Computer Science

WELCOME TO COUNTESTHORPE LEYSLAND COLLEGE

Year 11 ->Year 12

Purpose of Workbook

To become familiar of the expectations of the Computer Science department.

To gain a better understanding of Computer Science and prepare for the term ahead.

To begin a set of taster tasks in preparation for the Computer Science course.

The Department

At Countesthorpe Leysland Community College, the Computing and IT department employs 3 full time members of staff. All Three are specialists in the Computing field and all Three members have expertise to help you to achieve the best possible grade in Computer Science.

- Mr Ford
- Mr Chauhan
- Mr Holmes

Introduction to Course

The OCR Computer Science course encourages you to develop the understanding and application of the core concepts in computer science. You will analyse problems in computational terms and devise creative solutions by designing, writing, testing and evaluating programs. The OCR course was created with the input of actual Computer Science teachers so what you learn is important and the concepts are also what is required in industry.

Lesson structure

9 Hours per fortnight

Always in IT room

Two teachers – Mr Holmes & Mr Chauhan

Homework expected to be completed in none contact periods as well as at home

Course breakdown

- **Unit 01** –Computing Principles

This is a **Exam** unit worth **40%.**

- **Unit 02** – Algorithms and Problem Solving

This is a **Exam** unit **40%** .

- **Unit 03** – Individual Project

This is a **Controlled Assessment** unit **20%** .

}

- **100%**

=A-Level

Component 01-Computing Principles

EXAMINED ON FOLLOWING TOPICS…

- Components of a computer system
- Systems software
- Software development
- Exchanging data
- Networks
- Data types
- Data structures
- Boolean Algebra
- Computer ethics and law

1 Hours 30 Mins

Written paper

Component 02 -ALGORITHMS & PROBLEM SOLVING

EXAMINED ON FOLLOWING TOPICS…

- Computational thinking
- Programming techniques
- Algorithms

1 Hours 30 Mins

Written paper

Component 03 -individual project

What is the individual project…

Students will choose a program to create of their own choice. The program will be coded in a high level programming language and there will be a write up to document the project.

70 Marks

CONTROLLED ASSESSMENT

Recommended LANGUAGE

PYTHON

+ Familiar
+ Can be made OOP
+ Almost identical format to OCRs Pseudocode
+ Can create GUIs and Games
+ Is used in industry (Nokia, Google, Netflix)
+ Most have used this for GCSE Computer Science

If you would like to use another Programming Language, speak to you tutor first.

RESOURCES

www.python.org

www.codeacademy.com

https://www.tutorialspoint.com/tutorialslibrary.htm

Software required

You are not expected to pay for any specialist software as there are many open source or free programs available to complete the course. You are expected to download and install IDLE which is a free program to use during the course and it will also enable you to complete some of the tasks you are expected to complete. The software is available at:

https://www.python.org/downloads/

IDLE help

Use the following websites to help you if you are struggling to use IDLE

https://realpython.com/python-idle/

https://www.pitt.edu/~naraehan/python3/getting_started_win_first_try.html

https://www.dummies.com/programming/python/how-to-start-idle-in-python/

Tasks

There are 8 Word documents numbered 1 -8, your task is to follow the instructions and complete the programming tasks. The last tasks is focussed on career progression (see next slide) Some tasks will be easier than others, but some may take even longer to solve. Complete the tasks and use the print screen button on the keyboard and paste into a Word document to show evidence how you completed the tasks. Ensure the screen shots are big enough to see the programming code.

Activities specification map

The tasks you complete will benefit all Three components of the Computer Science course. The table below shows which units/components the activities will help aid.

