Hacettepe University

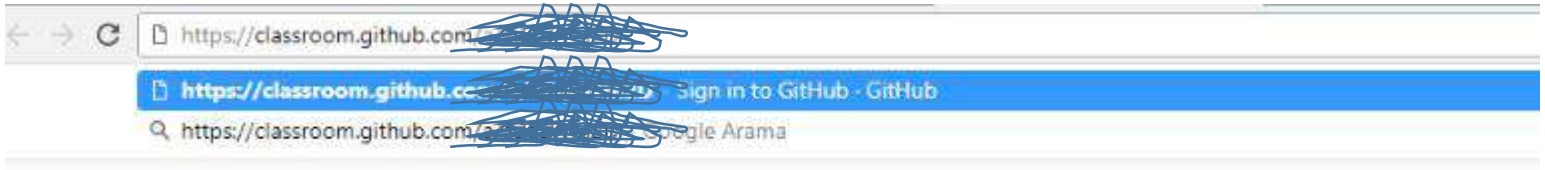
Computer Engineering Department

Using GitHub Classroom

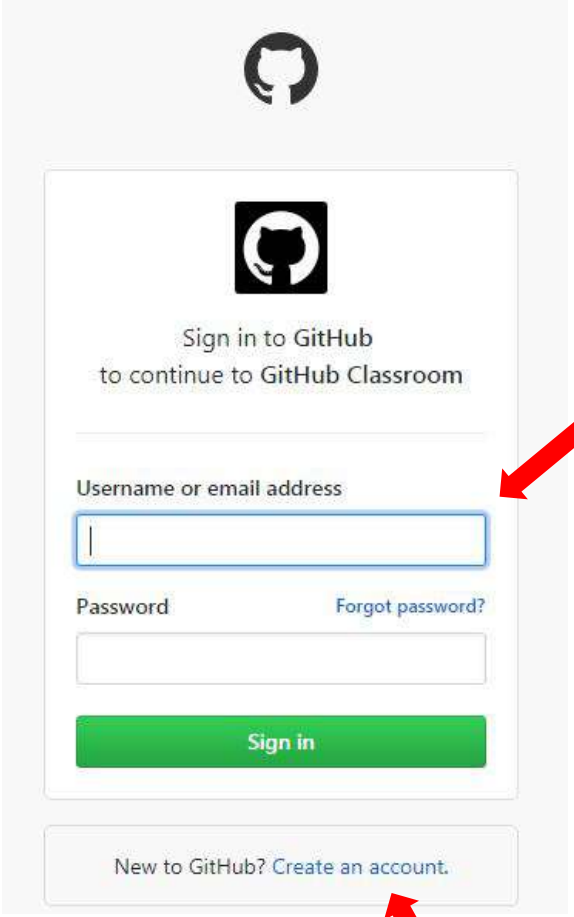
BBM103 Introduction to Programming Lab 1

Fall 2017

Signing Up to GitHub Classroom



Signing Up to GitHub Classroom

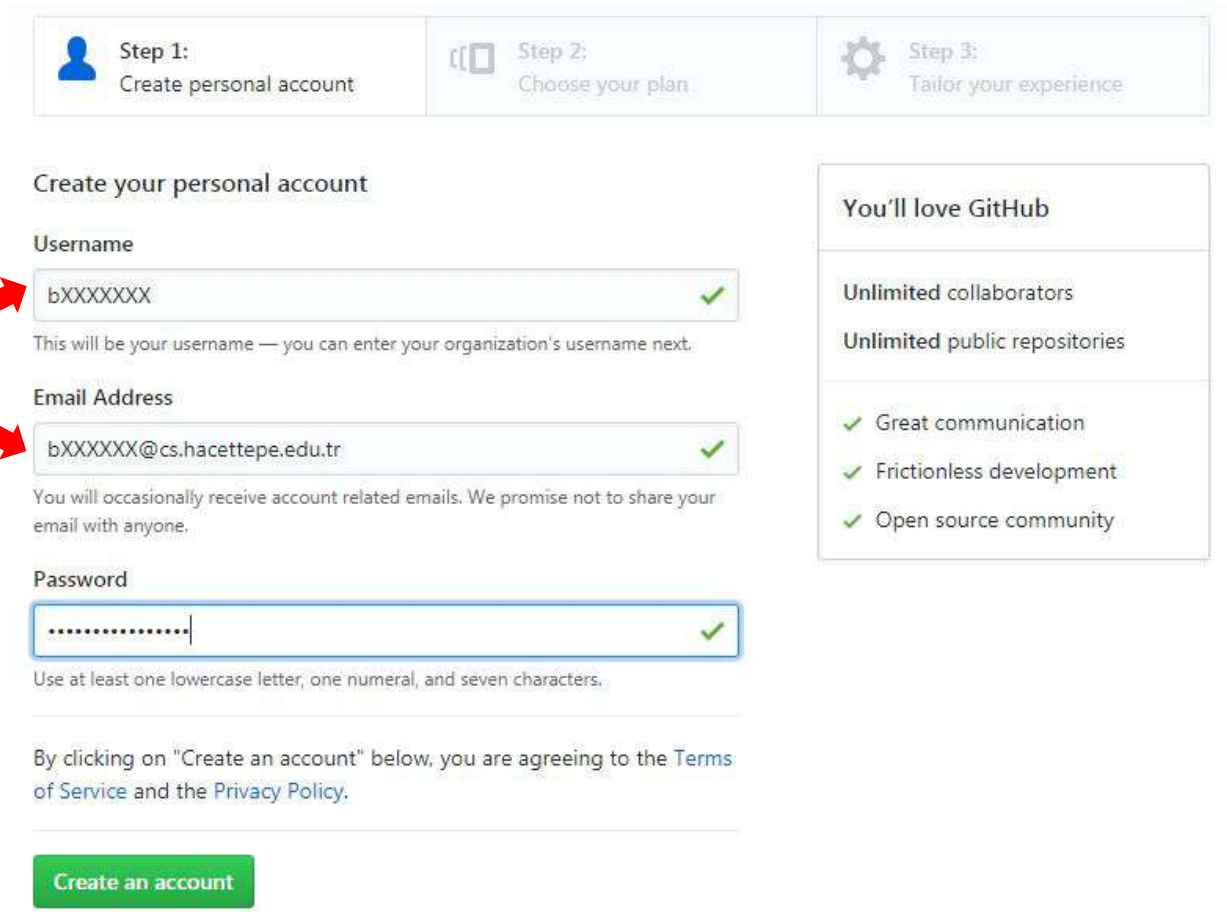


Fill textboxes and click sign in button to authorize

Click to sign up unless you have an educational account

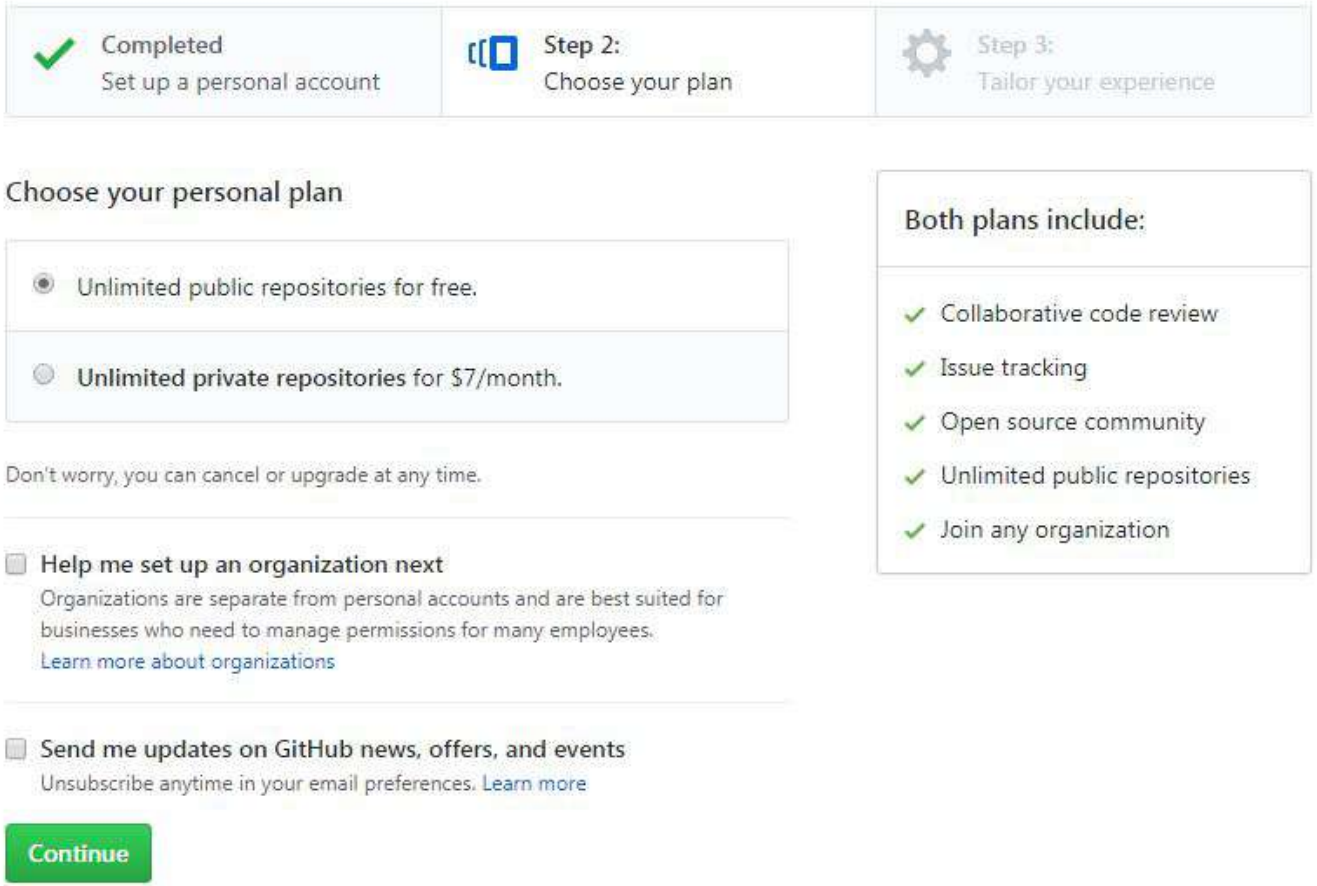
Signing Up to GitHub Classroom

You MUST create your account with your IDs beginning with ‘b’.



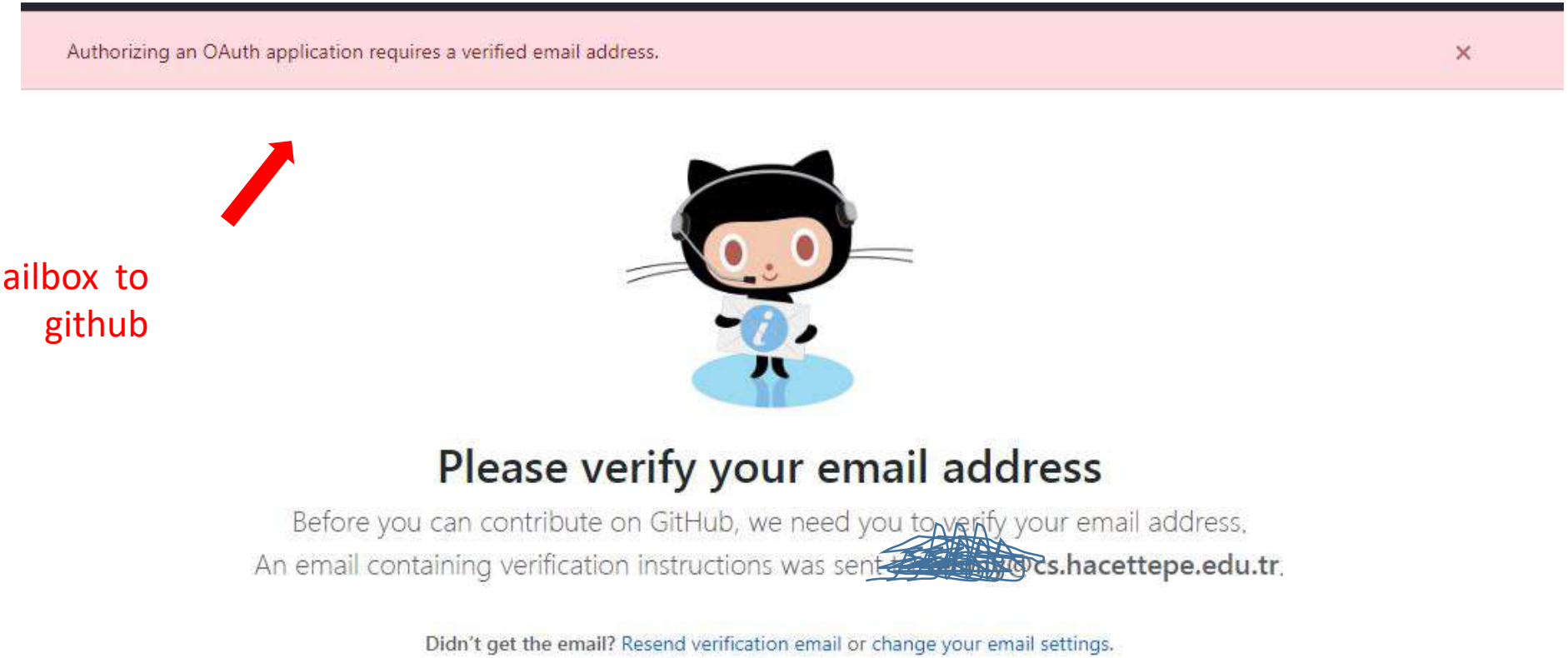
Signing Up to GitHub Classroom

There are two options. We recommend that you choose the 1st option unless you need a private repository.

