**Level 5: AI Game Programming**

**Week 3, Lab 3**

*Record all that you do in your lab logbook (hardcopy or Word file). What you do in the lab can be used in the assignment.*

In the assignment, it is rather impossible not to use Python. Therefore, it is crucial to familiarise yourself with the programming language and the programming environment, Google Colab. As you used Python and Google Colab in MCG unit last year already. Please treat the first part of this lab as a revision.

**Python an Google Colab Revision**

This week we will practice further to equip yourselves with the necessary knowledge before we fully program different artificial intelligence methods. This lab is designed to help you.

**1. Brief Python tutorial (skip this part if you already know Python well)**

(If you already know Python well, please go to Section 2.)

In this tutorial, we will cover the main aspects of Python language, including clear explanations, easy examples, exercises and solutions, and comparisons with C++. We will also show you how to use numpy and matplotlib libraries in Google Colab.

NB: More Python references can be found on:  
<https://www.python.org/about/gettingstarted/>   
<https://www.w3schools.com/python/>

**Important: unlike C++, Python uses *'indention'* to define blocks of code.**

**Variables and Data Types**

In Python, variables can be assigned values of different data types, such as integers, floats, strings, and booleans. Python is dynamically typed - no need to declare variables with a type. Here are some examples:

x = 5 # integer  
y = 3.14 # float  
z = "Hello, World!" # string  
a = True # boolean

In C++, variables are also assigned values of different data types, but the syntax is different:

int x = 5; // integer  
double y = 3.14; // float  
string z = "Hello, World!"; // string  
bool a = true; // boolean

**Arrays:** in Python, arrays are defined using square brackets []. Here are some examples:

# create a list with the values 1, 2, 3, 4, 5

my\_list = [1, 2, 3, 4, 5]

# print the first element of the list (which has an index of 0)

print(my\_list[0]) # prints 1

# append the value 6 to the end of the list

my\_list.append(6)

# remove the value 3 from the list

my\_list.remove(3)

In C++, arrays are defined using square brackets [], but the syntax is different:

int my\_array[] = {1, 2, 3, 4, 5};

cout << my\_array[0] << endl; // prints 1

**List**: Python has a built-in data structure called a list, which is similar to an array in C++. However, lists in Python can contain elements of different data types. Here's an example of a list in Python:

# create a list of fruits

fruits = ["apple", "banana", "orange"]

# print the first element of the list (which has an index of 0)

print(fruits[0]) # Output: apple

# print the length of the list

print(len(fruits)) # Output: 3

# append the string "grape" to the end of the list

fruits.append("grape")

# print the updated list

print(fruits) # Output: ["apple", "banana", "orange", "grape"]

C++: you can use the std::vector class to create a dynamic array that can store elements of different data types. Here's the equivalent C++ code:

// Vector

std::vector<std::string> fruits = {"apple", "banana", "orange"};

std::cout << fruits[0] << std::endl; // Output:

**Loops**

Python provides two types of loops - for and while. The for loop in Python is similar to C++’s range-based for loop. Remember, Python uses indention to define blocks of code. Here are some examples:

**Example:**

# loop through the range of numbers from 0 to 4 (inclusive)

for i in range(5):

print(i)

# create a while loop that will continue to execute as long as x is less than 10

while x < 10:

print(x)

x += 1

In C++, the syntax for loops is different:

for (int i = 0; i < 5; i++) {

cout << i << endl;

}

while (x < 10) {

cout << x << endl;

x++;

}

**Example:**

Python

for i in range(5):

print(i, end=' ')

C++

for (int i = 0; i < 5; i++) {

cout << i << " ";

}

**Conditional Statements**

Python provides if-else statements to execute code blocks based on certain conditions. Here are some examples:

# check if x is greater than y

if x > y:

print("x is greater than y")

# if x is not greater than y, check if x is equal to y

elif x == y:

print("x is equal to y")

# if x is not greater than y and not equal to y, then x must be less than y

else:

print("x is less than y")

In C++, the syntax for conditional statements is different:

if (x > y) {

cout << "x is greater than y" << endl;

} else if (x == y) {

cout << "x is equal to y" << endl;

} else {

cout << "x is less than y" << endl;