	Component 1									Component 2			Component 3
	Unit 1	Unit2	Unit 3	Unit 4	Unit 5	Unit 6	Uni 7	Unit 8	Unit 9	Unit 10	Unit 11	Unit 12	Project
Lists				✓							✓		✓
String Formatting	✓			✓	✓	✓	✓				✓		✓
String Operations				✓				✓			✓	✓	✓
Conditions				✓			✓	✓		✓	✓	✓	✓
Loops			✓								✓		✓
Functions		✓	✓							✓	✓		✓
Classes and Objects			✓								✓		✓
Language Investigation			✓				✓				✓		✓

Career task

There are many different fields in the Computing sector. Some of you may want to be a programmer whereas others may want to pursue a career in computer forensics. Many universities offer a range of courses covering different topics and subject areas. Your task is to research Two different universities (listed below) and pick Two different courses and determine:

- Structure of course
- Topics taught
- Assessment methods
- Length of course
- Entry requirements
- Job opportunities
- Potential salary (you may need to research this)

Write your findings on a Word document

De Montfort University

Manchester University

Deadline

All tasks need to be complete before week commencing 22nd June, please bring to the first transition lesson. Below is a recommended schedule to complete all the work. It is suggested as some tasks may take longer than others.

Task	Week Commencing
Lists	1/6/20
String Formatting	1/6/20
String Operations	1/6/20
Conditions	8/6/20
Loops	8/6/20
Functions	8/6/20
Classes and Objects	8/6/20
Language Investigation	15/6/20
Career Task	15/6/20

Support

For general sixth form enquires, Email

6thform@clcc.college

For computer science enquires, Email

schauhan@clcc.college

cholmes@clcc.college

Lists

Lists are very similar to arrays. They can contain any type of variable, and they can contain as many variables as you wish. Lists can also be iterated over in a very simple manner. Here is an example of how to build a list.

script.py

```
1   mylist = []
2   mylist.append(1)
3   mylist.append(2)
4   mylist.append(3)
5   print(mylist[0]) # prints 1
6   print(mylist[1]) # prints 2
7   print(mylist[2]) # prints 3
8
9   # prints out 1,2,3
10  for x in mylist:
11      print(x)
```

IPython Shell

```
1
2
3
1
2
3

In [1]:
```

Accessing an index which does not exist generates an exception (an error).

script.py

```
1   mylist = [1,2,3]
2   print(mylist[10])
```

IPython Shell

```
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
    print(mylist[10])
IndexError: list index out of range
```

Exercise

Using a Python Editing software, type in the code below.

Once typed in, **you will need to add numbers and strings to the correct lists using the "append" list method.**

You must add the numbers 1,2 and 3 to the "numbers" list

And you must add the words 'hello' and 'world' to the string variable

You will also have to fill in the variable second_name with the second name in the names list, using the brackets operator []

Screen shot your completed code (not the output!) into a Word document with the title "Lists"

Python 3.4.1: Untitled

File Edit Format Run Options Windows Help

```
numbers = []
strings = []
names = ["John", "Eric", "Jessica"]

# write your code here
second_name = None


# this code should write out the filled arrays and the second name in the names list (Eric).
print(numbers)
print(strings)
print("The second name on the names list is %s" % second_name)
```

Your output should resemble the screen shot below:

Python 3.4.1 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:45:13
D64)] on win32
Type "copyright", "credits" or "license()" for more info
>>> ========== RESTART ==========
>>>
[1, 2, 3]
['hello', 'world']
The second name on the names list is Eric
>>>
```

String Formatting

Python uses C-style string formatting to create new, formatted strings. The "%" operator is used to format a set of variables enclosed in a "tuple" (a fixed size list), together with a format string, which contains normal text together with "argument specifiers", special symbols like "%s" and "%d".

Let's say you have a variable called "name" with your user name in it, and you would then like to print(out a greeting to that user.)

script.py

```
1  # This prints out "Hello, John!"
2  name = "John"
3  print("Hello, %s!" % name)
```

IPython Shell

```
Hello, John!

In [1]:
```

To use two or more argument specifiers, use a tuple (parentheses):

script.py

```
1  # This prints out "John is 23 years old."
2  name = "John"
3  age = 23
4  print("%s is %d years old." % (name, age))
```

IPython Shell

```
John is 23 years old.