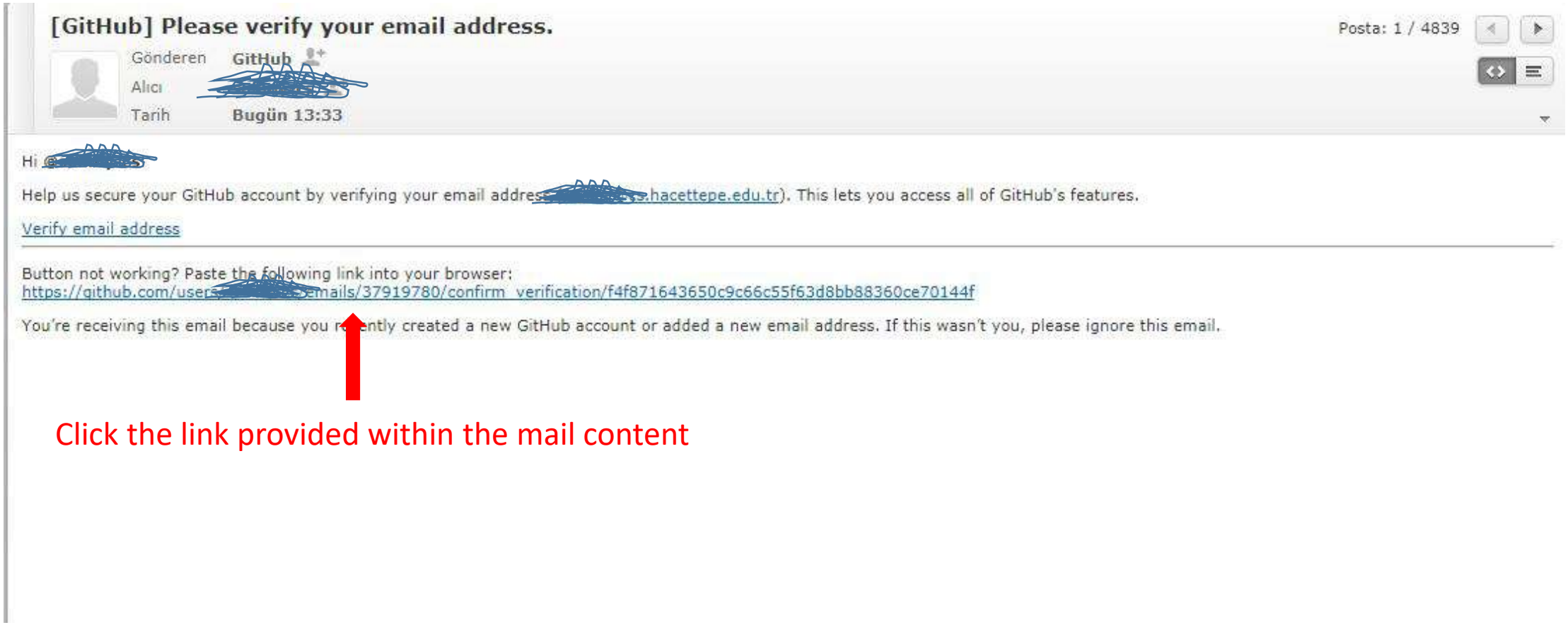


Signing Up to GitHub Classroom

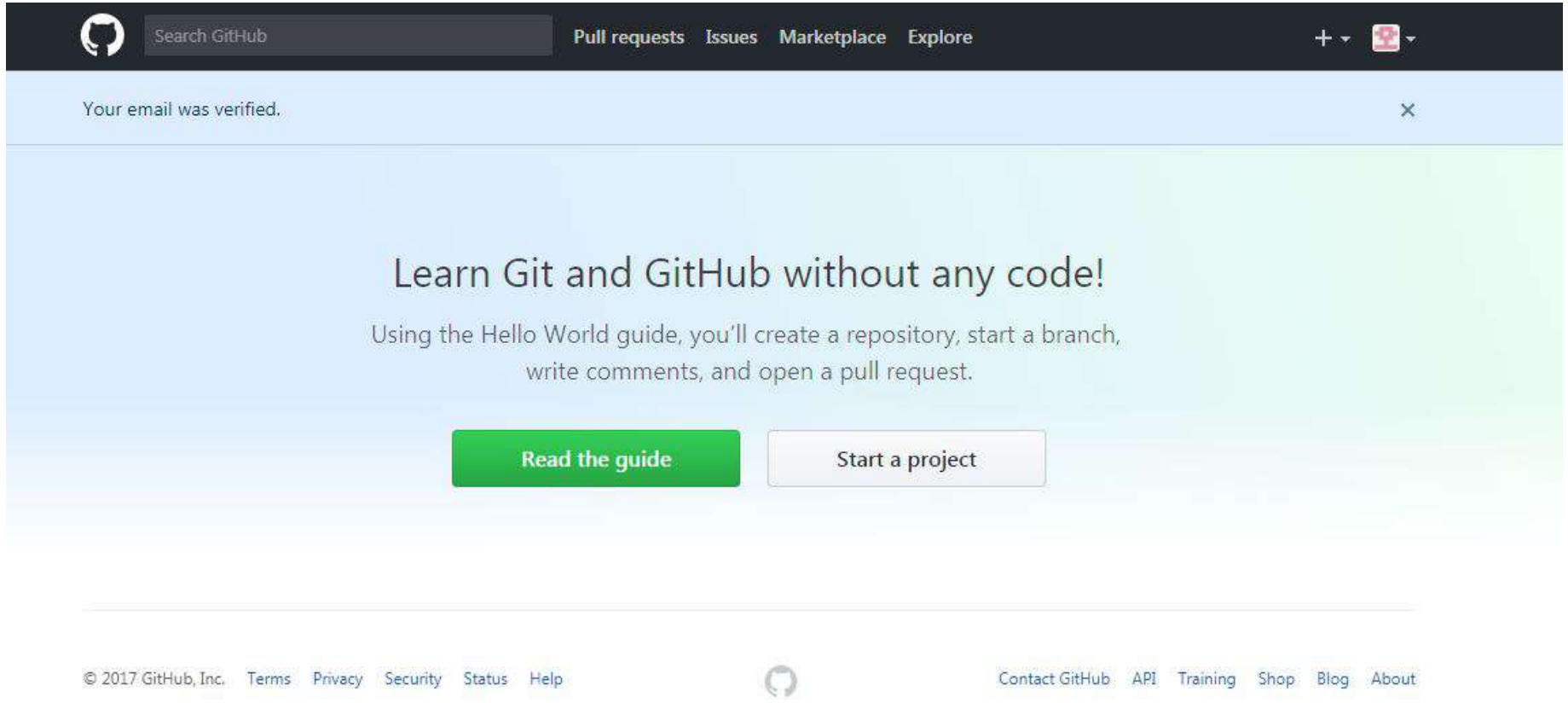
Open your mailbox to verify your github account.



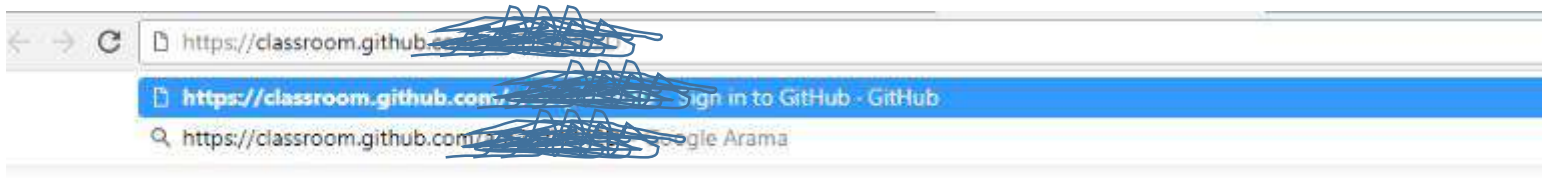
Signing Up to GitHub Classroom



Signing Up to GitHub Classroom

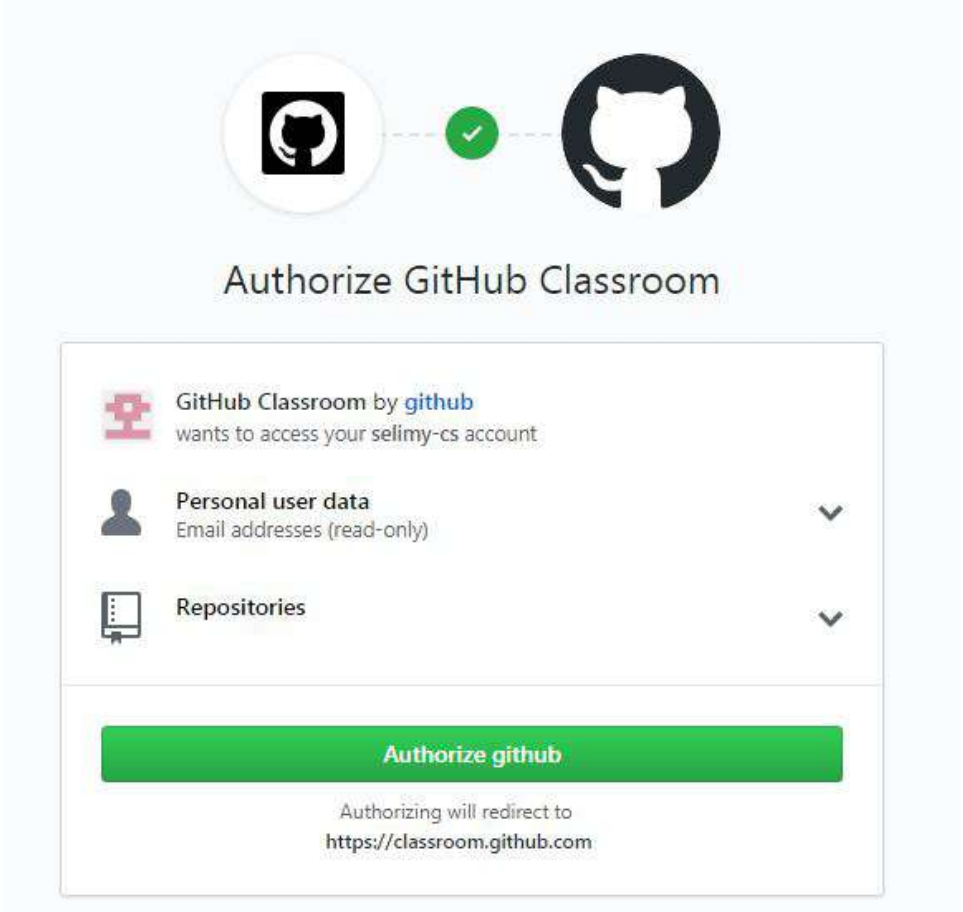


Signing Up to GitHub Classroom

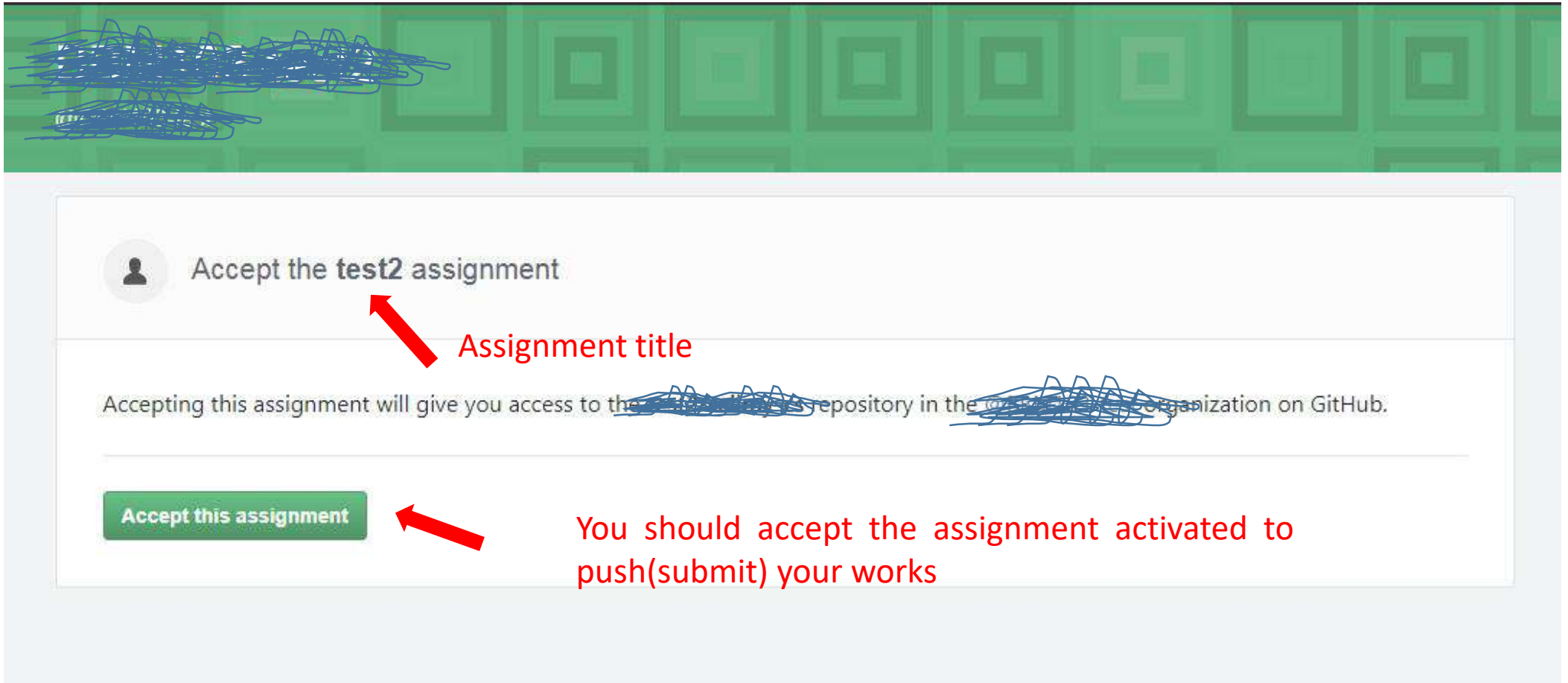


Joining BBM103 Classroom

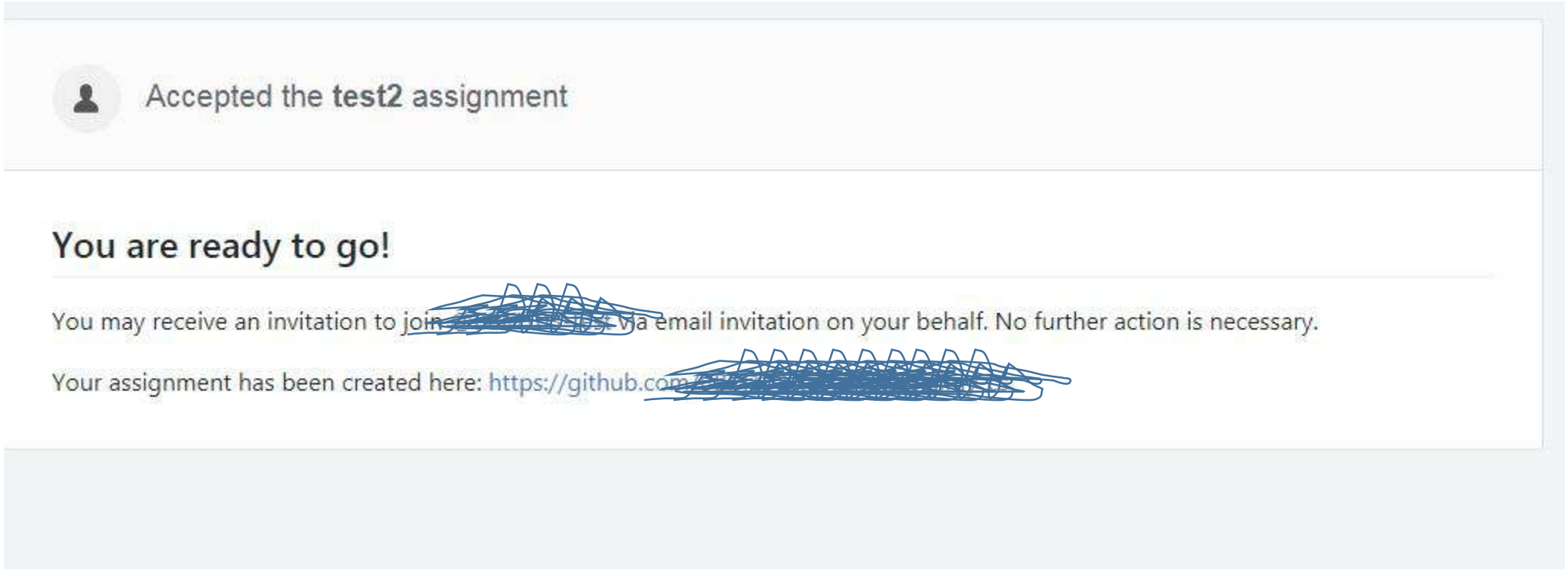
Now authorize github account.



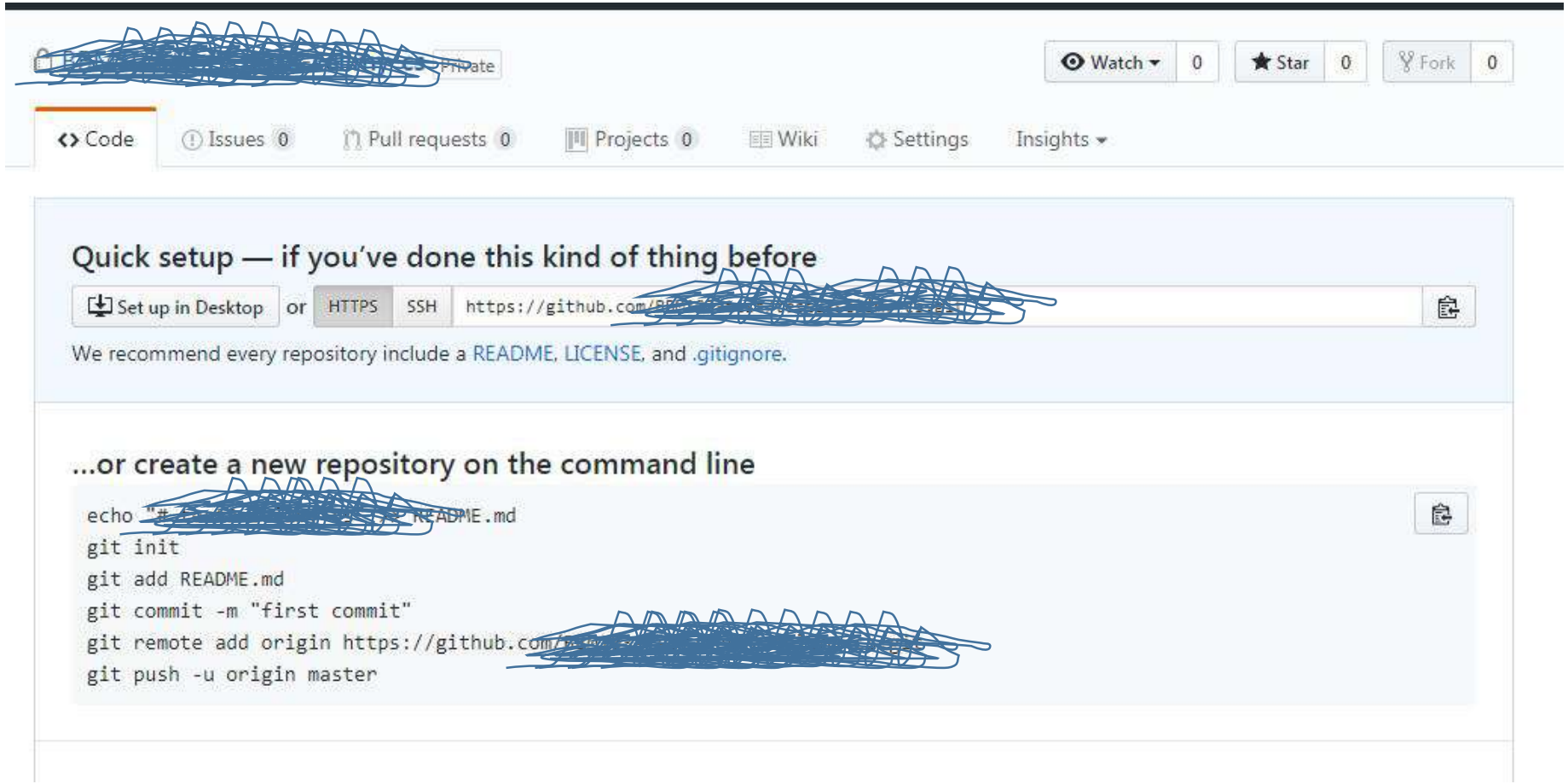
Joining BBM103 Classroom



Joining BBM103 Classroom



Joining BBM103 Classroom



Hacettepe University

Computer Engineering Department

How to Use the Linux Command Line

BBM103 Introduction to Programming Lab I

Fall 2017

The Shell & Terminal

- **The Shell** is a program that takes commands from the keyboard and gives them to the operating system to perform.

