SHS LEARNING ACTIVITY

Research2-05-01

Name: ______________________ **Score/Mark:** __________

Grade and Section: ______________________ **Date:** __________

Strand: ☐ **STEM** ☐ **ABM** ☐ **HUMSS** ☐ **ICT (*TVL Track*)**

Type of Activity : ☐ Concept Notes ☐ Skills: Exercise/Drill ☐ Illustration ☐ Laboratory Report ☐ Essay/Task Report ☐ Other: __________

Activity Title: 05-01.What are computers? v05

Learning Target: To describe that computers are not magic

Authors/References: H. Shim, V. Sojo / Wikipedia: Computer; Abacus
Exploring Computer Science Teacher Version v7.0

What are **computers**? Computers have become such an integral part of our everyday lives that we may not even know how to answer this simple question. However, computers are not magic, they are simply **big calculators that do exactly what they are told**.

The word "computer" used to mean a **person** who carried out computations – however, towards the 20th century, it started meaning a **machine** that carries out computations. For thousands of years, human civilizations have invented various devices to aid computation, such as the abacus, which has been around since 2700 BCE. In the early 19th century the first computing device was developed that had the input of programs and data (through punched cards, stiff paper containing data), an arithmetic logic unit, flow control, integrated memory, visual output and sound output – like our modern general-purpose computer!

The first fully automatic digital computer (the Z3) was developed in 1941. This was very close to modern machines, using a binary system (expressed as series of 0 and 1) instead of a decimal system. However, this computer occupied an entire room, and it was less powerful than a cheap modern wristwatch (a simple addition took 0.8 seconds, and multiplication took over 3 seconds!). The concept of a modern computer that is capable of executing any computable algorithms (instructions) was proposed by Alan Turing in 1936. In 1951, the world's first commercial general-purpose computer (the Ferranti Mark 1) was available, and ran the world's first office computer job. Today we have computers everywhere, from laptops, to desktops, to pocket calculators, to smartphones and even planes!

Exercises:

1. How would you define computing or computer science? (1 sentence)
2. Give two or three examples of computers from your daily life.

Name: ________ Score/Mark: ________
Grade and Section: ________ Date: ________
Strand: ☐ STEM ☐ ABM ☐ HUMSS ☐ ICT (*TVL Track*)
Type of Activity : ☐ Concept Notes ☐ Skills: Exercise/Drill ☐ Illustration ☐ Laboratory Report ☐ Essay/Task Report ☐ Other: ________

Activity Title: 05-02.Programming languages v04

Learning Target: To define programming languages as communication tools

Authors/References: H. Shim, V. Sojo / Wikipedia: Programming language; A.B. Downey, *Think Python*. Version 2.0.17

Computers just do what they are told, but since they use binary code (series of 0s and 1s), we need a way to communicate with them. Programming languages are communication tools that allow us, the programmers, to give instructions for the computer to follow. The history of programming languages is bound to the history of computers–e.g. FORTRAN, invented in 1954 at IBM, was one of the first high-level programming languages.

Nowadays, there is a wide range of programming languages available, including C, C++, Java, Python, JavaScript and R. They are all different in syntax (form) and semantics (meaning). Learning a programming language takes a lot of time and effort (just like learning English, Spanish, Mandarin or Visayan!) and we should choose according to our purpose. In research, R and Python are among the most frequently used, Python due to its ease for data manipulation, and R due to its statistical power. We will focus on Python for now. Imagine you want to ask your computer to analyze your data, which has 1 million entries. You can do this very quickly in Python!

Let's start with some basics about computers and programming:

Machine code: Binary sequences that the computer can understand.
High-level language: A programming language (like Python) that is designed to be easy for humans to read and write, and then translated ("compiled" or "interpreted") into machine code for the computer to run.
Compile: To translate a program written in a high-level language into machine code, all at once, in preparation for later execution.
Interpret: Executing a program in a high-level language by reading and translating it one line at a time. Python is interpreted, not compiled.
Algorithm: A general process for solving a category of problems.
Program: A full set of instructions for the computer to perform.
Script: A program with relatively simple instructions, in a file, typically in an interpreted language (run by the computer line by line).
Bug: An error in a program.
Debugging: The process of finding and removing a programming error.
Syntax error: An error in a program that makes it impossible for the computer to interpret (it doesn't understand what we meant, so it stops).
Semantic error: An error in a program that makes the computer do something other than what we intended (we gave it the wrong command).