In [1]:
```

Any object which is not a string can be formatted using the %s operator as well. The string which returns from the "repr" method of that object is formatted as the string. For example:

script.py

```
1  # This prints out: A list: [1, 2, 3]
2  mylist = [1,2,3]
3  print("A list: %s" % mylist)
```

IPython Shell

```
A list: [1, 2, 3]

In [1]:
```

Here are some basic argument specifiers you should know:

`%s - String (or any object with a string representation, like numbers)`

`%d - Integers`

`%f - Floating point numbers`

`%.<number of digits>f - Floating point numbers with a fixed amount of digits to the right of the dot.`

`%x/%X - Integers in hex representation (lowercase/uppercase)`

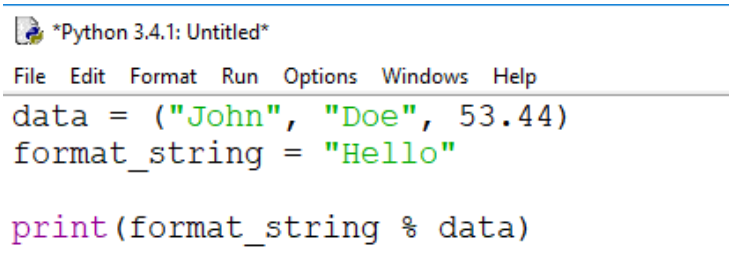
Exercise

Using a Python Editing software, type in the code below.

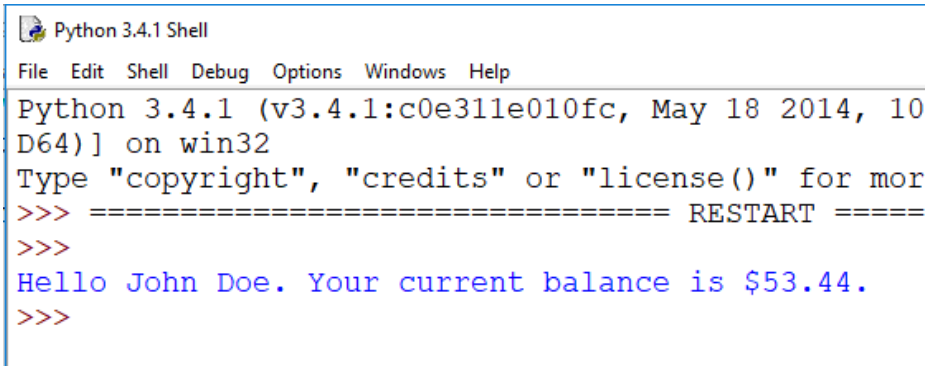
Once typed in, **you will need to write a format string which prints out the data using the following syntax:**

Hello John Doe. Your current balance is $53.44.

Screen shot your completed code (not the output!) into a Word document with the title "String Formatting"



Your output should resemble the screen shot below:



Simple (but cool) String Operations

Strings are bits of text. They can be defined as anything between quotes:

script.py

```
1  astring = "Hello world!"
2  astring2 = 'Hello world!'
```

IPython Shell

```
In [1]:
```

As you can see, the first thing you learned was printing a simple sentence. This sentence was stored by Python as a string. However, instead of immediately printing strings out, we will explore the various things you can do to them. You can also use single quotes to assign a string. However, you will face problems if the value to be assigned itself contains single quotes.For example to assign the string in these bracket(single quotes are ' ') you need to use double quotes only like this

script.py

```
1  astring = "Hello world!"
2  print("single quotes are ' '")
3
4  print(len(astring))
```

IPython Shell

```
single quotes are ' '
12

In [1]:
```

That prints out 12, because "Hello world!" is 12 characters long, including punctuation and spaces.

script.py

```
1  astring = "Hello world!"
2  print(astring.index("o"))
```

IPython Shell

```
4

In [1]:
```

That prints out 4, because the location of the first occurrence of the letter "o" is 4 characters away from the first character. Notice how there are actually two o's in the phrase - this method only recognizes the first.

But why didn't it print out 5? Isn't "o" the fifth character in the string? To make things more simple, Python (and most other programming languages) start things at 0 instead of 1. So the index of "o" is 4.

script.py