- **Terminal Emulator** is a program that opens a window and lets you interact with the shell.

Basic Commands

- When you open a terminal emulator, by default you are in the home directory of the logged in user.
- You will see the name of the logged in user

  followed by the hostname.

  - $ means you are logged in as a regular user

    ```
    [selimy@rdev akd]$
    ```

  - # means you are logged in as root.

    ```
    root@DESKTOP-5HD0AAS:/home/selim#
    ```

pwd

- **pwd** prints the full path of your current working directory.

ls, ll

- You can list all directories and files inside the current directory by using the **ls** (or **ls -l; ll** for listings including information such as the owner, size, date last modified and permissions) command.

```
[selimy@rdev ~]$ ls
2016Assignment4  bashsc.sh                BBM104_Assignment1  BBM203-17-Y-03      cloud      commandline.sh
App              BBM104-17-B-Assignment3  BBM104PA2           BBM203-17-Y-03_ext1 cloud.old  Maildir
[selimy@rdev ~]$ ll
toplam 76
drwxr-xr-x. 133 selimy akd   4096 Ara 24  2016 2016Assignment4
drwxr-xr-x.   7 selimy akd   4096 Mar 16  2017 App
-rw-r--r--.   1 selimy akd     99 Kas 26  2016 bashsc.sh
drwxr-xr-x.   6 selimy akd   4096 May  4 10:50 BBM104-17-B-Assignment3
drwxrwxr-x. 107 selimy akd   4096 Mar 10  2017 BBM104_Assignment1
drwxr-xr-x.   6 selimy akd   4096 Nis 18 21:43 BBM104PA2
drwxrwxrwx.  28 selimy akd   4096 Ağu 12 13:42 BBM203-17-Y-03
drwxrwxrwx.   6 selimy akd   4096 Ağu 12 13:42 BBM203-17-Y-03_ext1
drwxrwxr-x+   6 selimy akd   4096 Eki  9  2016 cloud
drwxr-xr-x.   4 root   root  4096 Eki  9  2016 cloud.old
-rw-r--r--.   1 selimy akd     86 Kas 26  2016 commandline.sh
drwxr-xrwx+   9 selimy akd   4096 Ağu 31 12:42 Maildir
```

cd

- The **cd** command is used to change the current directory.

```
[selma@rdev test]$ ls
some_directory_1  some_directory_2
[selma@rdev test]$ cd some_directory_1
[selma@rdev some_directory_1]$
```

- To change to the parent of the current directory use **cd ..**

```
[selma@rdev some_directory_1]$ cd ..
[selma@rdev test]$
```

- To return directly to the home directory use a tilde as the argument:

```
[selma@rdev test]$ cd ~
[selma@rdev ~]$
```

Manipulating Files

- **cp** - copy files and directories
- **rm** - remove files and directories
- **mv** - move or rename files and directories
- **mkdir** - create directories
- **cat** - create new file, concatenate files

cp

- **cp** copies files and directories. In its simplest form, it copies a single file:

```
[necva@rdev ~]$ ls
bbm103  but         error           Maildir                 output
bm104   cloud       Graph-Cluster   monopoly                public_html
bm3     cloud.old   HelloWorld.py   Morpheme-Graphcluster   Word2VecJava-master
[necva@rdev ~]$ cp HelloWorld.py Hello.py
[necva@rdev ~]$ ls
bbm103  cloud           Hello.py        Morpheme-Graphcluster
bm104   cloud.old       HelloWorld.py   output
bm3     error           Maildir         public_html
but     Graph-Cluster   monopoly        Word2VecJava-master
[necva@rdev ~]$
```

cp (cont.)

- You can specify the full path to where you want to copy your file:

```
[necva@rdev ~]$ ls
bbm103   but          error           Maildir                  output
bm104    cloud        Graph-Cluster   monopoly                 public_html
bm3      cloud.old    HelloWorld.py   Morpheme-Graphcluster    Word2VecJava-master
[necva@rdev ~]$ cp HelloWorld.py Hello.py
[necva@rdev ~]$ ls
bbm103   cloud          Hello.py        Morpheme-Graphcluster
bm104    cloud.old      HelloWorld.py   output
bm3      error          Maildir         public_html
but      Graph-Cluster  monopoly        Word2VecJava-master
[necva@rdev ~]$ cp HelloWorld.py bbm103/HelloWorld.py
[necva@rdev ~]$ cd bbm103
[necva@rdev bbm103]$ ls
HelloWorld.py
[necva@rdev bbm103]$
```

rm

If you want to delete any file or directory the command is '**rm**' (for files) and '**rm -r**' (for directories).

```
[necva@rdev ~]$ ls
bbm103  but         error          Maildir                  output
bm104   cloud       Graph-Cluster  monopoly                 public_html
bm3     cloud.old   HelloWorld.py  Morpheme-Graphcluster    Word2VecJava-master
[necva@rdev ~]$ rm -r bbm103
[necva@rdev ~]$ ls
bm104   cloud       Graph-Cluster  monopoly                 public_html
bm3     cloud.old   HelloWorld.py  Morpheme-Graphcluster    Word2VecJava-master
but     error       Maildir        output
[necva@rdev ~]$
```

mv

- **mv** command moves or renames files and directories depending on how it is used.

```
[necva@rdev ~]$ mv Hello.py bbm103
[necva@rdev ~]$ ls
bbm103   but         error          monopoly               public_html
bm104    cloud       Graph-Cluster  Morpheme-Graphcluster  Word2VecJava-master
bm3      cloud.old   Maildir        output
[necva@rdev ~]$ cd bbm103
[necva@rdev bbm103]$ ls
Hello.py
[necva@rdev bbm103]$
```

mkdir

- If you want to create new directories the command is **mkdir**.

```
[necva@rdev ~]$ mkdir bbm103
[necva@rdev ~]$ ls
bbm103   but         error          Maildir                  output
bm104    cloud       Graph-Cluster  monopoly                 public_html
bm3      cloud.old   HelloWorld.py  Morpheme-Graphcluster    Word2VecJava-master
[necva@rdev ~]$
```

cat

cat stands for **Concatenate (birleştirmek)**. It is used to **create new file** (with or without content), **concatenate files** and **display the output of files on the standard output**.

```
[necva@rdev ~]$ cat >newFile.txt
This file is created to show how we can create file.
You must type Ctrl+D to quit
[necva@rdev ~]$
```

```
[necva@rdev ~]$ ls
bbm103   cloud          Graph-Cluster           newFile.txt
bm104    cloud.old      Maildir                 output
bm3      #deneme.txt#   monopoly                public_html
but      error          Morpheme-Graphcluster   Word2VecJava-master
[necva@rdev ~]$ cat <newFile.txt
This file is created to show how we can create file.
You must type Ctrl+D to quit
[necva@rdev ~]$
```

zip & unzip

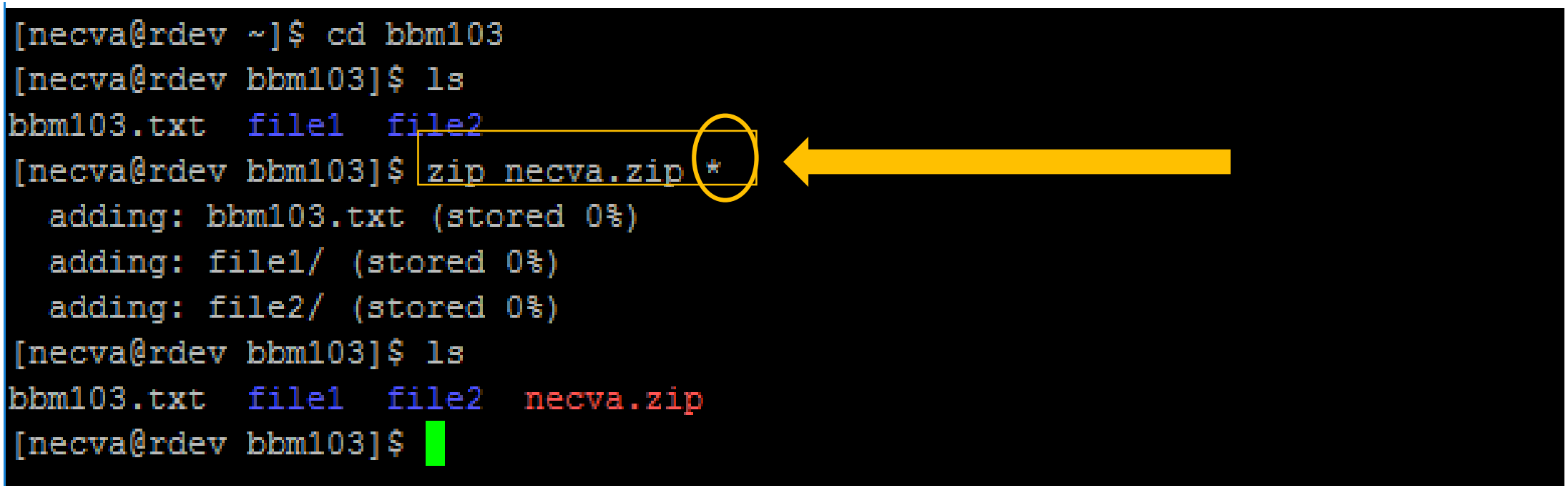
- **zip** and **unzip** commands create and extract zip archive files respectively.

```
[necva@rdev ~]$ cd bbm103
[necva@rdev bbm103]$ ls
bbm103.txt  file1  file2
[necva@rdev bbm103]$ zip necva.zip *
  adding: bbm103.txt (stored 0%)
  adding: file1/ (stored 0%)
  adding: file2/ (stored 0%)
[necva@rdev bbm103]$ ls
bbm103.txt  file1  file2  necva.zip
[necva@rdev bbm103]$
```

```
[selimy@rdev BBM103Linux]$ unzip test.zip -d testDir
Archive:  test.zip
  inflating: testDir/bashsc.sh
  inflating: testDir/bashsc.sh.bak
[selimy@rdev BBM103Linux]$ ls
bashsc.sh  bashsc.sh.bak  testDir  test.zip
[selimy@rdev BBM103Linux]$ cd testDir/
[selimy@rdev testDir]$ ls
bashsc.sh  bashsc.sh.bak
[selimy@rdev testDir]$
```

*

- The * character serves as a "wild card" for filename expansion. By itself, it matches every filename in a given directory.



- Most executable programs intended for command line use provide a formal piece of documentation called a ***manual*** or ***man page***. A special paging program called **man** is used to view them.

```
[necva@rdev bbm103]$ clear
[necva@rdev bbm103]$ man ls
LS(1)                          User Commands                          LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
       Sort entries alphabetically if none of -cftuvSUX nor --sort  is  speci□
       fied.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..
```

ssh

- **ssh** (Secure Shell client) is a program for logging into a remote machine and for executing commands on a remote machine.

```
selim@DESKTOP-5HD0AAS:~$ ssh cemil@dev.cs.hacettepe.edu.tr
cemil@dev.cs.hacettepe.edu.tr's password:
```

scp

- **scp** allows files to be copied to, from, or between different hosts. It uses **ssh** for data transfer and provides the same authentication and same level of security as **ssh**.

A simple example that illustrates how to send a file to dev space.

scp <localfile> <username>@dev.cs.hacettepe.edu.tr:/home/ogr/b****/<directory>

```
selim@selim-PC:~$ scp DPSO.pdf selimy@dev.cs.hacettepe.edu.tr:/home/akd/selimy/
selimy@dev.cs.hacettepe.edu.tr's password:
```

About chmod

- **chmod** is used to change the permissions of files or directories.
- Example: chmod 777 myFile

#	Permission	rwx
7	read, write and execute	rwx
6	read and write	rw-
5	read and execute	r-x
4	read only	r--
3	write and execute	-wx
2	write only	-w-
1	execute only	--x
0	none	---

Exercise

- **All tasks must be performed using linux commands:**
  1) Make a new directory named **`playing_with_linux_cmd`**
  2) Change your current working directory to the newly created one.
  3) List the contents of this directory to see that it is empty.
  4) Create a new text file **`jibberish.txt`** and write something funny in it before closing it.
  5) Create another new text file **`README.txt`** and write your life motto in it.
  6) Copy **`jibberish.txt`** into a text file named **`wise_sayings.txt`**
  7) Delete **`jibberish.txt`**
  8) Print out the content of **`wise_sayings.txt`**
  9) Create a new directory named **`my_precious`** and move **`wise_sayings.txt`** into that newly created directory. List the content of the current working directory to make sure that you have successfully moved the file.
  10) Change the permission of the file **`wise_sayings.txt`** to **read, write and execute**.
  11) Change your working directory to the parent directory of **`playing_with_linux_cmd`**
  12) Zip **`playing_with_linux_cmd`** as **`gameover.zip`**
  13) Use **scp** command to copy this zipped folder from your local computer to your home directory on our remote server **dev.cs.hacettepe.edu.tr**

Hacettepe University

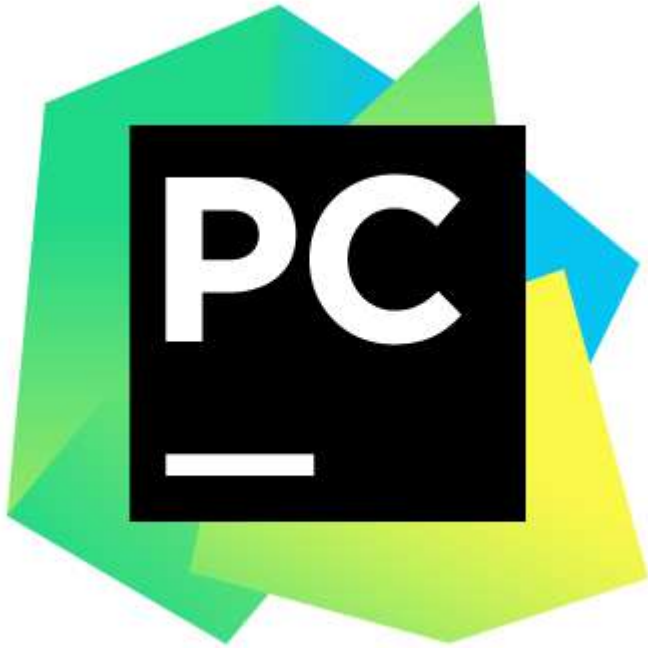
Computer Engineering Department

Programming in  python

BBM103 Introduction to Programming Lab 1

Week 4

Fall 2017



Install PyCharm

Download Link : https://www.jetbrains.com/pycharm-edu/download/#section=windows

Guide : https://www.jetbrains.com/help/pycharm/quick-start-guide.html

Writing Your First Program

- Example 1: **printing output**

```
print('Hello, World!')
```

New function:
`print()`

- Example 2:

```
language = "Python programming language"
print(language)
```

- Example 3: **printing multiple lines**

```
print("""
|===========BBM103===========|
|                            |
|  Welcome to Programming!   |
|  Python is easy and fun!   |
|                            |
|============================|
""")
```

- Example 4: **customizing the separator between printed items**

```
print("L", "i", "n", "u", "x", sep=".")

print(*"Linux", sep=".")
```

  - **Output of both lines:** `L.i.n.u.x`

Taking Input

- Example 5: **taking the input as a string**

```
name = input("What is your name? ")
print("Hello", name, end="!\n")
```

New function:
`input()`

- Example 6: **converting the input to integer**

```
number = int(input("Please enter a number: "))
print("The square of the number: ", number ** 2)
```

New function:
`int()`

```
# We can do the same operation with pow() function: pow(number,2)
```

- Example 7:

```
number1 = int(input("Enter the first number: "))
number2 = int(input("Enter the second number: "))
print(number1, "+", number2, "=", number1 + number2)
```

New function:
`str.format()`

- Example 8: **formatting output**

```
url = input("Please enter the url")
print("Error! Google Chrome couldn't find {} ".format(url))
```

- Example 9: **formatting output**

```
print('{0} and {1}'.format('Tom', 'Jerry'))
```

  - **Output:** `Tom and Jerry`

Control Flow - Branching

The simplest branching statement is a **conditional**. `<condition>` has a value `True` or `False`. `<expression>` is evaluated if `<condition>` is `True`.