	SHS LEARNING ACTIVITY	**Research2-05-03**

Name: ______________________ **Score/Mark:** __________

Grade and Section: ______________________ **Date:** __________

Strand: ☐ **STEM** ☐ **ABM** ☐ **HUMSS** ☐ **ICT (*TVL Track*)**

Type of Activity : ☐ Concept Notes ☐ Skills: Exercise/Drill ☐ Illustration ☐ Laboratory Report ☐ Essay/Task Report ☐ Other: __________

Activity Title: 05-03."Hello World!". Our first script. v09

Learning Target: To start coding in Python

Authors|References: H. Shim, V. Sojo | docs.python.org/3/tutorial

Remember that we learned how computers just do what they are told? Let's prove that by writing our first Python **script**:

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

And that's it! **Run** the script. The computer will do exactly what we told it to do: it will print "Hello World!". Well done, we are now coders!

We also saw that computers are big calculators. Let's do some simple math:

```
# the following code should produce 11:
9 + 5 - 3
```

The hash character **#** is used to write **comments** for ourselves and our fellow coders. **The computer will just ignore anything after a #**.

If we run the last script we'll see that nothing seems to happen. What's going on? The computer actually did calculate the result (11), but we didn't tell it to do anything with this result, so it just ran the script and **exited** (finished) without giving us any feedback (because we didn't ask it to). Remember: computers do just what programmers tell them to do.

Mathematical and logical operators:

Addition	`a + b`	Equality (is a equal to b?)	`a == b`
Subtraction	`a - b`	Inequality (≠)	`a != b`
Multiplication (a×b)	`a * b`	Greater than	`a > b`
Division (real num.)	`a / b`	Greater than or equal to (≥)	`a >= b`
Division (integer)	`a // b`	Not (a is not true)	`not a`
Int division remainder	`a % b`	And (are both a and b true?)	`a and b`
Powers (a^b)	`a ** b`	Or (is either a or b true?)	`a or b`

Note: in Python 2, "/" gives integer division. We are using Python 3 here.

EXERCISE: Get the computer to show the results of the computation above and the following ones, but **predict the result** before you run the script!

`6*4` , `6/4` , `6//4` , `6%4` , `(50-5*6)/4`

Optional: try `print(2+3)` versus `print("2"+"3")` versus `print("2+3")`

SHS LEARNING ACTIVITY

Research2-05-04

Name: ____________________ **Score/Mark:** __________

Grade and Section: ____________________ **Date:** __________

Strand: ☐ **STEM** ☐ **ABM** ☐ **HUMSS** ☐ **ICT (*TVL Track*)**

Type of Activity : ☐ Concept Notes ☐ Skills: Exercise/Drill ☐ Illustration ☐ Laboratory Report ☐ Essay/Task Report ☐ Other: __________

Activity Title: 05-04.Variables v07

Learning Target: To use variables in python

Authors|References: H. Shim, V. Sojo | docs.python.org/3/tutorial

In mathematics, variables are symbols that represent values. Similarly, we can use the equals sign "=" to **assign** a **value** to a **variable** in Python.

Exercise. Predict the output of the following script, then write and run it:

```
width = 10
length = 2
height = 5
volume = width * length * height
```

You can probably guess that, like before, the computer won't show us any output (because we didn't tell it to). Add a line to tell it to print the volume.

We can also **reassign** variables. That is, we can change their value:

```
x = 4
print(x) # should give 4
x = 5
print(x) # should give 5. Now try x = x*2, x = x*x, x = x
...and x += 1
```

Note: the . . . mean that something should be written on the previous line!

As we saw before, Python can also manipulate text, known in computing as character **strings**. Try this:

```
s = "First line.\nSecond line." # "\n" means newline
print(s)
```

We can even do some funny things, like multiplying strings:

```
s = "sigi,"
print(3*s + "I like" + 2*" halo")
```

Text is called a "character string" because that's what it is: a string of characters. This means we can actually print specific letters, using brackets:

```
s = "Guindulman"
print(s[0]) # first letter of the string "Guindulman"
print(s[-3]) # third letter from the end
print(s[1:3]) # a "slice" or piece of a string
```

Note that, like most programming languages, Python starts counting at zero.