```
1  astring = "Hello world!"
2  print(astring.count("l"))
```

IPython Shell

```
3

In [1]:
```

For those of you using silly fonts, that is a lowercase L, not a number one. This counts the number of l's in the string. Therefore, it should print 3.

script.py

```
1  astring = "Hello world!"
2  print(astring[3:7])
```

IPython Shell

```
lo w

In [1]:
```

This prints a slice of the string, starting at index 3, and ending at index 6. But why 6 and not 7? Again, most programming languages do this - it makes doing math inside those brackets easier.

If you just have one number in the brackets, it will give you the single character at that index. If you leave out the first number but keep the colon, it will give you a slice from the start to the number you left in. If you leave out the second number, it will give you a slice from the first number to the end.

You can even put negative numbers inside the brackets. They are an easy way of starting at the end of the string instead of the beginning. This way, -3 means "3rd character from the end".

script.py

```
1  astring = "Hello world!"
2  print(astring[3:7:2])
```

IPython Shell

```
l

In [1]:
```

This prints the characters of string from 3 to 7 skipping one character. This is extended slice syntax. The general form is [start:stop:step].

script.py

```
1  astring = "Hello world!"
2  print(astring[3:7])
3  print(astring[3:7:1])
```

IPython Shell

```
lo w
lo w

In [1]:
```

Note that both of them produce same output

There is no function like strrev in C to reverse a string. But with the above mentioned type of slice syntax you can easily reverse a string like this

script.py

```
1  astring = "Hello world!"
2  print(astring[::-1])
```

IPython Shell

```
!dlrow olleH

In [1]:
```

This

script.py

```
1  astring = "Hello world!"
2  print(astring.upper())
3  print(astring.lower())
```

IPython Shell

```
In [1]:
```

These make a new string with all letters converted to uppercase and lowercase, respectively.

script.py

```
1  astring = "Hello world!"
2  print(astring.startswith("Hello"))
3  print(astring.endswith("asdfasdfasdf"))
```

IPython Shell

```
In [1]:
```

This is used to determine whether the string starts with something or ends with something, respectively. The first one will print True, as the string starts with "Hello". The second one will print False, as the string certainly does not end with "asdfasdfasdf".

script.py

```
1  astring = "Hello world!"
2  afewwords = astring.split(" ")
```

IPython Shell

```
In [1]:
```

This splits the string into a bunch of strings grouped together in a list. Since this example splits at a space, the first item in the list will be "Hello", and the second will be "world!".

Exercise

Using a Python Editing software, type in the code below.

Once typed in, Try and fix the code to print out the correct information by **changing the string 's'**

Screen shot your completed code (not the output!) into a Word document with the title "String Operations"

File Edit Format Run Options Windows Help

```
s = "Hey there! what should this string be?"
# Length should be 20
print("Length of s = %d" % len(s))

# First occurrence of "a" should be at index 8
print("The first occurrence of the letter a = %d" % s.index("a"))

# Number of a's should be 2
print("a occurs %d times" % s.count("a"))

# Slicing the string into bits
print("The first five characters are '%s'" % s[:5]) # Start to 5
print("The next five characters are '%s'" % s[5:10]) # 5 to 10
print("The thirteenth character is '%s'" % s[12]) # Just number 12
print("The characters with odd index are '%s'" %s[1::2]) #(0-based indexing)
print("The last five characters are '%s'" % s[-5:]) # 5th-from-last to end

# Convert everything to uppercase
print("String in uppercase: %s" % s.upper())

# Convert everything to lowercase
print("String in lowercase: %s" % s.lower())

# Check how a string starts
if s.startswith("Str"):
    print("String starts with 'Str'. Good!")

# Check how a string ends
if s.endswith("ome!"):
    print("String ends with 'ome!'. Good!")

# Split the string into three separate strings,
# each containing only a word
print("Split the words of the string: %s" % s.split(" "))
```

Your output should resemble the screen shot below:

Python 3.4.1 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:45:13) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
Length of s = 20
The first occurrence of the letter a = 8
a occurs 2 times
The first five characters are 'Strin'
The next five characters are 'gs ar'
The thirteenth character is 'a'
The characters with odd index are 'tig r wsm!'
The last five characters are 'some!'
String in uppercase: STRINGS ARE AWESOME!
String in lowercase: strings are awesome!
String starts with 'Str'. Good!
String ends with 'ome!'. Good!
Split the words of the string: ['Strings', 'are', 'awesome!']
>>>
```