1

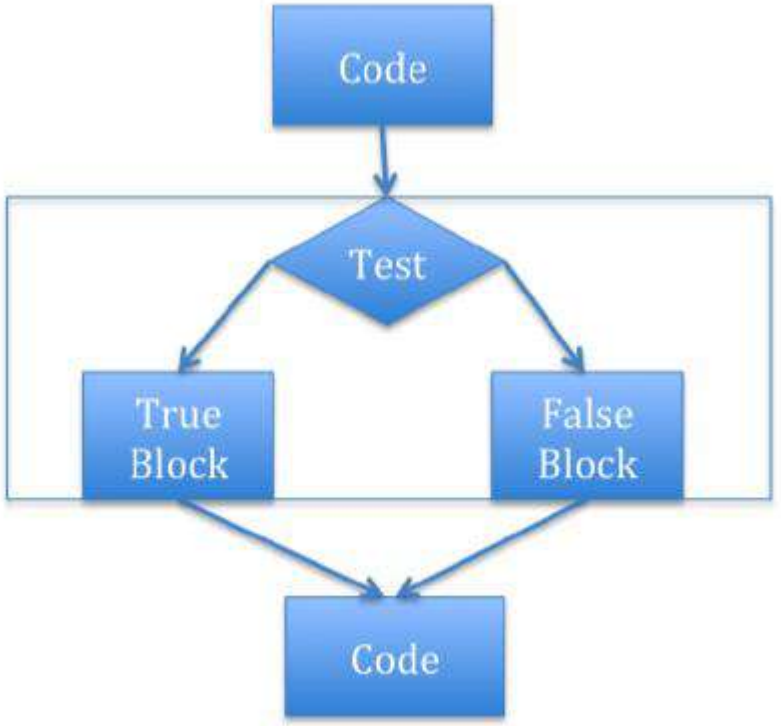
```
if <condition>:
    <expression>
    <expression>
    ...
```

2

```
if <condition>:
    <expression>
    <expression>
    ...
else:
    <expression>
    <expression>
    ...
```

3

```
if <condition>:
    <expression>
    <expression>
    ...
elif <condition>:
    <expression>
    <expression>
    ...
else:
    <expression>
    <expression>
    ...
```



- Example 10: **control flow**

```
question = input("Please enter a fruit name: ")
if question == "apple":
    print("Yes, apple is a fruit")
elif question == "banana":
    print("Yes, banana is a fruit")
elif question == "strawberry":
    print("Yes, banana is a fruit")
else:
    print("Your input " + question + " isn't a fruit.")
```

'==' is a comparison operator as opposed to '=' sign which is an assignment operator.

- Comparison operators in Python:

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

- Example 11: **control flow continued**

New function:
`len()`

```
username = input("Your username: ")
password = input("Your password : ")
total_weight = len(username) + len(password)
message = "Your username and password has a total of {} characters!"
print(message.format(total_weight))
if total_weight > 40:
    print("The total length of your username and password ",
          "should not exceed 40 characters!")
else:
    print("Welcome to the system!")
```

len() returns the length of its argument.

- Example 12: **control flow continued – checking if two numbers are divisible**

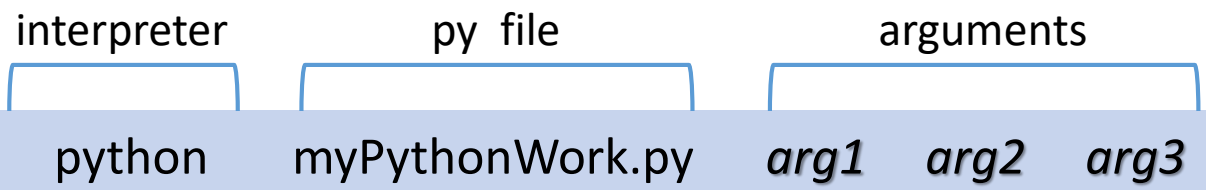
```
number1 = int(input("Please enter the number to be divided: "))
number2 = int(input("Please enter the divisor: "))
if number1 % number2 == 0:
    print("{} can divide {} without a remainder!".format(number2, number1))
else:
    print("{} can't divide {} without a remainder!".format(number2, number1))
```

'%' is a modulus operator which divides the left hand operand by the right hand operand and **returns the remainder**.

Command-line Arguments in Python

- Python provides a **getopt** module that allows you to use command line arguments.
- To access to any command-line argument you should use **sys** module.
- This modules provides two functionalities:
    1. **sys.argv** is the list of command-line arguments
    2. **len(sys.argv)** is the number of command-line arguments.

Example:



sys.argv[0] is the program file name, i.e., *myPythonWork.py*
sys.argv[1] is *arg1* whereas sys.argv[2] is *arg2*, and the like.

Exercises

1. Print a message which states whether a year which is taken as input is a leap year or not to the screen.

2. Convert a number which is taken from the input to binary format and print it to the screen.

3. Calculate the roots of $x^2 + bx + c = 0$ and print the result to the screen (b and c are taken from the input).

Exercise Solutions

```
"""Exercise-1 : Leap year"""

year = int(input("Please write a year to check whether it is a leap year or not\n"))

if year % 4 == 0 :
    print("{} is a leap year".format(year))
else :
    print("{} is not a leap year".format(year))
```

```
"""Exercise-2 : Number basen on 2"""

number = int(input("Please write a number\n"))
baseTwo =""

while number >= 2:
    baseTwo = str(number % 2) + baseTwo
    number = number // 2

baseTwo = str(number) + baseTwo
print(baseTwo)
```

```
"""Exercise-3 : Calculate of the roots of quadratic equation"""

numberB = int(input("Please write value of b\n"))
numberC = int(input("Please write value of c\n"))
numberA = 1

delta = pow(numberB , 2) - 4 * numberA * numberC

if delta > 0:
    root1 = (-numberB + pow (delta , 0.5)) / 2 * numberA
    root2 = (-numberB - pow (delta , 0.5)) / 2 * numberA

    print("Roots of the equation is {0} and {1}.".format(root1,root2))
elif delta ==0:
    root = (-numberB) / 2 * numberA

    print("Root of the equation is {}.".format(root))
else:
    print("There is no root in this equation")
```

Hacettepe University

Computer Engineering Department

Programming in  python

BBM103 Introduction to Programming Lab 1

Week 6

Fall 2017

Exercises

1. **Write a function that finds the *n*th largest element of the given list.**

```
Input: L = [1,5,6,4,2], n=3

Output: 4
```

2. **Write a function that determines if the given input string is a Palindrome or not.**

   - **A *palindrome* is a sequence of characters which reads the same backward as forward**

Exercises

3. Implement the following integer functions:

- a) Function ***celcius*** returns the Celsius equivalent of a Fahrenheit temperature.
- b) Function ***fahrenheit*** returns the Fahrenheit equivalent of a Celsius temperature.

$$F = \frac{9}{5}C + 32$$

Celsius to Fahrenheit Formula

Exercises

4. **Write a function *perfect* that determines if a number given as a parameter is a perfect number or not. Use this function in a program that determines and prints all the perfect numbers between 1 and 1000.**

- **Perfect Number:**
- **An integer number is said to be a *perfect number* if its factors, including 1 (but not the number itself), sum to the number. For example, 6 is a perfect number because 6 =1 + 2 + 3.**

Hacettepe University

Computer Engineering Department

Programming in HMMM

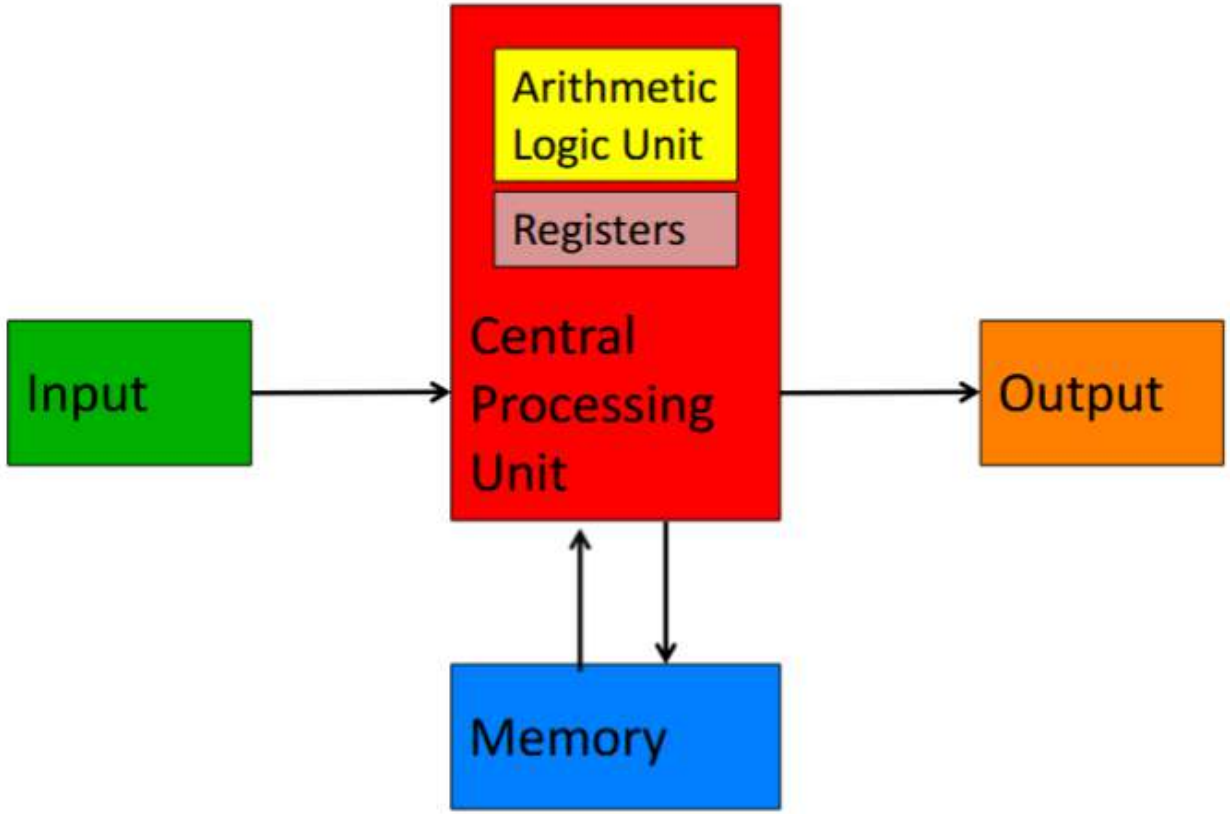
BBM103 Introduction to Programming Lab 1

Week 3

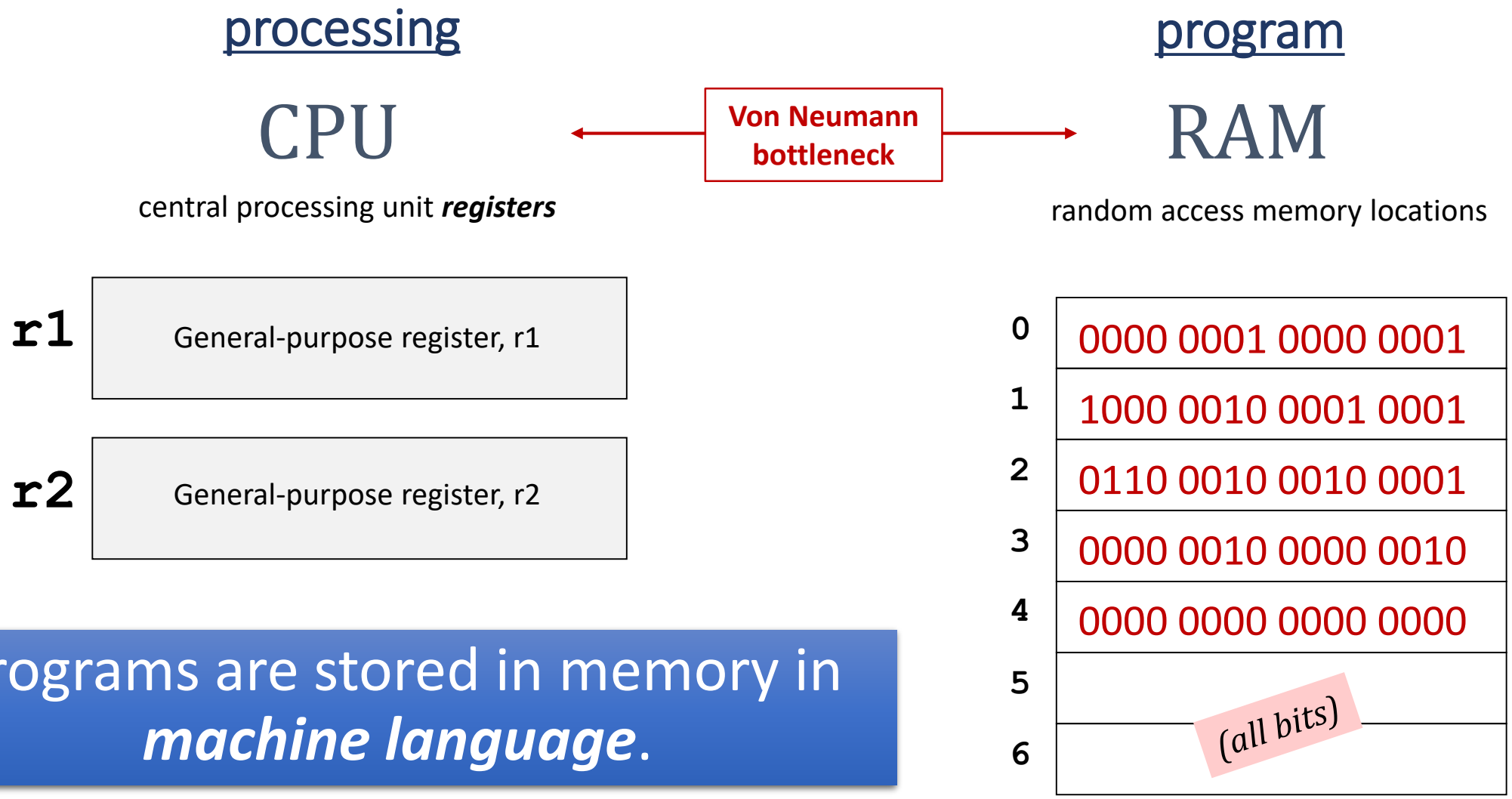
Fall 2017

Von Neumann Architecture

- A **program** (a list of instructions) is stored in the main memory.
  - **Stored Program Concept**
- Instructions are copied (one at a time) into the **instruction register** in the CPU for execution.



Von Neumann Architecture



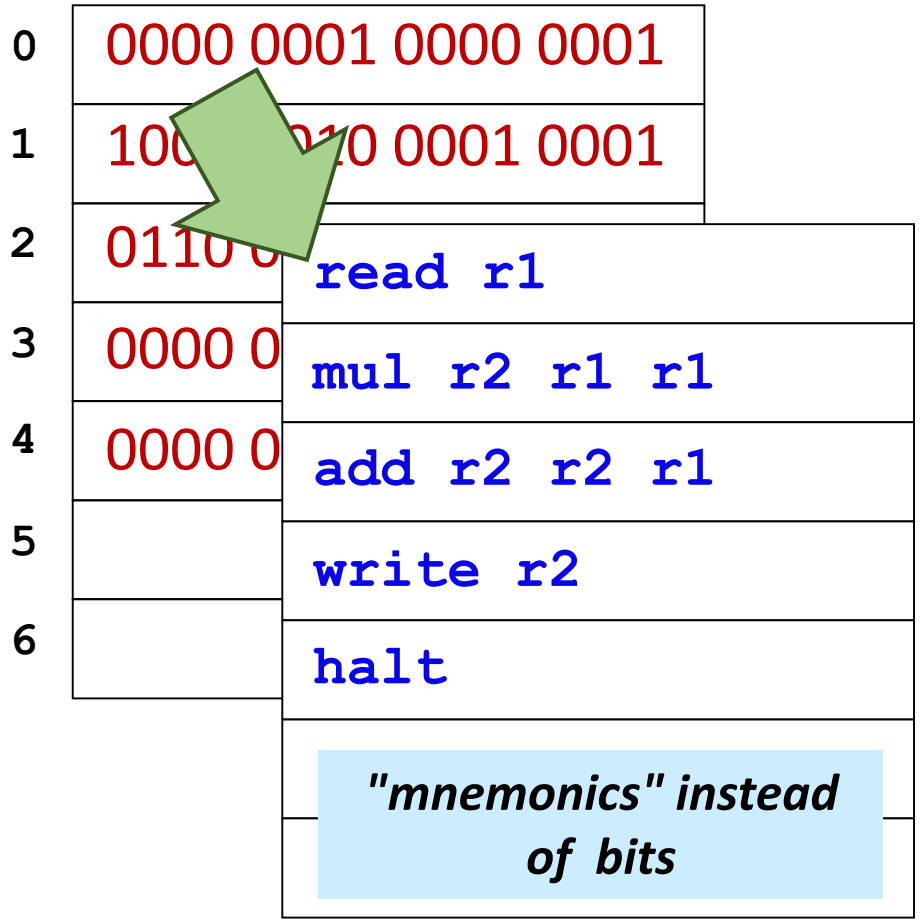
Programs are stored in memory in ***machine language***.

The Power of the Stored Program

- A program written in machine language is a series of binary numbers representing the instructions stored in memory.
- The ***stored program*** concept is a key reason why computers are so powerful:
  - **Running a different program does not require large amounts of time and effort to reconfigure or rewire hardware; it only requires writing the new program to memory.**

Assembly Language

- **Assembly language** is a human-readable machine language.
- Instead of programming in binary (0's and 1's), it is easier to use an assembly language.
- An ***assembler*** is a computer program that interprets software programs written in assembly language into machine language.



The Harvey Mudd Miniature Machine (HMMM)

- Hmmm (Harvey Mudd Miniature Machine) is a 16-bit, 23-instruction simulated assembly language with 2^8=256 16-bit words of memory.