	SHS LEARNING ACTIVITY	Research2-05-05

Name: ______________________ **Score/Mark:** __________
Grade and Section: ______________________ **Date:** __________
Strand: ☐ **STEM** ☐ **ABM** ☐ **HUMSS** ☐ **ICT (*TVL Track*)**
Type of Activity : ☐Concept Notes ☐Skills: Exercise/Drill ☐ Illustration ☐Laboratory Report ☐Essay/Task Report ☐Other: __________
Activity Title: 05-05.Data types and data structures v08
Learning Target: To define data types and use data structures in python
Authors|References: H. Shim, V. Sojo | docs.python.org/3/tutorial

Previously, we learnt about the growing importance of computational analysis, given the ever-increasing amounts of data in most fields. Python is known for its ease for data processing, and a crucial step for this is storing the data. First, we need to remember that there are different **types of data**, such as **integer** (**1**, 27, -4, 395), **float** (real numbers with decimals, like **1.0**, 2.17908) and **strings** (like "some text", **"1"**). Now, we can start organizing our data into **data structures**, such as **lists** and **dictionaries**.

Lists (also called arrays) are ordered successions of values. The position of an item in a list is called its **index**, and it **always starts at zero**.

```
squares = [1, 4, 9, 16, 25, 36, 49]
print(squares) # prints everything in the list
print(squares[0]) # the FIRST item, starting at zero!
print(squares[3]) # the FOURTH item, 16
print(squares[-2]) # second item from the end, 36
```

Lists can be **"sliced"** (or cut into fragments), just like we did with strings:

```
print(squares[2:4])
```

Note that this actually leaves out the last item.

```
print(squares[-4:]) # prints until the end
```

Lists can also be extended or **concatenated** (joined with other lists):

```
squares + [64, 81] # let's extend the list
print(squares) # it didn't change! Why?
```

That didn't really work! We forgot that we need to **reassign** the variable:

```
squares = squares + [64, 81, 10] # try printing again
```

You can also change the contents of lists. Let's correct the mistake at the end of the previous command:

```
squares[-1] = 100 # always print to check if it worked
```

Dictionaries are similar to lists, but they are unordered. Instead, they work with **key-value pairs**, where the value can be accessed with the key:

```
population = {'Jagna': 33892, 'Manila': 12877253}
print(population)
print(population['Jagna'])
population['Berlin'] = 6004857 # this adds a new pair
```

SHS LEARNING ACTIVITY	Research2-05-06

Name: ______________________ **Score/Mark:** ________

Grade and Section: ______________________ **Date:** ________

Strand: ☐ **STEM** ☐ **ABM** ☐ **HUMSS** ☐ **ICT (*TVL Track*)**

Type of Activity : ☐ Concept Notes ☐ Skills: Exercise/Drill ☐ Illustration ☐ Laboratory Report ☐ Essay/Task Report ☐ Other: ________

Activity Title: 05-06.Control statements: if–elif–else v08

Learning Target: To start programming by using control flows in Python

Authors | References: H. Shim, V. Sojo | docs.python.org/3/tutorial

Finally, we can take our first steps in actual programming. The beauty and power of programming is that we can use **control statements** to decide whether or not to execute some tasks, and to repeat other tasks in a sequential manner. We will focus on two types of control statements:

Control type	Keywords
Decision making	if, elif, else
Looping/iteration/repetition	while
Looping/iteration/repetition	for

The simplest control statement is the "if" decision block:

```
x = 1 # try changing this to 1+1, 2, -2, and run again
if x==2: # note the double equals for comparisons!
⇥ print("x equals two")
⇥ print("We're still inside the if")
print("We're outside the if")
```

IMPORTANT: Python requires that we use proper **indentation**. This means that every statement inside the "if" block has to start with **the same spacing** before it. Let's agree to use **1 TAB** always (TAB is the big key with long arrows, at the top-left of most keyboards, symbolized by ⇥ above).

But what if we want to do something when the condition is **not true**?

```
x = 5 # change to -5, 5-5, 5*0, and others
if x>0:
⇥ print("x is positive")
else:
⇥ print("I don't know whether x is zero or negative!")
```

EXERCISE: There is also an "elif" statement, which is short for else-if. We use these to test additional conditions between the opening if and the closing else. Modify the script adding an elif between the if and the else to test whether x < 0, so that we can find out if any given x is positive, negative, or zero. Test with various values of x as above.

SHS LEARNING ACTIVITY	Research2-05-07

Name: ____________________ **Score/Mark:** __________

Grade and Section: ____________________ **Date:** __________

Strand: ☐ **STEM** ☐ **ABM** ☐ **HUMSS** ☐ **ICT (*TVL Track*)**

Type of Activity : ☐Concept Notes ☐Skills: Exercise/Drill ☐ Illustration ☐Laboratory Report ☐Essay/Task Report ☐Other: __________

Activity Title: 05-07.Iterative (repetitive) statements ("loops") v06

Learning Target: To use while and for loops to repeat tasks in Python

Authors|References: H. Shim, V. Sojo | docs.python.org/3/tutorial

One of the greatest things about computers is that they can repeat tasks millions of times without complaining. We use two main **loops** to repeat tasks: **for** and **while**.