}

**Operators**

Python provides arithmetic operators such as `+`, `-`, `\*`, `/`, `%`, and `\*\*` for performing mathematical operations. Here are some examples:

Python

# Arithmetic operators

a = 5

b = 2

print(a + b) # Addition

print(a - b) # Subtraction

print(a \* b) # Multiplication

print(a / b) # Division

print(a % b) # Modulo

print(a \*\* b) # Exponentiation

# Comparison operators

print(a == b) # Equal to

print(a != b) # Not equal to

print(a > b) # Greater than

print(a < b) # Less than

print(a >= b) # Greater than or equal to

print(a <= b) # Less than or equal to

C++

// Arithmetic operators

int a = 5;

int b = 2;

std::cout << a + b << std::endl; // Addition

std::cout << a - b << std::endl; // Subtraction

std::cout << a \* b << std::endl; // Multiplication

std::cout << a / b << std::endl; // Division

std::cout << a % b << std::endl; // Modulo

std::cout << pow(a, b) << std::endl; // Exponentiation

// Comparison operators

std::cout << (a == b) << std::endl; // Equal to

std::cout << (a != b) << std::endl; // Not equal to

std::cout << (a > b) << std::endl; // Greater than

std::cout << (a < b) << std::endl; // Less than

std::cout << (a >= b) << std::endl; // Greater than or equal to

std::cout << (a <= b) << std::endl; // Less than or equal to

**Comparison Operators**

Python provides comparison operators such as `==`, `!=`, `>`, `<`, `>=`, and `<=` for comparing values. Here are some examples:

equal = x == y

not\_equal = x != y

greater\_than = x > y

less\_than = x < y

greater\_than\_or\_equal\_to = x >= y

less\_than\_or\_equal\_to = x <= y

In C++, the syntax for comparison operators is the same:

bool equal = x == y;

bool not\_equal = x != y;

bool greater\_than = x > y;

bool less\_than = x < y;

bool greater\_than\_or\_equal\_to = x >= y;

bool less\_than\_or\_equal\_to = x <= y;

**Logical Operators**

Python provides logical operators such as `and`, `or`, and `not` for combining conditions. Here are some examples:

and\_operator = x > 0 and y < 10

or\_operator = x < 0 or y > 10

not\_operator = not a

In C++, the syntax for logical operators is the same:

bool and\_operator = x > 0 && y < 10;

bool or\_operator = x < 0 || y > 10;

bool not\_operator = !a;

**Functions**

Python allows you to define your own functions using the `def` keyword. Return values with return statement. Here are some examples:

# define a function called my\_function that takes two parameters, x and y

def my\_function(x, y):

# return the sum of x and y

return x + y

# call the my\_function with arguments 3 and 4 and store the result in a variable called result

result = my\_function(3, 4)

# print the value of result, which should be 7

print(result) # prints 7

In C++, functions are defined using the `function` keyword:

int my\_function(int x, int y) {

return x + y;

}

int result = my\_function(3, 4);

cout << result << endl; // prints 7

**Classes and Objects**

A class keyword is defines a class, self refers to the current instance. An inheritance uses parentheses (object). Here are some examples:

# define a class called Vehicle

class Vehicle:

# define a constructor that takes two parameters, make and model

def \_\_init\_\_(self, make, model):

# set the make and model attributes of the object

self.make = make

self.model = model

# define a method called drive

def drive(self):

# print a message indicating that the vehicle is driving

print("Driving", self.model)

# define a subclass of Vehicle called Car

class Car(Vehicle):

# define a method called open\_trunk

def open\_trunk(self):

# print a message indicating that the trunk is opening

print("Opening trunk")

# create an instance of the Car class with make "Toyota" and model "Prius"

my\_car = Car("Toyota", "Prius")

# call the drive method of the my\_car object, which should print "Driving Prius"

my\_car.drive() # Prints "Driving Prius"

**Numpy and Matplotlib**

Numpy is a Python library for numerical computing, and Matplotlib is a Python library for data visualization. Here's how to use them in Google Colab:

Example

# install the numpy and matplotlib packages using pip

!pip install numpy matplotlib

# import the numpy and matplotlib.pyplot modules

import numpy as np

import matplotlib.pyplot as plt

# create an array of 100 evenly spaced values between 0 and 10 using numpy's linspace function

x = np.linspace(0, 10, 100)

# create an array of y values by taking the sine of each value in the x array

y = np.sin(x)

# plot the x and y values using matplotlib's plot function

plt.plot(x, y)

# display the plot using matplotlib's show function

plt.show()

This code installs the numpy and matplotlib libraries, imports them into the program, generates some data using numpy, and plots the data using matplotlib.