- In addition to the **program counter** and **instruction register**, there are 16 registers named `r0` through `r15`.

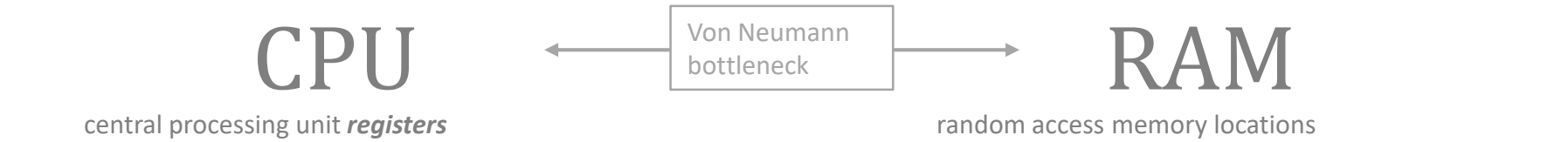
Hmmm assembly code

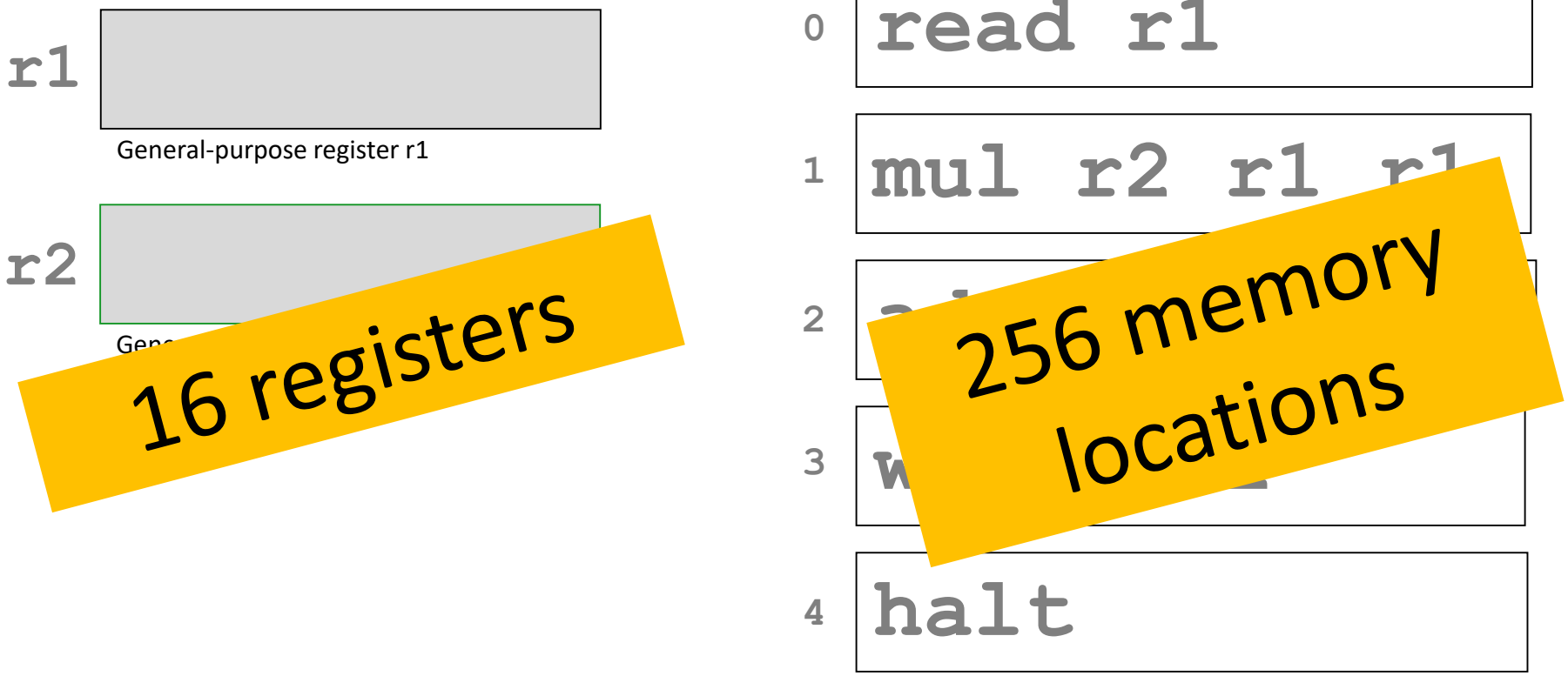
```
0     read    r1
1     read    r2
2     mul     r1  r1  r2
3     setn    r2  2
4     div     r1  r1  r2
5     write   r1
6     halt
```

Corresponding instructions in machine language

```
0000 0001 0000 0001
0000 0010 0000 0001
1000 0001 0001 0010
0001 0010 0000 0010
1001 0001 0001 0010
0000 0001 0000 0010
0000 0000 0000 0000
```

The Harvey Mudd Miniature Machine (HMMM)





The Harvey Mudd Miniature Machine (HMMM)

```
read r1
```
reads from keyboard into **reg1**

```
write r2
```
outputs **reg2** onto the screen

```
setn r1 42
```
reg1 = 42

you can replace 42 with anything from -128 to 127

```
addn r1 -1
```
reg1 = reg1 - 1

a shortcut

```
add r3 r1 r2
```
reg3 = reg1 + reg2

```
sub r3 r1 r2
```
reg3 = reg1 - reg2

```
mul r2 r1 r1
```
reg2 = reg1 * reg1

```
div r1 r1 r2
```
reg1 = reg1 / reg2

integers only!

The Harvey Mudd Miniature Machine (HMMM)

Instruction	Description	Aliases
System instructions		
halt	Stop!	
read rX	Place user input in register rX	
write rX	Print contents of register rX	
nop	Do nothing	
Setting register data		
setn rX N	Set register rX equal to the integer N (-128 to +127)	
addn rX N	Add integer N (-128 to 127) to register rX	
copy rX rY	Set rX = rY	mov
Arithmetic		
add rX rY rZ	Set rX = rY + rZ	
sub rX rY rZ	Set rX = rY - rZ	
neg rX rY	Set rX = -rY	
mul rX rY rZ	Set rX = rY * rZ	
div rX rY rZ	Set rX = rY / rZ (integer division; no remainder)	
mod rX rY rZ	Set rX = rY % rZ (returns the remainder of integer division)	
Jumps!		
jumpn N	Set program counter to address N	
jumpr rX	Set program counter to address in rX	jump
jeqzn rX N	If rX == 0, then jump to line N	jeqz
jnezn rX N	If rX != 0, then jump to line N	jnez
jgtzn rX N	If rX > 0, then jump to line N	jgtz
jltzn rX N	If rX < 0, then jump to line N	jltz
calln rX N	Copy the next address into rX and then jump to mem. addr. N	call
Interacting with memory (RAM)		
loadn rX N	Load register rX with the contents of memory address N	
storen rX N	Store contents of register rX into memory address N	
loadr rX rY	Load register rX with data from the address location held in reg. rY	loadi, load
storer rX rY	Store contents of register rX into memory address held in reg. rY	storei, store

Hmmm

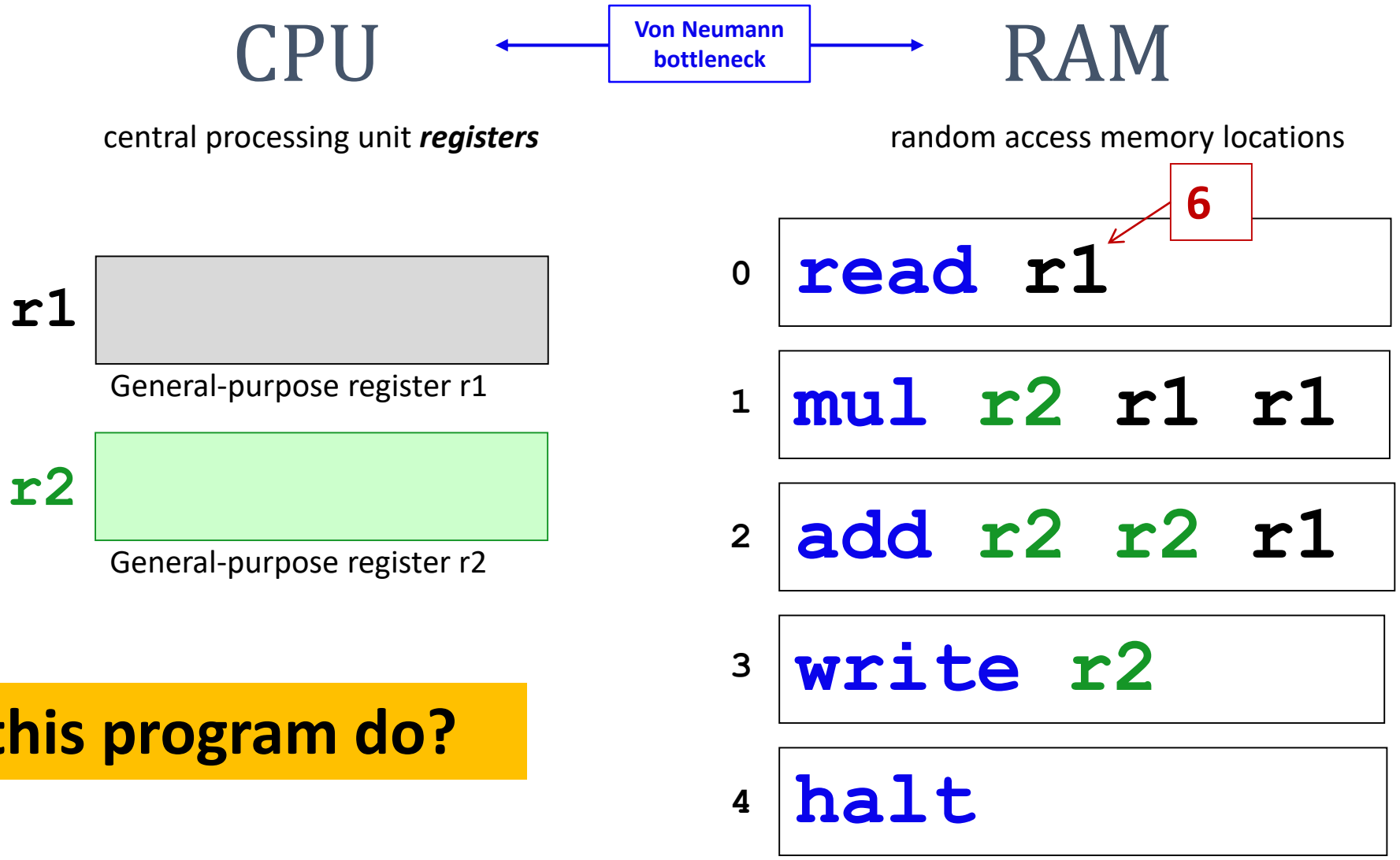
the complete reference

At
www.cs.hmc.edu/~cs5grad/cs5/hmmm/documentation/documentation.html

Example #1:

Screen

6 (input)

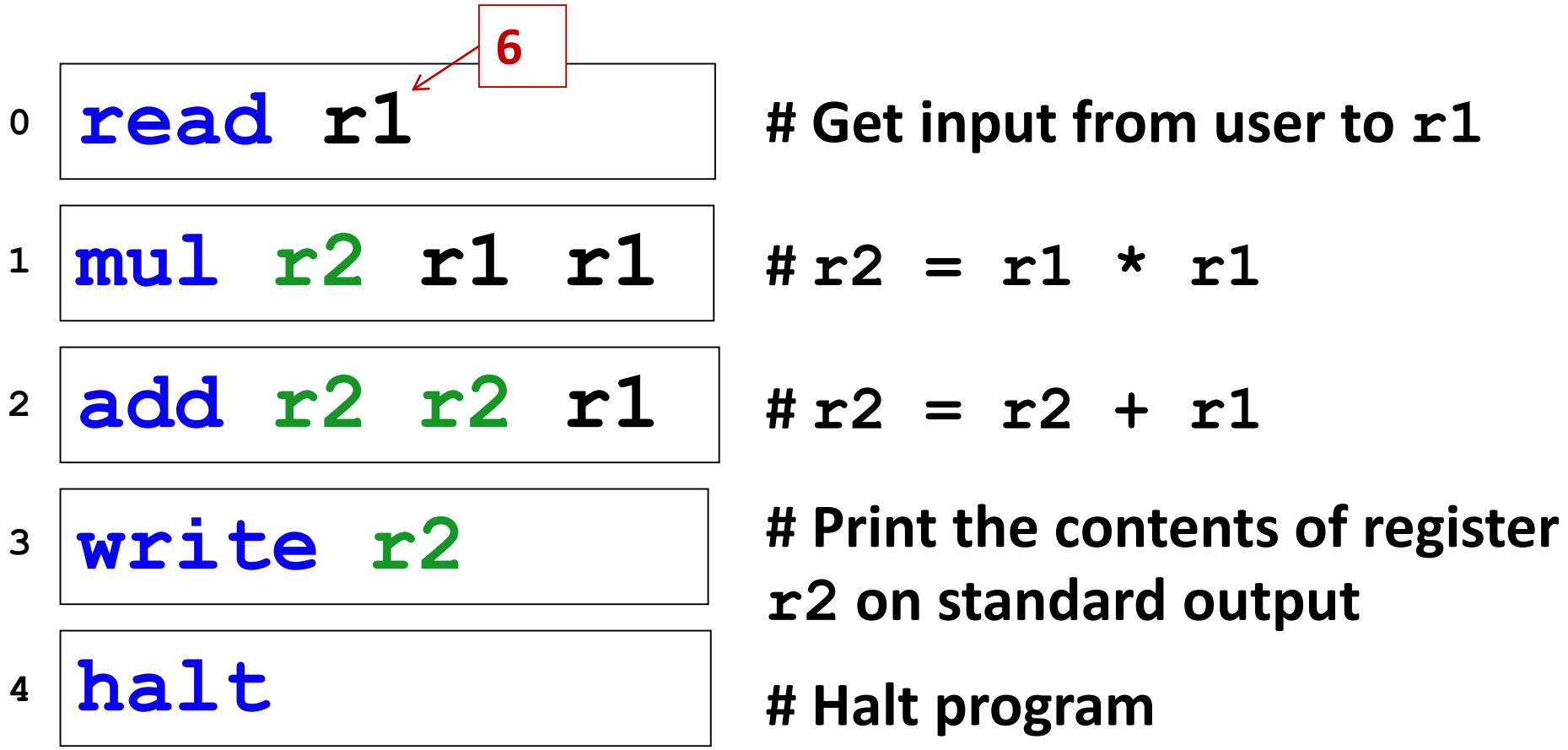


What does this program do?

Example #1 (cont.):

Screen

6 (input)



Jumps in HMMM

`jeqzn r1 42`	IF **r1 == 0** THEN jump to line number **42**
`jgtzn r1 42`	IF **r1 > 0** THEN jump to line number **42**
`jltzn r1 42`	IF **r1 < 0** THEN jump to line number **42**
`jnezn r1 42`	IF **r1 != 0** THEN jump to line number **42**

Unconditional jump

`jumpn 42` Jump to program line # **42**

Indirect jump

`jumpr r1` Jump to the line# ***stored*** in **r1**

Example #2:

Screen

-6 (input)

RAM

0	`read r1`
1	`jgtzn r1 7`
2	`setn r2 -1`
3	`mul r1 r1 r2`
4	`nop`
5	`nop`
6	`nop`
7	`write r1`
8	`halt`

Space for future expansion!

What function does this program implement?

Exercise

1. Write a Hmmm program to compute the following for **x** given as user input and output the result to the screen:

   a) If **x<0** → **3x – 4**

   b) else if x>0 → **X / 5**

   c) else → **X^2+10 / 5**

Hacettepe University

Computer Engineering Department

Programming in  python

BBM103 Introduction to Programming Lab 1

Week 5

Fall 2017

Control Flow – For Loops

```
for <variable> in range(some_number):
    <expression>
    <expression>
    ...
```

- Each time through the loop, `<variable>` takes a new value. It starts with the smallest value, and in the next loop it gets incremented, and so on, until it reaches the final value in the specified range.

- Example 1: **printing numbers in a given range**

```
for i in range(10):
    print(i)
```

New function:
`range()`

- Example 2: **printing numbers grater than a specified value**

```
numbers = "123456789"
for i in numbers:
    if int(i) > 3:
        print(i)
```

`range(number)` **generates integers from 0 up to, but not including** `number`.

- Example 3: **printing characters that are not in a string**

```
first_text = "This is a sample text for testing."
second_text = "This is another sample text."
for letter in first_text:
    if letter not in second_text:
        print(letter)
```

- Example 4: **printing numbers divisible by three**

```
for number in range(2,50):
    if int(number) % 3 == 0:
        print(number)
```

`range(start, stop)` generates integers from `start` up to `stop`, but not including `stop`.

- Example 5: **finding the cube root**

```
x = int(input('Enter an integer: '))
answer = None
cube_root_found = False
for i in range(0, abs(x)+1):
    if i**3 == abs(x):
        answer = i
        cube_root_found = True
if not cube_root_found:
    print (x, 'is not a perfect cube')
else:
    if x < 0:
        answer = -answer
    print('Cube root of', x,'is', answer)
```

New function:
`abs()`

This is not a very efficient algorithm, but it gets the job done!