for loops execute a specific number of times, defined within a "range":

```
for i in range(0, 5):
⇥ print("We are printing this line over and over again!")
```

Printing the same thing over and over is a bit boring. It would be more interesting to use the number **i** itself to do some calculations:

```
for i in range(0, 5):
⇥ print("The square of "+ str(i) + " is " + str(i*i))
```

Note that we had to tell Python to convert "i" to a string by using the inbuilt **function** `str()`. Otherwise, `print()` gets confused because we are giving it different **data types**. We will learn about functions soon.

for loops are very powerful. They can iterate (loop) over a list too:

```
squares = [1, 4, 9, 16, 25]
for square in squares:
⇥ print(square)
```

while loops execute as long as a given condition remains true. We can use operators such as >, <=, >=, ==, !=, or variables with a value of False or True (also, 0 counts as "False", and any other number is "True").

```
a = 0
while a < 10:
⇥ print(a)
⇥ a += 1 # it's the same as a=a+1, just a bit quicker
```

Because while loops run while a condition is true, they can be dangerous: what if we give it a condition that is always true, such as **while 1<2:**? We would send our obedient computer into an **infinite loop**! Let's be careful!

SHS LEARNING ACTIVITY

Research2-05-08

Name: ____________________ **Score/Mark:** __________

Grade and Section: ____________________ **Date:** __________

Strand: ☐ **STEM** ☐ **ABM** ☐ **HUMSS** ☐ **ICT (*TVL Track*)**

Type of Activity : ☐Concept Notes ☐Skills: Exercise/Drill ☐ Illustration ☐Laboratory Report ☐Essay/Task Report ☐Other: __________

Activity Title: 05-08.Functions v05

Learning Target: To produce reusable code by creating functions

Authors|References: H. Shim, V. Sojo | docs.python.org/3/tutorial

In programming, a **function** is a **reusable piece of code** that does a particular job. A function has a specified name by which it can be **called**. For example, we've already seen the `print(x)` function. Here, "x" is the **argument** of the function, i.e. the input that we give it. "print" is built into Python, but the great thing is that we can **define** our own functions (using "def"), and we can also have them **return** a result if we want to:

```
def function_name(arguments):
⇥ [operations and calculations in the function]
⇥ return [expression or value] # optional
```

Note: we don't have to return anything, and arguments are also optional.

Let's write a function to get the largest of two numbers:

```
def max_of_two_nums(num1, num2):
⇥ maximum = num1 # this may not be true, we will test it
⇥ if num2 > num1: # this is the test
⇥ ⇥ maximum = num2 # note the double TAB!
⇥ return maximum
```

Now we can run this function, e.g.: `print(max_of_two_nums(9, -17))`

Exercise: Create a function `max_of_list(the_list)` that returns the maximum from a list of numbers, such as `nums = [1, 45, 3.9, -7, 8]`. This is a little trickier, because we have to go over the whole list. Complete the code below:

```
def max_of_list(the_list):
⇥ maximum = the_list[0] # we will test if this isn't true
⇥ for num in the_list: # go over each number in the list
⇥ ⇥ # TO DO: test if this num is larger than maximum
⇥ ⇥ ⇥ # TO DO: change maximum to the current num
⇥ # TO DO: we need to return the maximum
```

Write all the code lines that start with "…TO DO:", and test the function.

SHS LEARNING ACTIVITY	Research2-05-09

Name: ____________________ **Score/Mark:** __________
Grade and Section: ____________________ **Date:** __________
Strand: ☐ **STEM** ☐ **ABM** ☐ **HUMSS** ☐ **ICT (*TVL Track*)**
Type of Activity : ☐Concept Notes ☐Skills: Exercise/Drill ☐ Illustration ☐Laboratory Report ☐Essay/Task Report ☐Other: __________
Activity Title: 05-09.Input v09
Learning Target: To read and use input from the user and from files
Authors | References: H. Shim, V. Sojo | docs.python.org/3/tutorial

So far, we've been defining all the data directly inside the script. However, we can also get data from the user of our programs:

```
name = input("Please enter your name: ")
age  = input("Now please enter your age: ")
print("Thank you, " + name + ". I hope you enjoyed ...
...your " + age + "th birthday!")
#print("Next year you'll be " + (age+1) + "years old.")
```

The last line is **commented out** because it has an error. Can you spot it?

