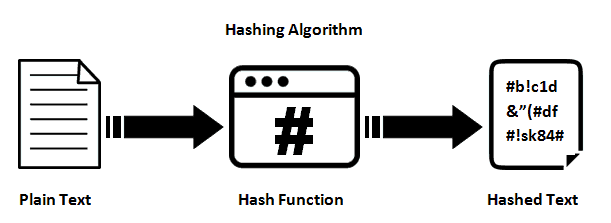
Hashes

Hashes is use to compare two files to know whether are they the same.

It is the way to convert the files, objects or words into a unique string that can be used to compare.

Hashes are calculated using hash algorithm such as the figure below:



You can find many online tools that can be used to calculate the hash value of your files or words. We will be using an online tool to convert text into hash value. [Online tool](https://www.tools4noobs.com/online_tools/hash/)

There are many algorithms that can be used to find the hash value. Today we will be looking at one of them which is **MD5.**

Let's look at storing your password in its hash value. Do you think other people can find out your password from its hash value? The answer is yes by using the most common method which is called Brute Force Attack!

Brute force is a method to guess or find out your password using:

* Trial and error
* Can be done by script or bot

In this GitHub repository I will be documenting my learning process.

# Basic Output

We can display any text on screen using the syntax print("<your\_text\_here>")

Let us try to print our name on screen.

print("HelloWorld")

->HelloWorld

Variables and Data types

* Variables are containers for storing data values. These variables can be associated with different data types such as
* Numeric: int, float, complex
* Boolean: bool

... and many more.

We can store any value or text in a variable. The syntax is <name\_of\_your\_variable> = <value/text>

Let us try to store our name in a variable and print it on screen.

name = "Sean"

print (name)

->Sean

I tried with other data types such as numbers and True/False.

True

False

# name the value and write number

print ("Value:" + true)

print ("Data Type:" + str(type(True)))

#set numbers to int 123

numbers = 123

#print number and set numbers from integer to string

print ("Number: " + str(numbers))

print (type(numbers))

#set boolean = to true

bool\_var = True

#print boolean variable

print (bool\_var)

#print type of syntax in this case Boolean Variable

print (type(bool\_var))

->Value:1

Data Type:<class 'bool'>

Number: 123

<class 'int'>

True

<class 'bool'>

We can store any value or text in a variable. The syntax is <name\_of\_your\_variable> = <value/text>

Let us try to store our name in a variable and print it on screen.

name = "Sean"

print (name)

->Sean

I tried with other data types such as numbers and True/False.

True

False

# name the value and write number

print ("Value:" + true)

print ("Data Type:" + str(type(True)))

#set numbers to int 123

numbers = 123

#print number and set numbers from integer to string

print ("Number: " + str(numbers))

print (type(numbers))

#set boolean = to true

bool\_var = True

#print boolean variable

print (bool\_var)

#print type of syntax in this case Boolean Variable

print (type(bool\_var))

->Value:1

Data Type:<class 'bool'>

Number: 123

<class 'int'>

True

<class 'bool'>

# Basic Input

Sometime, in our code, we want user to give some inputs and store them in our code for use later. Let us see how we can do that.

We are going to accept name as input and store in a variable. The syntax is <name\_of\_your\_variable> = input("<message\_requesting\_for\_input>"). Then, we will print it out on screen.

name = input("Enter your name please.")

print (name)

->Enter your name please. Sean

Sean

I tried to edit the code to accept two inputs instead, one for your first name and the another for your last name. Then, merge both strings together and display on screen.

name1 = input("First Name.")

name2 = input("Last Name.")

print (name1 + name2)

->First Name. sean Last Name. ng

Seanng

# Logical Conditions

Let us look at some mathematics operation in python and how we can make use of them.

Equal ==

Compare two data to see whether are they the same.

Not equal !=

Compare two data to see whether are they different.

Less than < / Less than or equal to <=

Compare two value to see whether the first value is less than (or equal to) to the second value.

Greater than > / Greater than or equal to >=

Compare two value to see whether the first value is greater than (or equal to) to the second value.

# Conditional Statements

We can use conditional statements to execute certain lines of code based on a decision. This can be used with the logical conditions above to form the condition.

Example, if my name is Kelvin, my surname is Yong. If I am not Kelvin, my surname is N/A.

name = input("Enter your name")

if name == "Kelvin":

  print(name + "Yong")

else:

  print(name + "N/A")

print("The End")

->Enter your nameKelvin

KelvinYong

The End

We can do the same with numbers. Example, my secret number is 123. If the user enter the right secret number, it will display "Access granted". Else, it will display "Access denied".

password = input("Please enter your password.")

if password == "123":

  print("Access Granted")

else:

  print("Access Denied")

->Please enter your password.123

Access Granted

Let us try to write a code that compares two numbers. If the first number is larger than the other one, it will display a message saying that the first number is larger than the other one. Do the same if the numbers are the equal or if the first number is smaller than the other one.

number1 = input("Enter a Number.")

number2 = input("Enter another number")

if number1 > number2:

  print(number1 + "is Bigger Than" + number2)

elif number1 < number2:

  print(number1 + "is Lesser Than" + number2)

else:

  print(number1 + "Equal To" + number2)

->Enter a Number.1 Enter another number 2

1 is Lesser Than 2

# Array

Python array is a list of data, unlike a variable which can only store one data.