Conditions

Python uses boolean variables to evaluate conditions. The boolean values True and False are returned when an expression is compared or evaluated. For example:

script.py

```
x = 2
print(x == 2) # prints out True
print(x == 3) # prints out False
print(x < 3) # prints out True
```

IPython Shell

```
True
False
True

In [1]:
```

Notice that variable assignment is done using a single equals operator "=", whereas comparison between two variables is done using the double equals operator "==". The "not equals" operator is marked as "!=".

Boolean operators

The "and" and "or" boolean operators allow building complex boolean expressions, for example:

script.py

```
name = "John"
age = 23
if name == "John" and age == 23:
    print("Your name is John, and you are also 23 years old.")

if name == "John" or name == "Rick":
    print("Your name is either John or Rick.")
```

IPython Shell

```
Your name is John, and you are also 23 years old.
Your name is either John or Rick.

In [1]:
```

The "in" operator

The "in" operator could be used to check if a specified object exists within an iterable object container, such as a list:

script.py

```
name = "John"
if name in ["John", "Rick"]:
    print("Your name is either John or Rick.")
```

IPython Shell

```
Your name is either John or Rick.

In [1]:
```

Python uses indentation to define code blocks, instead of brackets. The standard Python indentation is 4 spaces, although tabs and any other space size will work, as long as it is consistent. Notice that code blocks do not need any termination.

Here is an example for using Python's "if" statement using code blocks:

script.py

```
statement = False
another_statement = True
if statement is True:
    # do something
    pass
elif another_statement is True: # else if
    # do something else
    pass
else:
    # do another thing
    pass
```

IPython Shell

```
In [1]:
```

For example:

script.py

```
x = 2
if x == 2:
    print("x equals two!")
else:
    print("x does not equal to two.")
```

IPython Shell

```
x equals two!

In [1]:
```

A statement is evaulated as true if one of the following is correct: 1. The "True" boolean variable is given, or calculated using an expression, such as an arithmetic comparison. 2. An object which is not considered "empty" is passed.

Here are some examples for objects which are considered as empty: 1. An empty string: "" 2. An empty list: [] 3. The number zero: 0 4. The false boolean variable: False

The 'is' operator

Unlike the double equals operator "==", the "is" operator does not match the values of the variables, but the instances themselves. For example:

script.py

```
x = [1,2,3]
y = [1,2,3]
print(x == y) # Prints out True
print(x is y) # Prints out False
```

IPython Shell

```
True
False

In [1]:
```

The "not" operator

Using "not" before a boolean expression inverts it:

script.py

```
print(not False) # Prints out True
print((not False) == (False)) # Prints out False
```

IPython Shell

```
True
False

In [1]:
```

Exercise

Using a Python Editing software, type in the code below. Once typed in, **change the variables in the first section so that each of the IF statements resolve as True**

Screen shot your completed code (not the output!) into a Word document with the title "Conditions"

Python 3.4.1: Untitled

File Edit Format Run Options Windows Help

```
# change this code
number = 10
second_number = 10
first_array = []
second_array = [1,2,3]

if number > 15:
    print("1")

if first_array:
    print("2")

if len(second_array) == 2:
    print("3")

if len(first_array) + len(second_array) == 5:
    print("4")

if first_array and first_array[0] == 1:
    print("5")

if not second_number:
    print("6")
```

Your output should resemble the screen shot below:

Python 3.4.1 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May
D64)] on win32
Type "copyright", "credits" or "licens
>>> ================================ R
>>>
>>> ================================ R
>>>
1
2
3
4
5
6
>>>
```

Loops

There are two types of loops in Python, for and while.

The "for" loop

For loops iterate over a given sequence. Here is an example:

script.py