**Example**

# import the numpy module and alias it as np

import numpy as np

# create a numpy array with the values 1, 2, and 3

a = np.array([1, 2, 3])

# add 5 to each element of the array and print the result

print(a + 5) # Adds 5 to each element

# create a 3x4 numpy array filled with 1's

b = np.ones((3,4)) # 3x4 array of 1's

# print the shape of the array, which should be (3,4)

print(b.shape) # Prints (3,4)

**Example**

# import the matplotlib.pyplot module and alias it as plt

import matplotlib.pyplot as plt

# create two lists of x and y values

x = [1, 2, 3, 4]

y = [1, 4, 9, 16]

# plot the x and y values using matplotlib's plot function

plt.plot(x, y) # Plots the points

# display the plot using matplotlib's show function

plt.show() # Displays the plot

**Some common errors that students make when learning Python**

1. Syntax errors: These occur when the student writes code that does not follow the correct syntax rules of Python. This can include missing parentheses, commas, or quotation marks.

2. Indentation errors: Python uses indentation to indicate the start and end of code blocks. Students may forget to indent or indent too much, causing errors.

3. Name errors: These occur when the student tries to use a variable or function that has not been defined or is misspelled.

4. Type errors: These occur when the student tries to perform an operation on incompatible data types, such as trying to add a string and an integer.

5. Logic errors: These occur when the student writes code that does not produce the desired output, but does not result in an error message. This can be caused by incorrect conditional statements or loops.

**Some best practices for debugging Python code:**

1. Use print statements: Inserting print statements in your code can help you identify where the code is going wrong and what values are being assigned to variables.

2. Use a debugger: Python has built-in debugging tools that allow you to step through your code line by line and inspect variables at each step.

3. Check your assumptions: Make sure that your assumptions about the code are correct. Check that variables are being assigned the correct values and that functions are being called with the correct arguments.

4. Simplify the problem: If you are having trouble identifying the source of the error, try simplifying the problem by removing unnecessary code or breaking the code into smaller parts.

5. Use error messages: Python provides error messages that can help you identify the source of the error. Read the error message carefully and try to understand what it is telling you.

**Exercise**

**Exercise 1:** Create a Python program that assigns values to variables of different data types and prints them to the console.

**Exercise 2:** Create a Python program that uses a for loop to print the first 10 even numbers.

**Exercise 3:** Create a Python program that uses an if statement to check if a number is positive, negative, or zero.

**Exercise 4:** Create a Python program that calculates the area of a rectangle given its length and width.

**Exercise 5:** Create a Python program that uses a `while` loop to find the first Fibonacci number greater than 1000.

**Exercise 6:** Create a Python program that uses an `if` statement to check if a number is between 0 and 10, inclusive.

**Exercise 7:** Create a Python program that defines a function to calculate the factorial of a number.

**Exercise 8:** Create a Python program that uses numpy to generate a random array of 100 numbers between 0 and 1, and uses matplotlib to plot a histogram of the array.

**Exercise 9:** Create a Python program that uses a `for` loop to print the elements of an array.

**2. Implementing simple artificial neuron learning algorithm**

In this section, you will learn how to implement the algorithm using Python in Google Colab.

Here is the formal algorithm (training a simple neural model). The aim is to find to the values of in the algorithm so that when applying input , it will produce the correct corresponding .

**Step 1: Initialisation**  
Set initial weights , threshold and .  
**Step 2: Activation**  
Activate the neural by applying inputs and desired output . Calculate the actual output at iteration

where is the number of the perceptron inputs, and step is a step activation function.  
**Step 3: Weight training**  
Update the weights of the perceptron

where is the weight correction at iteration .  
The weight correction is computed by the delta rule:

**Step 4: Iteration**  
Increase iteration by one, go back to Step 2 and repeat the process until convergence.

The inputs and the corresponding desired output are defined as:

|  |  |  |
| --- | --- | --- |
| Input | | Output |
|  |  | *Yd* |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

For your convenience, here is Matlab code implementation of the algorithm.