Food for thought:

- **Why?**
- **How can we make it more efficient?**

`abs(number)` **returns the absolute value of** `number`.

- Example 6: **split**

```
sentence="Yürüdüğümüz yol bitmiş , bir başka sokağa açılmıştı ."

for word in sentence.split():
    print(word)
```

New function:
`split()`

- Example 7: **even numbers**

```
numbers="12,15,47,86,98"

for number in numbers.split(","):
    if int(number)%2 ==0:
        print(number,"is even")
```

Control Flow – While Loops

```
while (condition is True):
    <expression>
    <expression>
    ...
```

- *While* loops are used for repeating sections of code until a defined condition is no longer met. If the condition is initially false, the loop body will not be executed at all.

A loop becomes infinite loop if a condition never becomes FALSE. You must use caution when using while loops because of the possibility that this condition never resolves to a FALSE value. This results in a loop that never ends. Such a loop is called an infinite loop.

- Example 8: **input condition**

```
n = input("Please enter 'hello':")
while n.strip() != 'hello':
    n = input("Please enter 'hello':")
```

- Example 9: **subtraction**

```
i = 0

# While loop condition.
while i > 100:
    print(i)
    # Subtract two.
    i -= 2
```

- Example 10: **guess**

```
import random

number = random.randint(1, 25)

number_of_guesses = 0

guess=0

while number_of_guesses < 5 and guess!=number:
    print('Guess a number between 1 and 25:')

    guess = input("Please enter a number")
    guess = int(guess)

    number_of_guesses = number_of_guesses + 1
```

New function:
`randint()`

Functions

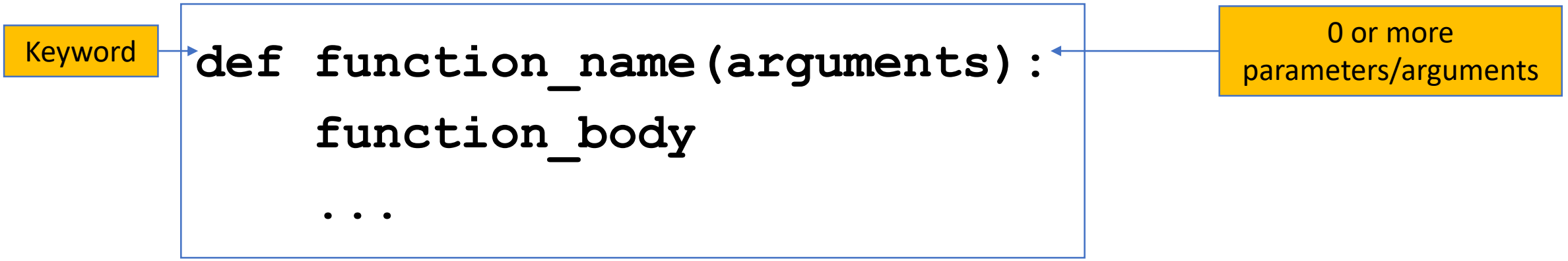
- **Good programming practice**: It is **functionality** that is important, not the amount of code!
- **The importance of functions**:
  - Break your code into separate, independent parts that will work together to solve the ultimate problem (**DECOMPOSITION**).
  - Hide the details of your computation as long as you know what it produces (**ABSTRACTION**)

Functions cont.

- **The advantages of functions**:
  - Break your code into **simpler independent modules**
  - These modules can be **reused** as many times as you like
  - And they need to be **debugged only once**
  - Keep your code **more organized and easier to understand**

Functions cont.

Defining functions:



Calling functions:

```
function_name(arguments)
```

- Example 1: **defining a <u>void</u> function (function that does not return a value)**

```
def greeting(name):
    print("Good afternoon, " + name + ".")

greeting("Emre")
```

Defining function

Calling function

  - **Output:** `Good afternoon, Emre.`

- Example 2: **defining a <u>fruitful</u> function (function that returns a value)**

```
def maximum(x, y):
    if x > y:
        return x
    else:
        return y
max_number = maximum(1, 5)
print("The maximum of two numbers is", max_number)
```

The function **returns** a value

- Example 3: **an example of a Boolean function (function that returns True or False)**

```
def is_even(number):
    return number%2 == 0
```

  - This function will return True if number is even, and False otherwise.

- An example use of this function:

Calling the function from within an **`if`** statement

```
if is_even(number):
    print (number," is an even number.")
else:
    print (number," is an odd number.")
```

- Example 4: **a function that calculates the factorial* of a number**

```
def factorial(number):
    product = 1
    for i in range(1, number+1):
        product = product * i
    return product
```

This line can be written more compactly as:

```
product *= i
```

- An example use of this function:

```
print("The factorial of 6: 6! = ", factorial(6))
```

* The **factorial** of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n. For example, 4! = 4 × 3 × 2 × 1 = 24

- Example 5: **Calculating area of plane shapes**

```
def triangle_area(b, h):
    return b*h/2
def square_area(a):
    return a*a
def rectangle_area(a, b):
    return a*b
user_choice = int(input("""Choose a shape you wish to calculate the area of:
(1) Triangle
(2) Square
(3) Rectangle\n1-3: """))
if user_choice == 1:
    base = int(input("Enter the length of the triangle base: "))
    height = int(input("Enter the height of the triangle: "))
    print("The area of the triangle is", triangle_area(base, height))
elif user_choice == 2:
    side = int(input("Enter the length of the square side: "))
    print("The area of the square is", square_area(side))
elif user_choice == 3:
    width = int(input("Enter the width of the rectangle: "))
    height = int(input("Enter the height of the rectangle: "))
    print("The area of the triangle is", rectangle_area(width, height))
else:
    print("Sorry, there is no such option.")
```

`\n` is the newline character

Exercises

1. Write a program that asks for a number N as a user input, and calculates the sum of odd numbers, and the average of even numbers starting from 1 up to and including N.
2. Write a Boolean function that checks if a string contains '@' sign and at least one '.' dot (disregard the order for the sake of simplicity). Use that function to check if a user input is a valid e-mail or not.
3. Guessing game! Pick a number randomly. While user does not guess the number correctly give an instruction about the number and take another guess from user.

   Instruction: If the guessed number is lower than the picked number print

   «increase your number»

   else print

   «decrease your number»

Things to remember

- Indentation is very important in Python! To indicate a block of code in Python, you **must indent each line of the block by the same amount**.

- **Practice makes perfect**: the more you practice programming the easier it gets. It is easy to get stuck in the beginning, but don't get discouraged. Work with simple examples first. Move on to the harder examples when you have fully grasped the simple ones.

- It is a lot of fun telling your computer what to do! **Stay motivated**.

Hacettepe University

Computer Engineering Department

Programming in  python

BBM103 Introduction to Programming Lab 1

Week 7

Fall 2017

Collections

- **A Collection Groups Similar Things**

  List: ordered

  Set: unordered, no duplicates

  Tuple: unmodifiable list

  Dictionary: maps from keys to values

Lists

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is **that items in a list need not be of the same type**.

Creating a list:

```
my_list = [1, 2, 3, 4, 5]
```

Splitting a string to create a list:

```
s = 'spam-spam-spam'
delimiter = '-'
s.split(delimiter)
```

split()

Output: `['spam', 'spam', 'spam']`

Lists

Making a list of chars in a string:

```
s = 'spam'
t = list(s)
print(t)
```

```
Output:
['s', 'p', 'a', 'm']
```

Joining elements of a list into a string:

```
t = ['programming', 'is', 'fun']
delimiter = ' '
delimiter.join(t)
```

join()

```
Output: 'programming is fun'
```

Accessing Values in Lists by Index:

```
list1 = [1, 2, 3, 4, 5]
print ("list1[0]: ", list1[0])
print ("list1[1:5]: ", list2[1:5])
```

```
Output:
list1[0]: 1
list1[1:5]:  [2, 3, 4, 5]
```

Updating Lists:

```
list = [1, 2, 3, 4, 5]
print ("Value available at index 2: ",list[2])
list[2] = 6
print ("New value available at index 2: ", list[2])
```

```
Output:
Value available at index 2:  3
New value available at index 2:  6
```

Deleting List Elements

```
list = [1, 2, 3, 4, 5]
print(list)
del list[2]
print ("After deleting value at index 2: ",list)
```

```
Output:
[1, 2, 3, 4, 5]
After deleting value at index 2:  [1, 2, 4, 5]
```

Basic List Operations

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

List Functions & Methods:

len (list) : Gives the total length of the list.

Example:

```
list = [1, 2, 3, 4, 5]
print ('length of the list is',len(list))
```

```
Output:
length of the list is 5
```

max(list): Returns the item from the list with the maximum value.

Example:

```
list=[456, 700, 200]
print ("Max value element:", max(list))
```

```
Output:
Max value element: 700
```

min(list): Returns the item from the list with the minimum value.

Example:

```
list=[456, 700, 200]
print ("Min value element:", min(list))
```

```
Output:
Min value element: 200
```

list.append(obj): Appends object obj to list

Example:

```
list = [123, 'xyz', 'zara', 'abc']
list.append(2009)
print ("Updated List: ", list)
```

```
Output:
Updated List:  [123, 'xyz', 'zara', 'abc', 2009]
```

list.count(obj): Returns the count of how many times obj occurs in a list

Example:

```
aList = [123, 'xyz', 'zara', 'abc', 123]
print ("Count for 123: ", aList.count(123))
print ("Count for zara: ", aList.count('zara'))
```

```
Output:
Count for 123:  2
Count for zara:  1
```

list.extend(seq) : Appends the contents of seq to list

Example:

```
aList = [123, 'xyz', 'zara']
bList = [2009, 'manni']
aList.extend(bList)
print ("Extended List: ", aList )
```

```
Output:
Extended List:  [123, 'xyz', 'zara', 2009, 'manni']
```

list.index(obj): Returns the lowest index of obj in the list

Example:

```
list=[456, 700, 200]
print ("Index of 700: ", list.index(700) )
```

```
Output:
Index of 700:  1
```

list.insert(index, obj): Inserts object obj into the list at offset index

Example:

```
aList = [123, 'xyz', 'zara', 'abc']
aList.insert(3, 2009)
print ("Final List: ", aList)
```

```
Output:
Final List:  [123, 'xyz', 'zara', 2009, 'abc']
```

list.pop(obj=list[-1]): Removes and returns the last obj from list

Example:

```
aList = [123, 'xyz', 'zara', 'abc']
print ("A List: ", aList.pop())
print ("B List: ", aList.pop(2))
```

```
Output:
A List:  abc
B List:  zara
```

list.remove(obj): Removes object obj from list

Example:

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz']
aList.remove('xyz')
print ("List: ", aList)
aList.remove('abc');
print ("List: ", aList)
```

```
Output:
List :  [123, 'zara', 'abc', 'xyz']
List :  [123, 'zara', 'xyz']
```

list.reverse(): Reverses the objects of a list

Example:

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz']
aList.reverse()
print ("List: ", aList)
```

```
Output:
List:  ['xyz', 'abc', 'zara', 'xyz', 123]
```

list.sort([func]): Sorts objects of list, uses compare func if given

Example:

```
aList = ['xyz', 'zara', 'abc', 'xyz']
aList.sort()
print ("List: ", aList)
```

```
Output:
List:  ['abc', 'xyz', 'xyz', 'zara']
```

List Comprehensions

```
liste = [i for i in range(1000)]
```

Method 1:

```
liste = [i for i in range(1000) if i % 2 == 0]
```

Method 2:

```
liste = []
for i in range(1000):
    if i % 2 == 0:
        liste += [i]
```

Sets

Sets are lists with no duplicate entries.

Operation	Equivalent	Result
s.update(t)	s |= t	return set s with elements added from t
s.intersection_update(t)	s &= t	return set s keeping only elements also found in t
s.difference_update(t)	s -= t	return set s after removing elements found in t
s.symmetric_difference_update(t)	s ^= t	return set s with elements from s or t but not both
s.add(x)		add element x to set s
s.remove(x)		remove x from set s; raises KeyError if not present
s.discard(x)		removes x from set s if present
s.pop()		remove and return an arbitrary element from s; raises KeyError if empty
s.clear()		remove all elements from set s

Example:

```
list = ["elma", "armut", "elma", "kebap", "şeker",
..."armut","çilek", "ağaç", "şeker", "kebap", "şeker"]
for i in set(list):
   print(i)
```

Output:
çilek
elma
kebap
armut
ağaç
şeker

Example:

```
list = ["elma", "armut", "elma", "kiraz",
... "çilek", "kiraz", "elma", "kebap"]
for i in set(list):
   print("{} count:  {}".format(i, list.count(i)))
```

Output:
armut count: 1
çilek count: 1
elma count: 3
kiraz count: 2
kebap count: 1

Tuples

- A tuple is a sequence of **immutable** Python objects. Tuples are sequences, just like lists.
- What are the differences between tuples and lists ?

Tuples

- A tuple is a sequence of **immutable** Python objects. Tuples are sequences, just like lists.
- The differences between tuples and lists are,
  - the **tuples cannot be changed** unlike lists,
  - tuples use parentheses, whereas lists use square brackets.

Creating a tuple:

```
tup = (1, 2, 3, 4, 5 )
```

Accessing Values in Tuples:

```
tup1 = ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5, 6, 7 )
print ("tup1[0]: ", tup1[0])
print ("tup2[1:5]: ", tup2[1:5])
```

```
Output:
tup1[0]:  physics
tup2[1:5]:  (2, 3, 4, 5)
```

Updating Tuples

```
tup1 = (12, 34.56)
tup2 = ('abc', 'xyz')
tup3 = tup1 + tup2
print (tup3)
```

```
Output:
(12, 34.56, 'abc', 'xyz')
```

Dictionaries

- Dictionary as an unordered set of *key: value* pairs, with the requirement that the keys are **unique** (within one dictionary).

- Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by *keys*, which can be any immutable type.

Creating a dictionary:

```
dict = {'Name':'Zara','Age':7,'Class':'First'}
```

Accessing Values in a Dictionary:

```
dict ={'Name':'Zara','Age':7,'Class':'First'}
print ("dict['Name']: ",dict['Name'])
print ("dict['Age']: ",dict['Age'])
```

```
Output:
dict['Name']:  Zara
dict['Age']:  7
```

Updating a Dictionary

```
dict ={'Name':'Zara','Age':7,'Class':'First'}
dict['Age']=8
dict['School']="DPS School"
print ("dict['Age']: ",dict['Age'])
print ("dict['School']: ",dict['School'])
```

```
Output:
dict['Age']:  8
dict['School']:  DPS School
```

SN	Methods with Description
1	**dict.clear() :** Removes all elements of dictionary dict
2	**dict.copy() :** Returns a shallow copy of dictionary dict
3	**dict.fromkeys() :** Create a new dictionary with keys from seq and values set to value.
4	**dict.get(key, default=None) :** For key key, returns value or default if key not in dictionary
5	**dict.has_key(key) :** Returns true if key in dictionary dict, false otherwise
6	**dict.items() :** Returns a list of dict's (key, value) tuple pairs
7	**dict.keys() :** Returns list of dictionary dict's keys
8	**dict.setdefault(key, default=None) :** Similar to get(), but will set dict[key]=default if key is not already in dict
9	**dict.update(dict2) :** Adds dictionary dict2's key-values pairs to dict
10	**dict.values() :** Returns list of dictionary dict's values

Example:

```
phone_book = {"ahmet öz" : "0532 532 32 32",
"mehmet su": "0543 543 42 42",
"seda naz" : "0533 533 33 33",
"eda ala" : "0212 212 12 12"}
person = input("Please enter a name of a person: ")
if person in phone_book:
  answer = "{} adlı kişinin telefon numarası: {}"
  print(answer.format(person, phone_book [person]))
else:
      print("This name is not in this telephone book!")
```

Example:

```
names = ["ahmet", "mehmet", "fırat", "zeynep",
"selma", "abdullah", "cem"]
dict = {i: len(i) for i in names}
```

Create a dictionary from a list

File I/O

The open Function:

Example:

```
# Open a file
fo = open("foo.txt", "wb")
print ("Name of the file: ", fo.name)
print ("Closed or not : ", fo.closed)
print ("Opening mode : ", fo.mode)
```

Output:

Name of the file: foo.txt

Closed or not : False

Opening mode : wb

File I/O

grades.txt
98-86-100-54-63
54-89-78-90-85
0-95-70-69-87-55

Opening files to read:

```
my_file = open("grades.txt", "r")
first_line = my_file.readline()
grades = first_line.split('-')
print ("Grades from the first line: ", grades)
my_file.close()
```

Output:

```
Grades from the first line:  ['98', '86', '100', '54', '63\n']
```

File I/O

Opening modes

Sr.No.	Modes & Description
1	**r** Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
2	**rb** Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
3	**r+** Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
4	**rb+** Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
5	**w** Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
6	**wb** Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
7	**w+** Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
8	**wb+** Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
9	**a** Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
10	**ab** Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
11	**a+** Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
12	**ab+** Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

Opening files and reading all lines:

```
my_file = open("expenses.txt", "r")
total_expense = 0
for line in my_file.readlines():
    expenses_list = line.split('-')
    for expense in expenses_list:
        total_expense += int(expense)
print("Total expense was:", total_expense)
my_file.close()
```

expenses.txt

100-54-63
78-90-85
70-69-87-55

Output:

```
Total expense was: 751
```

Example:

```
file = open("input.txt","r")
for aline in file.readlines():
    list = aline.split(':')
    print("name:",list[0],"phone number:",list[1])
file.close()
```

input.txt

Ahmet Özbudak : 0533 123 23 34
Mehmet Sülün : 0532 212 22 22
Sami Sam : 0542 333 34 34

Output:

```
name: Ahmet Özbudak  phone number:  0533 123 23 34
name: Mehmet Sülün  phone number:  0532 212 22 22
name: Sami Sam  phone number:  0542 333 34 34
```

Opening files to write (print output):

```
my_file = open("output.txt", "w")
my_file.write("I am writing this output to a file")
my_file.close()
```

Output: The sentence “I am writing this output to a file” will be written into a file named **`output.txt`**

New function:
`f.write(string)`

writes the contents of *string* to the file, returning the number of characters written.