More often, we want to read a large amount of data for analysis. Let's just load a small file (data.csv) that should be in the computers at school:

```
with open("data.csv") as datafile: # open the file
⇥ for line in datafile: # and read it line by line
⇥ ⇥ print(line)
```

Open the file in Notepad. Afterwards, open it in Excel. It's actually a table of values! We can analyze it by splitting it at the commas:

```
with open('data.csv', 'r') as datafile: # open the file
⇥ for line in datafile: # and read it line by line
⇥ ⇥ fields = line.split(',') # split it at the commas
⇥ ⇥ province = fields[0]
⇥ ⇥ capital = fields[1]
```

Now go on and play! Assign fields to variables. Try printing each province with its capital, such as, "`The capital of Bohol is Tagbilaran`".

The 'r' tells Python to open the file in **reading** mode (to protect it so that we don't destroy the data). This is because we can also **write** ('w') to files:

```
out_text = "some text\nto send out\nto a file\n"
with open('output.txt', 'w') as out_file:
  out_file.write(out_text) # now open the file to check
```

There is also an 'a' mode for **appending** to a file. **Exercise:** try it out!

SHS LEARNING ACTIVITY

Research2-05-010

Name: ______________________ **Score/Mark:** __________
Grade and Section: ______________________ **Date:** __________
Strand: ☐ **STEM** ☐ **ABM** ☐ **HUMSS** ☐ **ICT (*TVL Track*)**
Type of Activity : ☐ Concept Notes ☐ Skills: Exercise/Drill ☐ Illustration
☐ Laboratory Report ☐ Essay/Task Report ☐ Other: __________

Activity Title: What are computers made of? v 03
Learning Target: To describe that the computer is a machine
Authors/References: H. Shim, V. Sojo
Exploring Computer Science Teacher Version v7.0
https://en.wikipedia.org/wiki/Computer

Have you ever wondered what computers are made of? Have you ever seen the inside of a computer? There are two aspects to functional computers: hardware and software. Hardware refers to all parts of a computer that are physical objects: circuits, graphic/sound cards, computer chips, memory (RAM), displays, keyboards, speakers, printers, mouse, power supplies, etc. Software refers to all parts of the computer that are not physical: programs, data, libraries, protocols, etc. Knowing all these parts by heart is not essential for research – even for computational research. However, it is important to know how computers function since their performance is related to your performance in research!

Hardware

- Central Processing Unit (CPU) – It manages the various components of the computer. It reads and interprets instructions, and transforming them into a set of signals that activate other systems of the computer.
- Memory (RAM) – In modern computers, each memory cell stores binary numbers in groups of 8 bits called a “byte”. Thus, each byte can represent 2^8=256 different numbers. Random-Access Memory (RAM) can be read and written whenever the CPU commands it.
- Input/Output (I/O) – The means through which computers exchange information with the user. For input, you may use a keyboard, mouse, etc., and for output you may use a display, printer, etc.

Software

- Programs – A computer program is a sequence of instructions for computers to execute in order to produce specific outcomes. Examples include word processors and web browsers.
- Programming languages – They are communication tools between the user and computers so a program can be created, implemented and executed. Examples include C, C++, Java, Python and R.

	SHS LEARNING ACTIVITY	Research2-05-011

Name: ________________________ **Score/Mark:** __________
Grade and Section: ____________________ **Date:** __________
Strand: ☐ **STEM** ☐ **ABM** ☐ **HUMSS** ☐ **ICT (*TVL Track*)**
Type of Activity : ☐Concept Notes ☐Skills: Exercise/Drill ☐ Illustration
☐Laboratory Report ☐Essay/Task Report ☐Other: __________
Activity Title: 05-03. Computer Buying Project v 02
Learning Target: To ask practical questions on the specifications of a computer
Authors/References: H. Shim, V. Sojo
Exploring Computer Science Teacher Version v7.0
https://en.wikipedia.org/wiki/Computer

The CVIF wants to buy a dozen computers, so that the students can learn programming languages such as Python and R. You are in charge of choosing computers – you may ask questions such as “What will be the use of the computer? What is the budget? What are the specifications you need for the CVIF students to learn programming?” You can look online or interview the CVIF staff or teacher.

EXERCISES:

1. What is the use of the computer? (1 sentence max)
2. What is the limitation in price range? (1 sentence max)
3. What are the specifications you need to learn basic programming? (1 sentence max)
4. Complete the computer comparison chart.

	Computer 1	**Computer 2**	**Computer 3**	**Computer 4**
Laptop or Desktop				
Operating System				
Processor (CPU)				
Memory (RAM)				
Hard Drive (Storage)				
Screen size				
Cost				
Others				

5. Choose a computer and justify your choice using the information you gathered. (1 sentence max)