```
primes = [2, 3, 5, 7]
for prime in primes:
    print(prime)
```

IPython Shell

```
2
3
5
7
```

For loops can iterate over a sequence of numbers using the "range" and "xrange" functions. The difference between range and xrange is that the range function returns a new list with numbers of that specified range, whereas xrange returns an iterator, which is more efficient. (Python 3 uses the range function, which acts like xrange). Note that the range function is zero based.

script.py

```
# Prints out the numbers 0,1,2,3,4
for x in range(5):
    print(x)

# Prints out 3,4,5
for x in range(3, 6):
    print(x)

# Prints out 3,5,7
for x in range(3, 8, 2):
    print(x)
```

IPython Shell

```
0
1
2
3
4
3
4
5
3
5
7

In [1]:
```

"while" loops

While loops repeat as long as a certain boolean condition is met. For example:

script.py

```
# Prints out 0,1,2,3,4

count = 0
while count < 5:
    print(count)
    count += 1  # This is the same as count = count + 1
```

IPython Shell

```
0
1
2
3
4

In [1]:
```

"break" and "continue" statements

break is used to exit a for loop or a while loop, whereas **continue** is used to skip the current block, and return to the "for" or "while" statement. A few examples:

script.py

```
# Prints out 0,1,2,3,4

count = 0
while True:
    print(count)
    count += 1
    if count >= 5:
        break

# Prints out only odd numbers - 1,3,5,7,9
for x in range(10):
    # Check if x is even
    if x % 2 == 0:
        continue
    print(x)
```

IPython Shell

```
0
1
2
3
4
1
3
5
7
9

In [1]:
```

can we use "else" clause for loops?

unlike languages like C,CPP.. we can use **else** for loops. When the loop condition of "for" or "while" statement fails then code part in "else" is executed. If **break** statement is executed inside for loop then the "else" part is skipped. Note that "else" part is executed even if there is a **continue** statement.

Here are a few examples:

script.py

```
# Prints out 0,1,2,3,4 and then it prints "count value reached 5"

count=0
while(count<5):
    print(count)
    count +=1
else:
    print("count value reached %d" %(count))

# Prints out 1,2,3,4
for i in range(1, 10):
    if(i%5==0):
        break
    print(i)
else:
    print("this is not printed because for loop is terminated because of break but not due to fail in condition")
```

IPython Shell

```
0
1
2
3
4
count value reached 5
1
2
3
4

In [1]:
```

Exercise

Using a Python Editing software, type in the code below. Once typed in, **loop through and print out all even numbers from the numbers list in the same order they are received.**

Don't print any numbers that come after 237 in the sequence

Screen shot your completed code (not the output!) into a Word document with the title "Loops"

Python 3.4.1: Untitled

File Edit Format Run Options Windows Help

```
numbers = [
    951, 402, 984, 651, 360, 69, 408, 319, 601, 485, 980, 507, 725, 547, 544,
    615, 83, 165, 141, 501, 263, 617, 865, 575, 219, 390, 984, 592, 236, 105, 942, 941,
    386, 462, 47, 418, 907, 344, 236, 375, 823, 566, 597, 978, 328, 615, 953, 345,
    399, 162, 758, 219, 918, 237, 412, 566, 826, 248, 866, 950, 626, 949, 687, 217,
    815, 67, 104, 58, 512, 24, 892, 894, 767, 553, 81, 379, 843, 831, 445, 742, 717,
    958, 609, 842, 451, 688, 753, 854, 685, 93, 857, 440, 380, 126, 721, 328, 753, 470,
    743, 527
]

# your code goes here
```

Your output should resemble the screen shot below:

Python 3.4.1 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 1
D64)] on win32
Type "copyright", "credits" or "license()" for mo
>>> ================================ RESTART ====
>>>
402
984
360
408
980
544
390
984
592
236
942
386
462
418
344
236
566
978
328
162
758
918
>>>
```

Functions

What are Functions?

Functions are a convenient way to divide your code into useful blocks, allowing us to order our code, make it more readable, reuse it and save some time. Also functions are a key way to define interfaces so programmers can share their code.