Opening files to write (print output) cont.:

```
my_file = open("myage.txt", "w")
my_age = 20
my_file.write("I am " + str(my_age) + " years old.")
my_file.close()
```

Output: The sentence “I am 20 years old.” will be written into a file named **`myage.txt`**

`file.write(string)` takes only one argument, so you need to change any other types into strings and concatenate (+) all parts before passing them as an argument.

Exercise

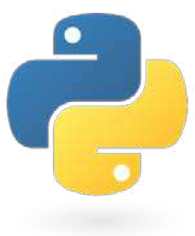
- Write a program that reads an input file **`grades.txt`** which stores student names and their grades separated by a colon (:), prints out the name of the student with the highest grade, the name of the student with the lowest grade, and the average grade for this class. Your program should also write the same output to an output file named **`class_stats.txt`**
- **Note: use a dictionary to store the information from grades.txt**

grades.txt

```
Ahmet Özbudak:87
Mehmet Sülün:99
Sami Sam:45
Leyla Tan:93
Emre Göz:32
```

Hacettepe University

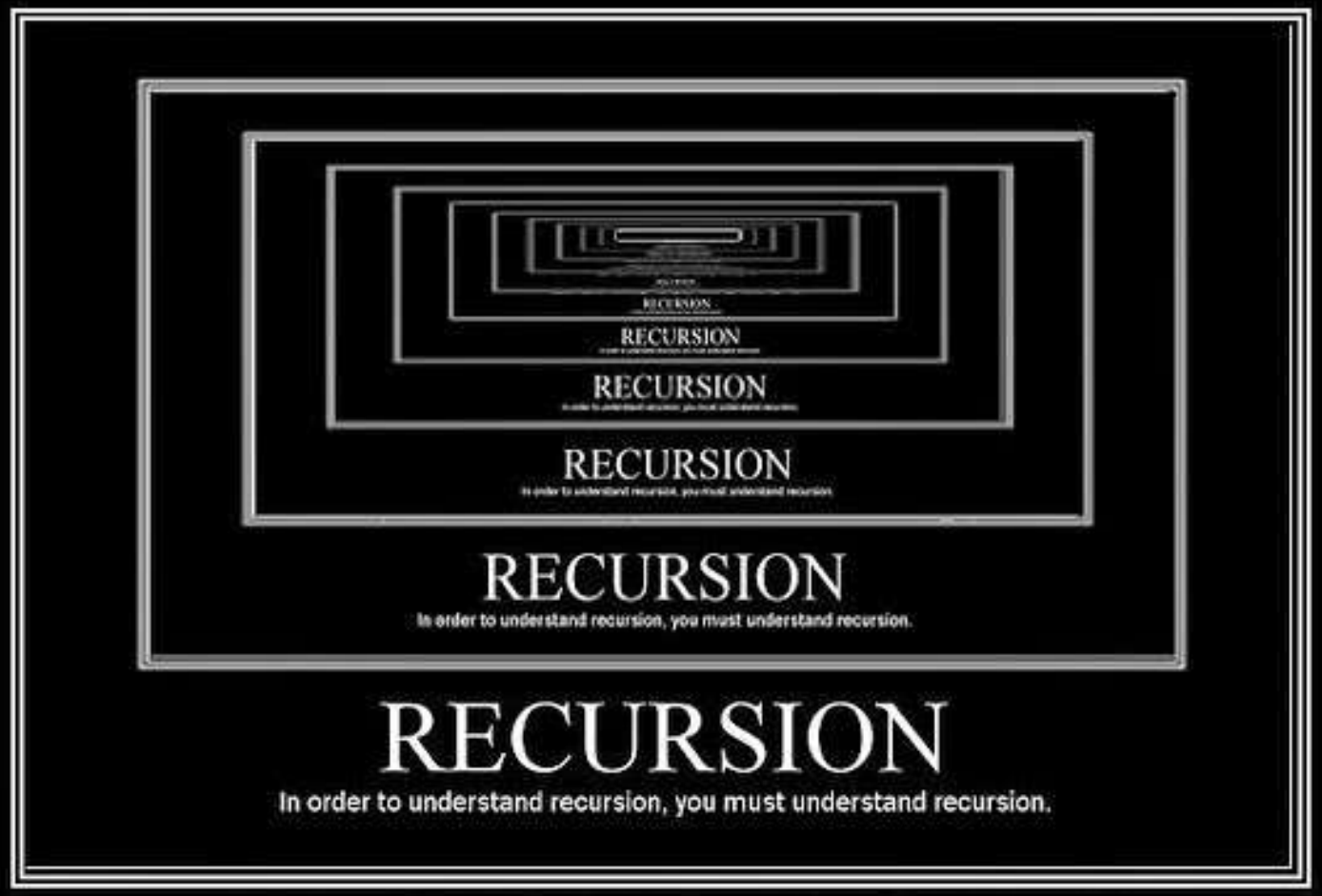
Computer Engineering Department

Programming in  python

BBM103 Introduction to Programming Lab 1

Week 8

Fall 2017



RECURSION

In order to understand recursion, you must understand recursion.

WHAT IS RECURSION?

- **Goal**: simplify the problem by solving the same problem for smaller input
  - Solve problems by **divide(decrease)-and-conquer**
- Function calls itself (but not infinitely!)
  - One or more base cases

ITERATION vs. RECURSION

- An **ITERATIVE** function is one that loops to repeat some part of the code.

- A **RECURSIVE** function is one that calls itself again to repeat the code.

Multiplication Example: ITERATIVE Solution

a*b is equal to “add **a** to itself **b** times”

a*b = a + a + a + a + … + a

b times

```
def multiply_iterative(a, b):
    result = 0
    while b > 0:
        result += a
        b -= 1
    return result
```

Iteration

Multiplication Example: RECURSIVE Solution

$$a*b = \underbrace{a + a + a + a + \ldots + a}_{\text{b times}} = \boxed{a + a*(b-1)}$$

b-1 times

```
def mult_recursive(a, b):
    if b == 1:
        return a
    else:
        return a + mult_recursive(a, b-1)
```

Base case

Recursive Step

Factorial Example: ITERATIVE Solution

$$n! = n*(n-1)*(n-2)*(n-3)* \ldots * 1$$

```
def factorial_iterative(n):
    result = 1
    while n > 0:
        result *= n
        n -= 1
    return result
```

Iteration

Factorial Example: RECURSIVE Solution

n! = n*(n-1)*(n-2)*(n-3)* … * 1

- **Base Case:** if n = 1 → 1! = 1
- **Recursive step:** n! = n * (n-1)!

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

Base case

Recursive Step

ITERATION vs. RECURSION

- recursion may be simpler, more intuitive, and also efficient and natural for a programmer.

- BUT! Recursion may not be efficient from the computer's point of view.
  - Ex. Computing n^{th} Fibonacci number recursively takes $O(2^n)$ steps!

Example: Fibonacci Numbers

The Fibonacci numbers are the numbers of the following sequence of integer values:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
The Fibonacci numbers are defined by:
$F_n = F_{n-1} + F_{n-2}$
with $F_0 = 0$ and $F_1 = 1$

```
def fibonacci(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a
number=input("Please enter a number to print fibonacci numbers!")
print(fibonacci(int(number)))
```

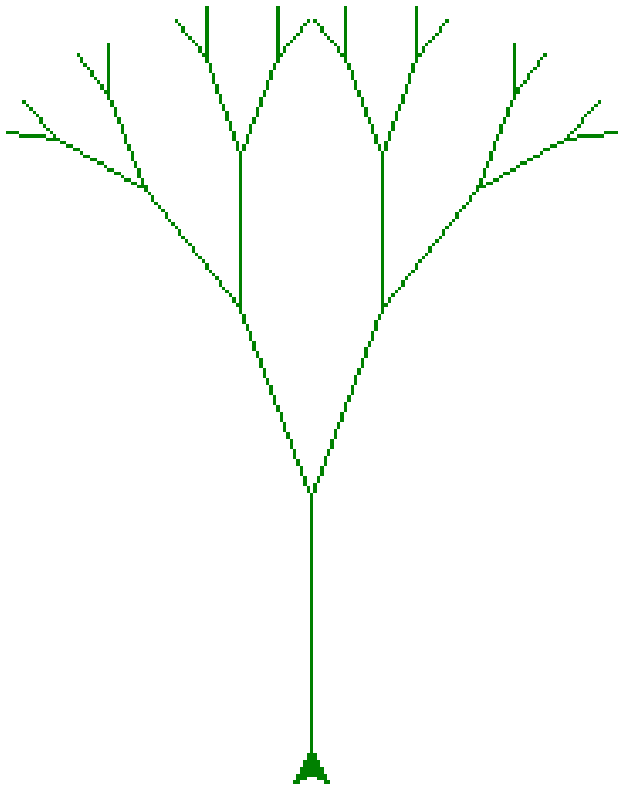
Output:

```
Please enter a number to print fibonacci numbers!4
3
```

Example: Visualizing Recursion

```
34 import turtle
35
36 def tree(branchLen,t):
37     if branchLen > 5:
38         t.forward(branchLen)
39         t.right(20)
40         tree(branchLen-15,t)
41         t.left(40)
42         tree(branchLen-15,t)
43         t.right(20)
44         t.backward(branchLen)
45
46 def main():
47     t = turtle.Turtle()
48     myWin = turtle.Screen()
49     t.left(90)
50     t.up()
51     t.backward(100)
52     t.down()
53     t.color("green")
54     tree(75,t)
55     myWin.exitonclick()
56
57 main()
```

Output:



Example: Computing Exponent

```
 9 def exp(x, n):
10     """
11     Computes the result of x raised to the power of n.
12 
13         >>> exp(2, 3)
14         8
15         >>> exp(3, 2)
16         9
17     """
18     if n == 0:
19         return 1
20     else:
21         return x * exp(x, n-1)
22 
23 number1=input("print a number as base")
24 number2=input("print a number as exponent")
25 print(exp(int(number1),int(number2)))
```

Lets look at the execution pattern.

```
exp(2, 4)
+-- 2 * exp(2, 3)
|       +-- 2 * exp(2, 2)
|       |       +-- 2 * exp(2, 1)
|       |       |       +-- 2 * exp(2, 0)
|       |       |       |       +-- 1
|       |       |       +-- 2 * 1
|       |       |       +-- 2
|       |       +-- 2 * 2
|       |       +-- 4
|       +-- 2 * 4
|       +-- 8
+-- 2 * 8
+-- 16
```

We can compute exponent in fewer steps if we use successive squaring.

```
25 def fast_exp(x, n):
26     if n == 0:
27         return 1
28     elif n % 2 == 0:
29         return fast_exp(x*x, n/2)
30     else:
31         return x * fast_exp(x, n-1)
32 
33 number1=input("print a number as base")
34 number2=input("print a number as exponent")
35 print(fast_exp(int(number1),int(number2)))
36 
```

Lets look at the execution pattern now.

```
fast_exp(2, 10)
+-- fast_exp(4, 5) # 2 * 2
|   +-- 4 * fast_exp(4, 4)
|   |       +-- fast_exp(16, 2) # 4 * 4
|   |       |   +-- fast_exp(256, 1) # 16 * 16
|   |       |   |   +-- 256 * fast_exp(256, 0)
|   |       |   |               +-- 1
|   |       |   |   +-- 256 * 1
|   |       |   |   +-- 256
|   |       |   +-- 256
|   |       +-- 256
|   +-- 4 * 256
|   +-- 1024
+-- 1024
1024
```

Example: Flatten a List

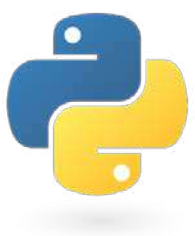
```
39 def flatten_list(a, result=None):
40     if result is None:
41         result = []
42 
43     for x in a:
44         if isinstance(x, list):
45             flatten_list(x, result)
46         else:
47             result.append(x)
48 
49     return result
50 listToFlat=[ [1, 2, [3, 4] ], [5, 6], 7]
51 print(listToFlat)
52 faltList=flatten_list(listToFlat)
53 print(faltList)
```

Output:

```
[[1, 2, [3, 4]], [5, 6], 7]
[1, 2, 3, 4, 5, 6, 7]
```

Hacettepe University

Computer Engineering Department

Programming in  python

BBM103 Introduction to Programming Lab 1

Week 8

Fall 2017

Lab Exercises

1. Write python programs

   a) that find **greatest element** in the list whose elements are provided as command-line arguments. (a.py)

   ['34' , '11' , '42' , '3' , '16' , '7'] -> 42

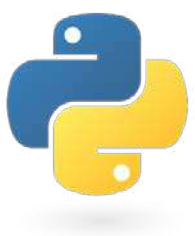
   b) that return the *level of depth* of a nested list. (b.py)

[['1' , '4' , '7'], 'a' ,['b' ,['t' ,['9' , '1' ,['u' ,['8'], '1'], '9'], '3']], 'r'] -> 5

Note: Use **recursive functions** in both programs.

Hacettepe University

Computer Engineering Department

Programming in  python

BBM103 Introduction to Programming Lab 1

Week 9

Fall 2017

Lab Exercises

1. Write python program

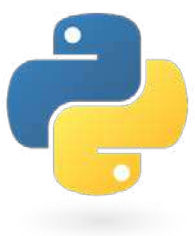
   a) that creates a dictionary such that its key value (from *1* to *n*) determines the number of stars in its value (e.g. {'3' -> ['*', '*', '*']}) than displays them in the following pattern (with n = 8)

```
*
**
***
****
*****
******
*******
********
```

Note: Use **dictionary comprehensions** in all programs.

Hacettepe University

Computer Engineering Department

Programming in  python

BBM103 Introduction to Programming Lab 1

Week 9

Fall 2017

Sorting – sorted()

- The syntax of sorted() method is:

```
sorted(iterable[, key][, reverse])
```

Parameters

- sorted() takes two three parameters:
- **iterable** - sequence (string, tuple, list) or collection (set, dictionary, frozen set) or any iterator
- **reverse (Optional)** - If true, the sorted list is reversed (or sorted in Descending order)
- **key (Optional)** - function that serves as a key for the sort comparison

Example:

```
pySet = {'e', 'a', 'u', 'o', 'i'}
print(sorted(pySet, reverse=True))

# dictionary
pyDict = {'e': 1, 'a': 2, 'u': 3, 'o': 4, 'i': 5}
print(sorted(pyDict, reverse=True))

# frozen set
pyFSet = frozenset(('e', 'a', 'u', 'o', 'i'))
print(sorted(pyFSet, reverse=True))
```

Output:

```
['u', 'o', 'i', 'e', 'a']
['u', 'o', 'i', 'e', 'a']
['u', 'o', 'i', 'e', 'a']
```

Example:

```
def takeSecond(elem):
    return elem[1]

random = [(2, 2), (3, 4), (4, 1), (1, 3)]

sortedList = sorted(random, key=takeSecond)

print('Sorted list:', sortedList)
```

Output:

```
Sorted list: [(4, 1), (2, 2), (1, 3), (3, 4)]
```

Sorting – operator.itemgetter()

```
from operator import itemgetter


lis = [{ "name" : "Nandini", "age" : 20},
{ "name" : "Manjeet", "age" : 20 },
{ "name" : "Nikhil" , "age" : 19 }]

print ("The list printed sorting by age: ")
print (sorted(lis, key=itemgetter('age')))

print ("The list printed sorting by age and name: ")
print (sorted(lis, key=itemgetter('age', 'name')))

print ("The list printed sorting by age in descending order: ")
print (sorted(lis, key=itemgetter('age'),reverse = True))
```

