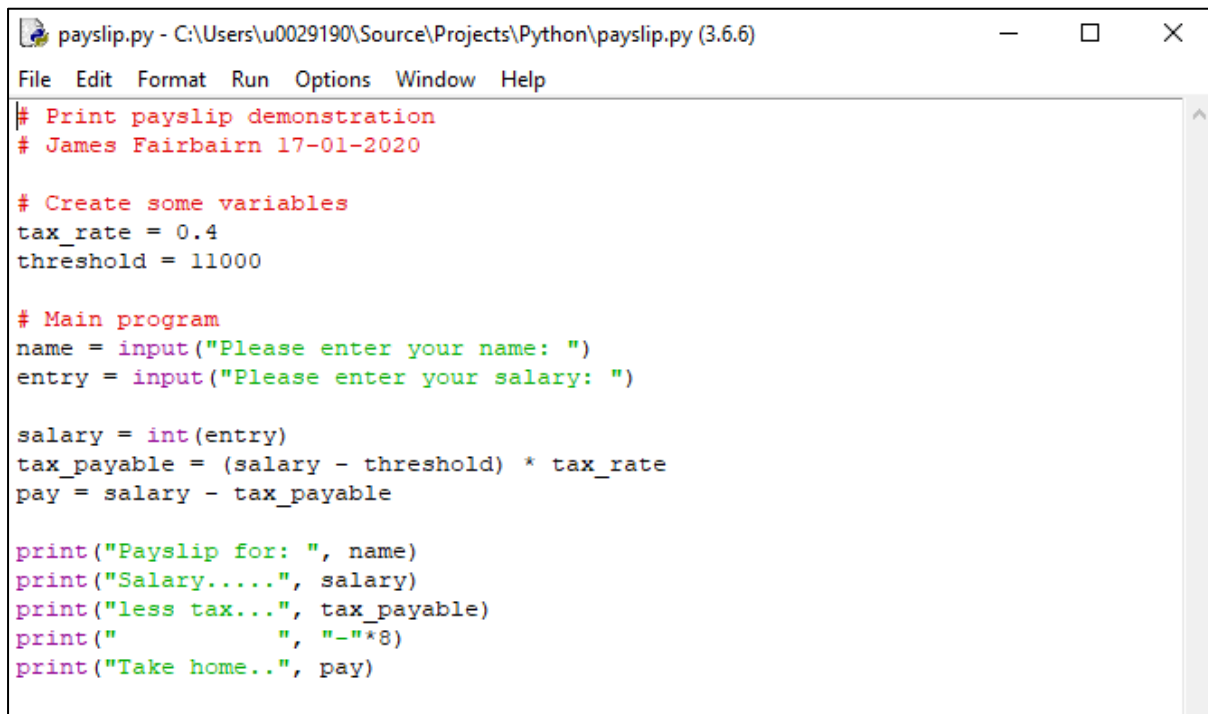
```
program_4.py - C:\Users\u0029190\Source\Projects\Python\program_4.py (3.6.6)
File Edit Format Run Options Window Help
# Input and number demonstration
# James Fairbairn 17-01-2020

# Main program
entry = input("Please enter a number: ") # get a number from the user
num = int(entry)
square = num**2
print(square)
```

What did we do to correct the problem?