How do you write functions in Python?

As we have seen on previous tutorials, Python makes use of blocks.

A block is a area of code of written in the format of:

script.py

```
block_head:
    1st block line
    2nd block line
    ...
```

IPython Shell

```
In [1]:
```

Where a block line is more Python code (even another block), and the block head is of the following format: block_keyword block_name(argument1,argument2, ...) Block keywords you already know are "if", "for", and "while".

Functions in python are defined using the block keyword "def", followed with the function's name as the block's name. For example:

script.py

```
def my_function():
    print("Hello From My Function!")
```

IPython Shell

```
In [1]:
```

Functions may also receive arguments (variables passed from the caller to the function). For example:

script.py

```
def my_function_with_args(username, greeting):
    print("Hello, %s , From My Function!, I wish you %s"%
(username, greeting))
```

IPython Shell

```
In [1]:
```

Functions may return a value to the caller, using the keyword- 'return' . For example:

script.py

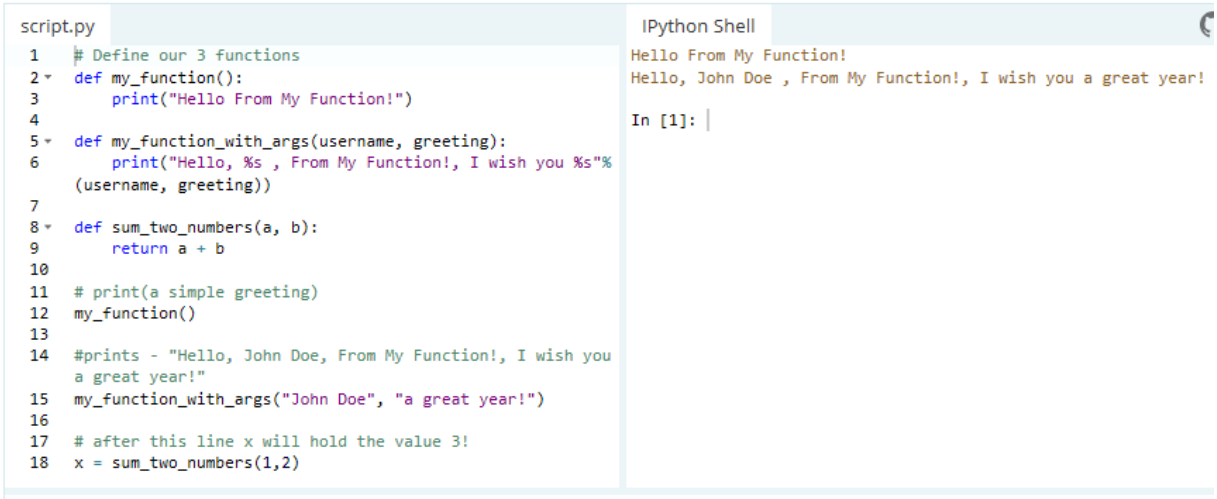
```
def sum_two_numbers(a, b):
    return a + b
```

IPython Shell

```
In [1]:
```

How do you call functions in Python?

Simply write the function's name followed by (), placing any required arguments within the brackets. For example, lets call the functions written above (in the previous example):



Exercise

Using a Python Editing software, type in the code below. Once typed in, **you'll use an existing function, and while adding your own to create a fully functional program**

1. Add a function named list_benefits() that returns the following list of strings:

   "More organised code", "More readable code", "Easier code reuse", "Allowing programmers to share and connect code together"

2. Add a function named build_sentence(info) which receives a single argument containing a string and returns a sentence starting with the given string and ending with the string:

   " is a benefit of functions!"