Example, a list of fruits of apple, banana and orange.

fruits = ["apple","banana","orange"]

print(fruits[0])

->apple

Try create a list of numbers and print each numbers out.

number = [1,2,3]

print(number[0])

print(number[1])

print(number[2])

->1

2

3

# Loops

Loops allow us to execute lines of code multiple times until a certain condition is met. It saves us the trouble to copy and paste the same lines of code just to make it run several times.

There are two types of loops which we will look at today.

while loop check the condition for it to continue looping but it requires the relevant variables to be ready prior to this loop and we need to update the relevant variable manually. Let us look at the example below:

i = 0

while i < 10:

  print(i)

  i = i + 1

->0

1

2

3

4

5

6

7

8

9

for loop iterating over a sequence and stop once it reaches the end of the sequence. Let us look at the example below:

num = "0123456789"

for i in num:

  print (i)

0

1

2

3

4

5

6

7

8

9

Try use the for loop to print the list of fruits that we created previously. for i in fruits:

  print(i)

apple

banana

orange

# File Input

Another way to get input into our code is through external file. We can open the file, read it or even write into the file. Our focus will be just reading the file and save the content of the file as our input to be used later.

To read the content of the file, we need to open it first using the syntax:

<file\_object> = open("<file\_path>")

The open function will open the file specified at the location that you type under <file\_path> and store something called file object in any name that you give under <file\_object>.

Once we have the file object, we can use it to read the content of the file and store the content, line-by-line, in an array. The syntax is:

<array\_name> = <file\_object>.readlines()

Let us try file input in our code. First, we need a file as input. Open your notepad, type your first name in the first line and your last name in the second line. Save the file as **name.txt**.

Next, if you look at the left side of your browser, you will see a side bar. Click on the **folder** icon and drag and drop your **name.txt** into there. That way, your code will be able to find the file to open later. Once you are done with the preparation of the input file, we can begin to code.

 name = open("name.txt")

 name\_str = name.readlines()

 print(name\_str)

['Sean \n', 'Ng\n']

Challenge

Using what I learned here, I tried the followings:

* Using notepad, create a list of objects. The objects need to be listed horizontally. It is up to you what are the objects in the file. You can name the file any name you want but later in the code, remember to put the right name. Also, ensure that you drag and drop the file in the Google Collab folder, like that you did previously with name.txt.
* In your code, open and read the file. Store the file content in an array.
* Ask the user to type in a guess of the object in the file.
* Your code will check whether did the user type in a text that matches one of the objects in the file/array.
* Print out whether the guess is correct or wrong.

with open('name.txt') as file:

    contents = file.read()

    search\_word = input("Enter a Valorant Gun Name: ")

    if search\_word in contents:

        print ('Available')

    else:

        print ('Not Available')

Enter a Valorant Gun Name: ghost

Available

# Code our brute force program

First step of our brute force program is to code the hash value calculation part. Why is this important? Remember that one way to keep our passwords safe is to store them as hash values. Since we cannot convert the hash values back to the original form of passwords, what we can do is to guess the password first, convert the guessed password to its hash value and compare the hash value against the actual password's hash value. If it is a match, we found the password.

Here is how we code the algorithm to calculate the hash value of a text:

import hashlib

text\_str = input("Enter text: ")

text\_utf8 = text\_str.encode("utf-8")

hash\_value = hashlib.md5(text\_utf8).hexdigest()

print("Text: " + text\_str)

print("Hash Value: " + hash\_value)

->Enter text: hello

Text: hello

Hash Value: 5d41402abc4b2a76b9719d911017c592

With this, we built our own tool to calculate the hash value of a text or password using the MD5 algorithm.

Next, let us build a brute force attack tool to crack a password stored in its hash value. This is how our tool will work:

* First, it generates all the possible combinations for password.
* For each generated password, it calculates the hash value of the generated password.
* Then, it compares the hash value of the generated password against the hash value of the password that we want to crack.
* If it matches, we have found the password. If not, it will generate the next possible password and repeat the whole process.

Let us start small by guessing just one character password and the password consists only with numbers.

characters = "0123456789"

password = input("Enter hashed password: ")

for a in characters:

  text\_utf8 = a.encode("utf-8")

  hash\_value = hashlib.md5(text\_utf8).hexdigest()

if hash\_value == password:

  print("Password Found. The password is " + a)

Now, let us scale up our tools to crack two characters password. Notice that we have one for loop to go through all the characters for checking, one-by-one. For two characters passwords, we need two for loops arranged in a nested manner. Here, nested means one loop inside another. The first loop is to go through all possible first character of the passwords while the second loop is for the second character.

characters = "0123456789"

password = input("Enter hashed password: ")

for a in characters:

  for b in characters:

    guess\_pwd = a + b

    text\_utf8 = guess\_pwd.encode("utf-8")

    hash\_value = hashlib.md5(text\_utf8).hexdigest()

    if hash\_value == password:

      print("Password Found. The password is " + a)

This is all for now I will be learning to code more complex bots in the future!