Pay slip Program: payslip.py

Now for a more complex example. Create a new module and save as *payslip.py*.



```
payslip.py - C:\Users\u0029190\Source\Projects\Python\payslip.py (3.6.6)
File Edit Format Run Options Window Help
# Print payslip demonstration
# James Fairbairn 17-01-2020

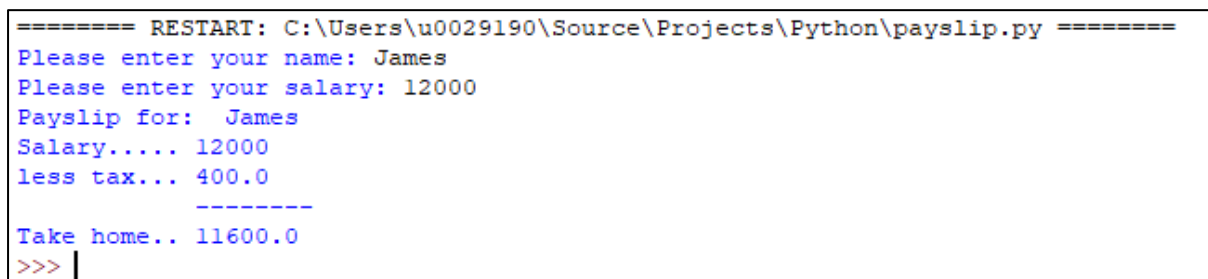
# Create some variables
tax_rate = 0.4
threshold = 11000

# Main program
name = input("Please enter your name: ")
entry = input("Please enter your salary: ")

salary = int(entry)
tax_payable = (salary - threshold) * tax_rate
pay = salary - tax_payable

print("Payslip for: ", name)
print("Salary.....", salary)
print("less tax...", tax_payable)
print("          ", "-"*8)
print("Take home..", pay)
```

Save and Run the module using an initial salary of 12000.



```
===== RESTART: C:\Users\u0029190\Source\Projects\Python\payslip.py =====
Please enter your name: James
Please enter your salary: 12000
Payslip for: James
Salary..... 12000
less tax... 400.0
          -----
Take home.. 11600.0
>>> |
```

The module works, but is poorly presented.

Modify the `print` code to take advantage of the string padding method `rjust`.

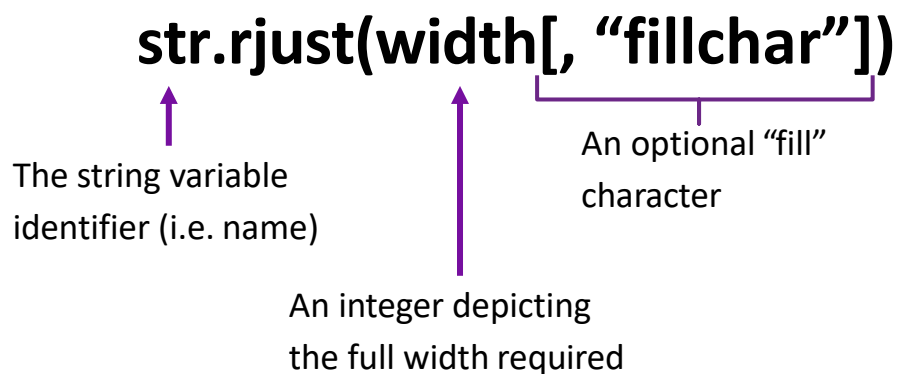
```
print("\n")
print("Payslip for", name.rjust(8, " "))
print("Salary.....", str(salary).rjust(8, " "))
print("less tax...", str(tax_payable).rjust(8, " "))
print("          ", "-"*8)
print("Take home..", str(pay).rjust(8, " "))
```

Now Save a Run the module:

```
===== RESTART: C:\Users\u0029190\Source\Projects\Python\payslip.py =====
Please enter your name: James
Please enter your salary: 12000

Payslip for      James
Salary.....   12000
less tax...     400.0
-----
Take home..    11600.0
```

Much better! Let's explore the `rjust` method.



This method can be invoked on a string to pad it to facilitate better layout.

<code>name = "Monty"</code>	A string variable containing the value Monty
<code>name.rjust(10, ".")</code>	Pads Monty with 5 preceding dots (because the width is 10 and Monty already has 5 characters)
<code>"Monty".rjust(10, ".")</code>	Padding is applied directly to the literal (string).
<code>name.rjust(10)</code>	If the fill character is unspecified, the default fill is a space.

Note

1. If the width specified is less than the length of the string, the original string is returned.
2. Use `ljust()` method to pad with trailing characters.
3. The original string value is not changed unless you assign it to the justified string:
`name = name.rjust(8, "-")`

Sum and Product: sum_and_product.py

Create a module to input three integers and print out their sum and product as illustrated below:

```
==== RESTART: C:\Users\u0029190\Source\Projects\Python\sum_and_product.py ====
Integer 1: 3
Integer 2: 4
Integer 3: 5
The sum is 12 and the product is 60
Done!
```

Conversions: conversion.py

Convert km h^{-1} to m s^{-1}

Create a module to convert km h^{-1} to m s^{-1} . Your program should ask the user for the speed in km h^{-1} in and then print out its equivalent in m s^{-1} . You may need to research how to do the conversion and make some brief notes prior to coding.