Niteration = 20;

w1 = 0.3; w2 = -0.1;

Theta = 0.2; alpha = 0.1;

x1(1) = 0; x2(1) = 0; Yd(1) = 0;

x1(2) = 0; x2(2) = 1; Yd(2) = 0;

x1(3) = 1; x2(3) = 0; Yd(3) = 0;

x1(4) = 1; x2(4) = 1; Yd(4) = 1;

p = 1;

while p <= (Niteration-4)

for i = 1:4

p

X = x1(i)\*w1 + x2(i)\*w2

if X >= Theta

Y = 1

else

Y = 0

end

e = Yd(i)-Y

w1 = w1 + alpha\*x1(i)\*e

w2 = w2 + alpha\*x2(i)\*e

p = p+1;

end

end

Or in C++ (please double check the code)

int Niteration = 20;

double w1 = 0.3, w2 = -0.1;

double Theta = 0.2, alpha = 0.1;

double x1[5] = {0, 0, 1, 1}, x2[5] = {0, 1, 0, 1}, Yd[5] = {0, 0, 0, 1};

int p = 1;

while (p <= (Niteration-4)) {

    for (int i = 0; i < 4; i++) {

        double X = x1[i]\*w1 + x2[i]\*w2;

        int Y = (X >= Theta) ? 1 : 0;

        double e = Yd[i]-Y;

        w1 = w1 + alpha\*x1[i]\*e;

        w2 = w2 + alpha\*x2[i]\*e;

        p++;

    }

}

**Task**

Implement the algorithm in Python. For your convenience, your the Matlab code or C++ code as your base code.

If you run the code in Matlab, at the end the code, it should produce ?

What about your Python implementation?

How would you know that your code is correct? How can you test it?

Hint: check each stept carefully to ensure that you get correct results at each step.

Think carefully and be aware of the step function!

Hint: correct results should be .

**Solutions to Section 1**

**Exercise 1:** Create a Python program that assigns values to variables of different data types and prints them to the console.

**Solution 1:**

x = 5  
y = 3.14  
z = "Hello, World!"  
a = True  
print(x)  
print(y)  
print(z)  
print(a)

**Exercise 2:** Create a Python program that uses a for loop to print the first 10 even numbers.

**Solution 2:**

for i in range(2, 21, 2):

print(i)

**Exercise 3:** Create a Python program that uses an if statement to check if a number is positive, negative, or zero.

**Solution 3:**

x = 5

if x > 0:

print("x is positive")

elif x == 0:

print("x is zero")

else:

print("x is negative")

**Exercise 4:** Create a Python program that calculates the area of a rectangle given its length and width.

**Solution 4:**

length = 5

width = 3

area = length \* width

print(area)

**Exercise 5:** Create a Python program that uses a `while` loop to find the first Fibonacci number greater than 1000.

**Solution 5:**

a = 0

b = 1

while b <= 1000:

c = a + b

a = b

b = c

print(b)

**Exercise 6:** Create a Python program that uses an `if` statement to check if a number is between 0 and 10, inclusive.

**Solution 6:**

x = 5

if x >= 0 and x <= 10:

print("x is between 0 and 10")

else:

print("x is not between 0 and 10")

**Exercise 7:** Create a Python program that defines a function to calculate the factorial of a number.

**Solution 7:**

def factorial(n):

if n == 0:

return 1

else:

return n \* factorial(n-1)

result = factorial(5)

print(result)

**Exercise 8:** Create a Python program that uses numpy to generate a random array of 100 numbers between 0 and 1, and uses matplotlib to plot a histogram of the array.

**Solution 8:**

import numpy as np

import matplotlib.pyplot as plt

data = np.random.rand(100)

plt.hist(data)

plt.show()

**Exercise 9:** Create a Python program that uses a `for` loop to print the elements of an array.

**Solution 9:**

my\_list = [1, 2, 3, 4, 5]

for i in my\_list:

print(i)