3. Run and see all functions work together!

Screen shot your completed code (not the output!) into a Word document with the title "Functions"

File Edit Format Run Options Windows Help

```
# Modify this function to return a list of strings as defined above
def list_benefits():
    pass

# Modify this function to concatenate to each benefit - " is a benefit of functions!"
def build_sentence(benefit):
    pass

def name_the_benefits_of_functions():
    list_of_benefits = list_benefits()
    for benefit in list_of_benefits:
        print(build_sentence(benefit))

name_the_benefits_of_functions()
```

Your output should resemble the screen shot below:

Python 3.4.1 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:45:13) [MSC v.1600 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
More organized code is a benefit of functions!
More readable code is a benefit of functions!
Easier code reuse is a benefit of functions!
Allowing programmers to share and connect code together is a benefit of functions!
>>>
```

Classes and Objects

Objects are an encapsulation of variables and functions into a single entity. Objects get their variables and functions from classes. Classes are essentially a template to create your objects.

A very basic class would look something like this:

script.py

```
class MyClass:
    variable = "blah"

    def function(self):
        print("This is a message inside the class.")
```

IPython Shell

```
In [1]:
```

We'll explain why you have to include that "self" as a parameter a little bit later. First, to assign the above class(template) to an object you would do the following:

script.py

```
class MyClass:
    variable = "blah"

    def function(self):
        print("This is a message inside the class.")

myobjectx = MyClass()
```

IPython Shell

```
In [1]:
```

Now the variable "myobjectx" holds an object of the class "MyClass" that contains the variable and the function defined within the class called "MyClass".

Accessing Object Variables

To access the variable inside of the newly created object "myobjectx" you would do the following:

script.py

```
class MyClass:
    variable = "blah"

    def function(self):
        print("This is a message inside the class.")

myobjectx = MyClass()

myobjectx.variable
```

IPython Shell

```
Out[1]: 'blah'

In [1]:
```

So for instance the below would output the string "blah":

script.py

```
class MyClass:
    variable = "blah"

    def function(self):
        print("This is a message inside the class.")

myobjectx = MyClass()

print(myobjectx.variable)
```

IPython Shell

```
blah

In [1]:
```

You can create multiple different objects that are of the same class(have the same variables and functions defined). However, each object contains independent copies of the variables defined in the class. For instance, if we were to define another object with the "MyClass" class and then change the string in the variable above:

script.py

```
class MyClass:
    variable = "blah"

    def function(self):
        print("This is a message inside the class.")

myobjectx = MyClass()
myobjecty = MyClass()

myobjecty.variable = "yackity"

# Then print out both values
print(myobjectx.variable)
print(myobjecty.variable)
```

IPython Shell

```
blah
yackity

In [1]:
```

Accessing Object Functions

To access a function inside of an object you use notation similar to accessing a variable:

script.py

```
class MyClass:
    variable = "blah"

    def function(self):
        print("This is a message inside the class.")

myobjectx = MyClass()

myobjectx.function()
```

IPython Shell

```
This is a message inside the class.

In [1]:
```

The above would print out the message, "This is a message inside the class."

Exercise

Using a Python Editing software, type in the code below. Once typed in, you will have a class defined for vehicles. **Create two new vehicles called car1 and car2.**

- **Set car1 to be a red convertible worth $60,000.00 with a name of Fer**
- **Set car2 to be a blue van names Jump worth $10,000.00**

Screen shot your completed code (not the output!) into a Word document with the title "Classes and Objects"

Python 3.4.1: Untitled

File Edit Format Run Options Windows Help

```
# define the Vehicle class
class Vehicle:
    name = ""
    kind = "car"
    color = ""
    value = 100.00
    def description(self):
        desc_str = "%s is a %s %s worth $%.2f." % (self.name, self.color, self.kind, self.value)
        return desc_str
# your code goes here

# test code
print(car1.description())
print(car2.description())
```

Your output should resemble the screen shot below:

Python 3.4.1 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:45:13) [
D64)] on win32
Type "copyright", "credits" or "license()" for more informa
>>> ========== RESTART ==========
>>>
Fer is a red convertible worth $60000.00.
Jump is a blue van worth $10000.00.
>>>
```

Language Investigation

Using the Internet complete the table below which is based on different programming languages. Do not just copy and paste text as you are not learning anything.

Programming Language	Type of Language	Common uses	Features	Compatibility	Example Syntax	Support Types	Advantages	Disadvantages
Visual Basic								
C++								
Python								
Java								