Use the following data to test your program:

72 km /h is equal to 20 m/s

What is 108 km/h in m/s?

Testing

By trying different values in your program, determine how many km/h is 40 m/s.

Convert m/s to km/h

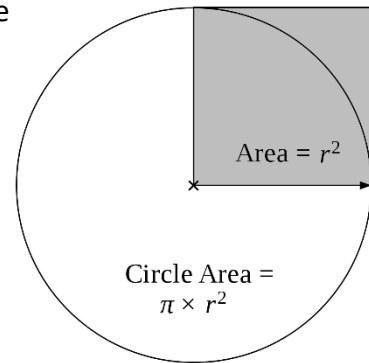
Create a new module to convert m/s to km/h and test your program by using the results from the previous program.

Area of a Circle: circlearea.py

Write a program to calculate and output the area of a circle.

Hint: Python comes with the value of the mathematical constant pi (π) built in as `math.pi`. You'll be making heavy use of this over the next set of tasks. Remember, you need to add `import math` at the top of your file to be able to use this!

1. Create a new module file called "circlearea.py".
2. Write Python code in your module to ask the user for the radius of a circle. The program will then calculate, store and output the area of the circle to the user. You can find the formula for the area of a circle here:
<http://www.calculateme.com/cArea/AreaOfCircle.htm>
3. Modify the program to also store and output the circumference (perimeter) of the circle. You can find the formula for this here:
<https://www.mathplanet.com/education/pre-algebra/more-about-equation-and-inequalities/calculating-the-circumference-of-a-circle>



Area of Ellipse: ellipsearea.py

Write a program to calculate and output the area of an ellipse (oval).

1. Start with your solution to question 1. Copy it to a new file called "ellipsearea.py".
2. Modify the program to take a second radius for an ellipse. The program should then calculate, store and output the area of the ellipse to the user. You can find the formula for the area of an ellipse here:
<http://www.calculateme.com/cArea/AreaOfEllipse.htm>

Volume of Cylinder: cylindervolume.py

Write a program to calculate and output the volume of a cylinder.

1. Create a fresh module called "cylindervolume.py".
2. Write a program to compute the area of a cylinder, asking the user for the radius and height. You can find the formula for the volume of a cylinder here:
<http://www.calculateme.com/cVolume/VolumeOfCylinder.htm>

Square Footage of a House

Write a program to calculate and output the square footage of a house.

1. Write a program that asks the user for the width and length of 4 rooms of a house, you can assume that the rooms are rectangular (i.e. no alcoves, etc.). Once all the information has been collected, the program will then display the square footage per room and then the total square footage for the entire house.
2. Copy your solution and modify it to also ask the height of each room so that it can calculate the volume for each room, and the entire house.
3. Use the usual $\text{width} * \text{length}$ formula for calculating the area of a rectangle and $\text{width} * \text{length} * \text{height}$ for calculating the volume of a cuboid.

Fahrenheit-Celsius Conversion

Write a program that prompts the user to input a temperature in Fahrenheit. The program will then output the temperature in Celsius. The format of the output is required to be:

```
>>> 71 Degrees Fahrenheit is 21.67 degrees Celsius
```

Get the formula for the conversion here:

<http://www.calculateme.com/cTemperature/FahrenheitToCelsius.htm>

Celsius-Fahrenheit Conversion

Write a program that prompts the user to input a temperature in Celsius. The program will then output the temperature in Fahrenheit. The format of the output is required to be:

```
>>> 22 Celsius is 71.6 degrees Fahrenheit.
```

Get the formula for the conversion here:

<http://www.calculateme.com/cTemperature/CelsiusToFahrenheit.htm>

Document History

Revision 0 (24-Sep-20): This is the initial version of the 2020/21 exercise.