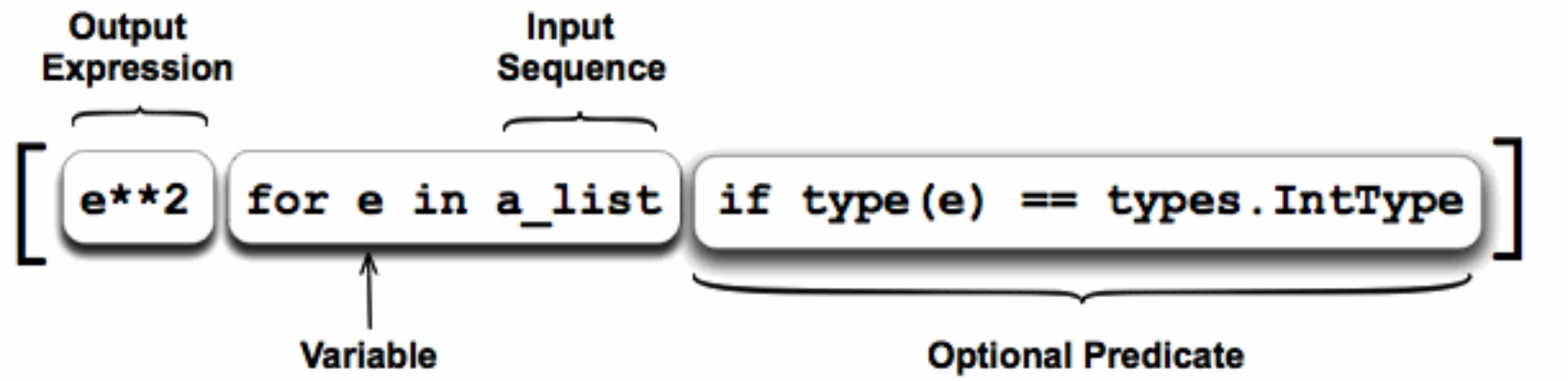
Output:

```
The list printed sorting by age:
[{'name': 'Nikhil', 'age': 19}, {'name': 'Nandini', 'age': 20}, {'name': 'Manjeet', 'age': 20}]

The list printed sorting by age and name:
[{'name': 'Nikhil', 'age': 19}, {'name': 'Manjeet', 'age': 20}, {'name': 'Nandini', 'age': 20}]

The list printed sorting by age in descending order:
[{'name': 'Nandini', 'age': 20}, {'name': 'Manjeet', 'age': 20}, {'name': 'Nikhil', 'age': 19}]
```

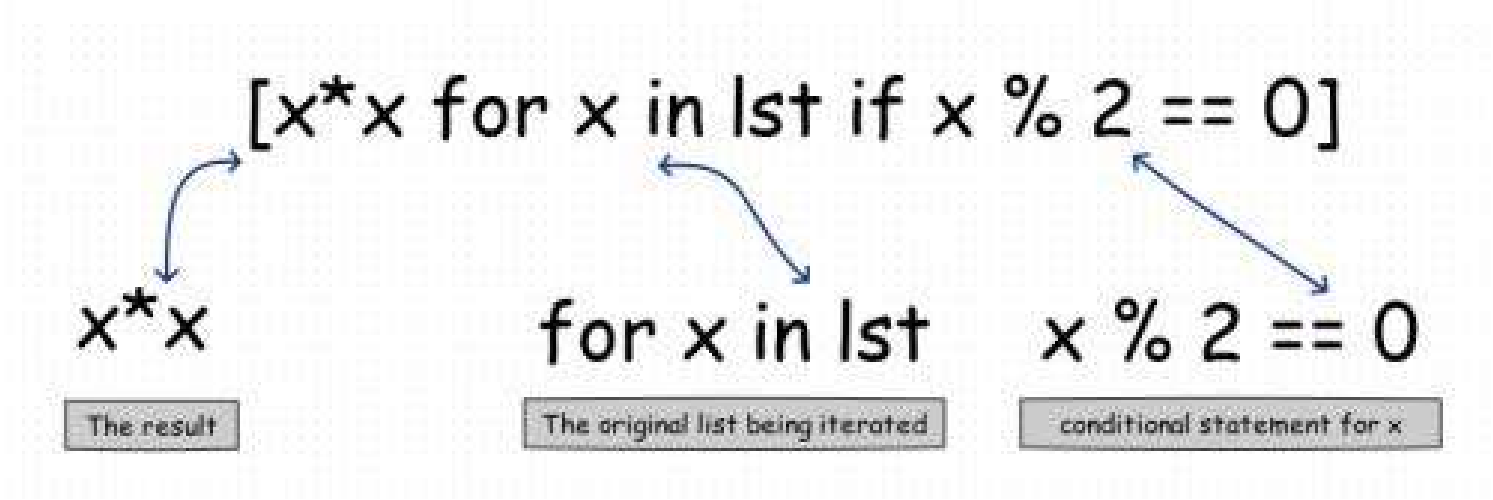
Python Comprehensions

- Python comprehensions are syntactic constructs that enable sequences to be built from other sequences in a clear and concise manner. Python comprehensions are of three types namely:
  - ❑ list comprehensions,
  - ❑ set comprehensions and
  - ❑ dict comprehensions.

Comprehensions



Example:



List Comprehensions

- List comprehensions provide a concise way to create a new list of elements that satisfies a given condition from an **iterable**. An **iterable** is any python construct that can be looped over.

Example: for loop

```
squares = []
for x in range(10):
    squares.append(x**2)
print(squares)
```

list comprehension

```
squares = [x**2 for x in range(10)]
print(squares)
```

Output:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Output:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Example:

```
even_squares = [i**2 for i in range(10) if i % 2 == 0]
print(even_squares)
```

Output:

```
[0, 4, 16, 36, 64]
```

Example:

```
S = [x**2 for x in range(10)]
V = [2**i for i in range(13)]
M = [x for x in S if x % 2 == 0]

print(S)
print(V)
print(M)
```

Output:

```
S: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
V: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]
M: [0, 4, 16, 36, 64]
```

Nested *for* loops in List Comprehensions

- List comprehensions can also be used with multiple or nested *for* loops.

Example: nested for loops

```
combs = []
for x in [1,2,3]:
    for y in [3,1,4]:
        if x != y:
            combs.append((x, y))

print(combs)
```

list comprehension

```
combs1=[(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
print(combs1)
```

Output:

```
[(1, 3), (1, 4), (2, 3), (2, 1),
(2, 4), (3, 1), (3, 4)]
```

Output:

```
[(1, 3), (1, 4), (2, 3), (2, 1),
(2, 4), (3, 1), (3, 4)]
```

Set Comprehensions

- In set comprehensions, we use the braces rather than square brackets.

Example:

```
x = {i**2 for i in range(10)}
print(type(x))
print(x)
```

Output:

```
<class 'set'>
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
```

Dict Comprehensions

Example:

```
x = {i:i**2 for i in range(10)}
print(type(x))
print(x)
```

Output:

```
<class 'dict'>
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

Example:

```
noprimes = [j for i in range(2, 8) for j in range(i*2, 50, i)]
primes = [x for x in range(2, 50) if x not in noprimes]

print (noprimes)
print (primes)
```

Output:

```
Noprimes: [4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30,
 32,34, 36,38, 40, 42, 44, 46, 48, 6, 9, 12, 15, 18, 21, 24,27,
 30,33,36, 39, 42, 45, 48, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44,
 48, 10,15,20, 25, 30, 35, 40, 45, 12, 18, 24, 30, 36, 42, 48,14,
 21,28,35, 42, 49]
Primes[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

Example:

```
words = 'The quick brown fox jumps over the lazy dog'.split()
print (words)
stuff = [[w.upper(), w.lower(), len(w)] for w in words]
for i in stuff:
    print(i)
```

Output:

```
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
['THE', 'the', 3]
['QUICK', 'quick', 5]
['BROWN', 'brown', 5]
['FOX', 'fox', 3]
['JUMPS', 'jumps', 5]
['OVER', 'over', 4]
['THE', 'the', 3]
['LAZY', 'lazy', 4]
['DOG', 'dog', 3]
```

Example:

```
def zip(lst1, lst2):
    """
    Made an assumption both lst1 and lst2 will have the same length.
    Used the range function to get the position the item so that we can use the position
    as the index key for both list.
    """
    return [(lst1[i], lst2[i]) for i in range(len(lst1))]

print(zip([1, 2, 3], ["a", "b", "c"]))
```

Output:

```
[(1, 'a'), (2, 'b'), (3, 'c')]
```

Example:

```
non_flat = [ [1,2,3], [4,5,6], [7,8] ]
list=[y for x in non_flat for y in x]
print(list)
```

Output:

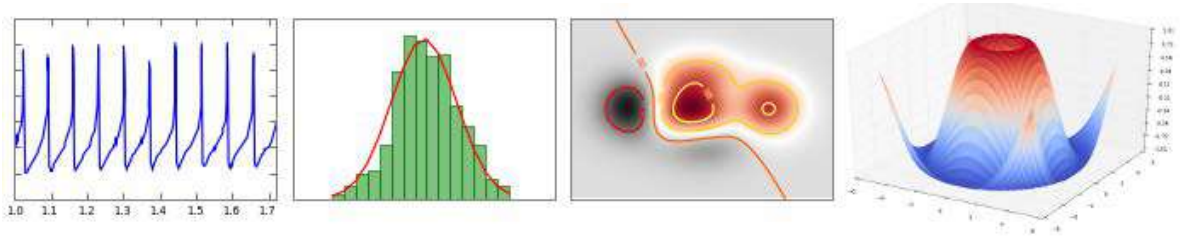
```
[1, 2, 3, 4, 5, 6, 7, 8]
```

Example:

```
def map(func, lst):
    """
    This was pretty simple following the basic formula.
    Since we can pass functions around as an argument, the map function
    receives the the function to be applied.  The function is then applied to
    each item in the list.
    """
    return [func(i) for i in lst]

def square(x):
    return x * x

assert map(square, range(5)) == [0,1,4,9,16]
```

2D Data Plotting in Python: matplotlib

- **matplotlib** is a Python 2D plotting library
- You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc.



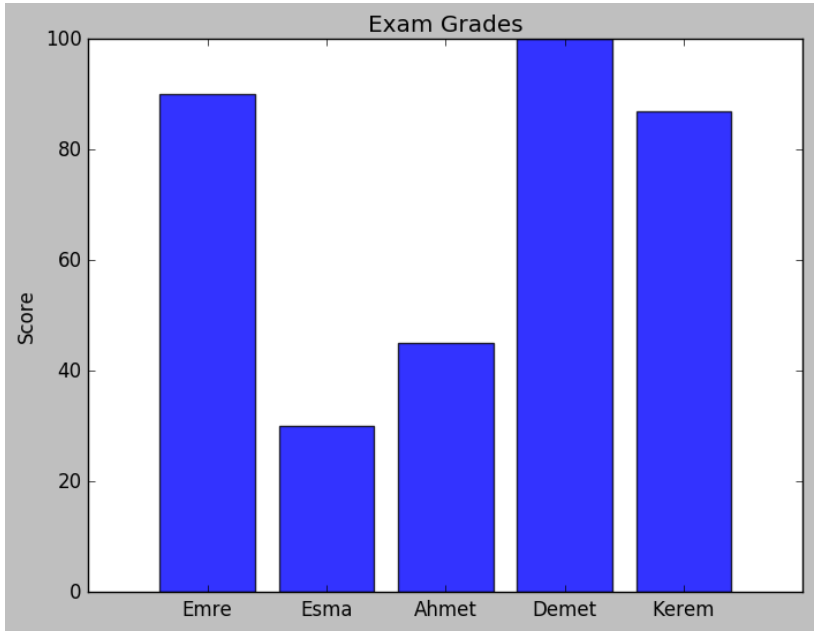
- Installing matplotlib: http://matplotlib.org/users/installing.html
- matplotlib in PyCharm: https://www.jetbrains.com/help/pycharm/2016.1/matplotlib-support.html
- Or use Anaconda that provides numerous built-in Python packages including matplotlib: https://www.continuum.io/downloads

Vertical Bar Chart Plotting

- **Example:**

```
1 import matplotlib.pyplot as plot
2
3 students = ['Emre', 'Esma', 'Ahmet', 'Demet', 'Kerem']
4 grades = [90,30,45,100,87]
5 x_pos = [x for x in range(len(students))]
6
7 plot.bar(x_pos, grades, align='center', color='b', alpha=0.8)
8 plot.xticks(x_pos, students)
9 plot.ylabel('Score')
10 plot.title('Exam Grades')
11
12 plot.show()
```

- **Output:**

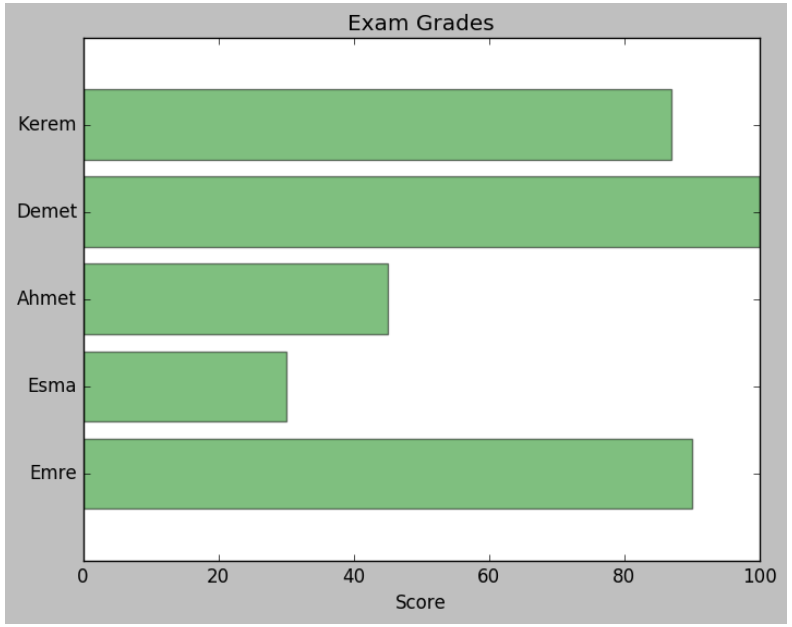


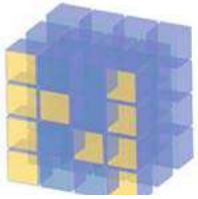
Horizontal Bar Chart Plotting

- **Example:**

```
1 import matplotlib.pyplot as plot
2
3 students = ['Emre', 'Esma', 'Ahmet', 'Demet', 'Kerem']
4 grades = [90,30,45,100,87]
5 y_pos = [x for x in range(len(students))]
6
7 plot.barh(y_pos, grades, align='center', color='g', alpha=0.5)
8 plot.yticks(y_pos, students)
9 plot.xlabel('Score')
10 plot.title('Exam Grades')
11
12 plot.show()
```

- **Output:**





NumPy - scientific computing with Python

- **NumPy** (http://www.numpy.org) is the fundamental package for scientific computing with Python. It supports among other things:
  - a powerful N-dimensional array object,
  - sophisticated (broadcasting) functions,
  - useful linear algebra, Fourier transform, and random number capabilities,
  - efficient multi-dimensional container of generic data,
  - arbitrary data-types.

- Installing Packages in PyCharm (search for numpy): https://www.jetbrains.com/help/pycharm/2016.1/installing-uninstalling-and-upgrading-packages.html

- Or use Anaconda that provides numerous built-in Python packages including NumPy: https://www.continuum.io/downloads

A simple plot with a custom dashed line

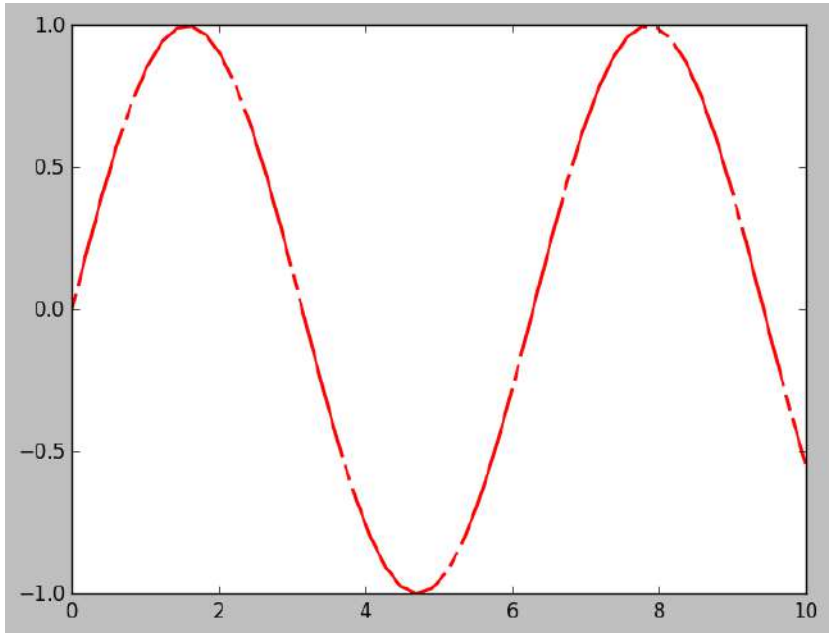
- **Example:**

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.linspace(0, 10)
5
6 line, = plt.plot(x, np.sin(x), '--', linewidth=2, color="r")
7
8 dashes = [10, 5, 100, 5]  # 10 points on, 5 off, 100 on, 5 off
9 line.set_dashes(dashes)
10
11 plt.show()
```

New function: `numpy.linspace(start, stop)`

Returns evenly spaced numbers over a specified interval [*start*, *stop*].

- **Output:**



A simple plot of fill function

- **Example:**

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 x = np.linspace(0, 1)
5 y = np.sin(4 * np.pi * x) * np.exp(-5 * x)
6
7 plt.fill(x, y, 'y')
8 plt.grid(True)
9 plt.show()
```

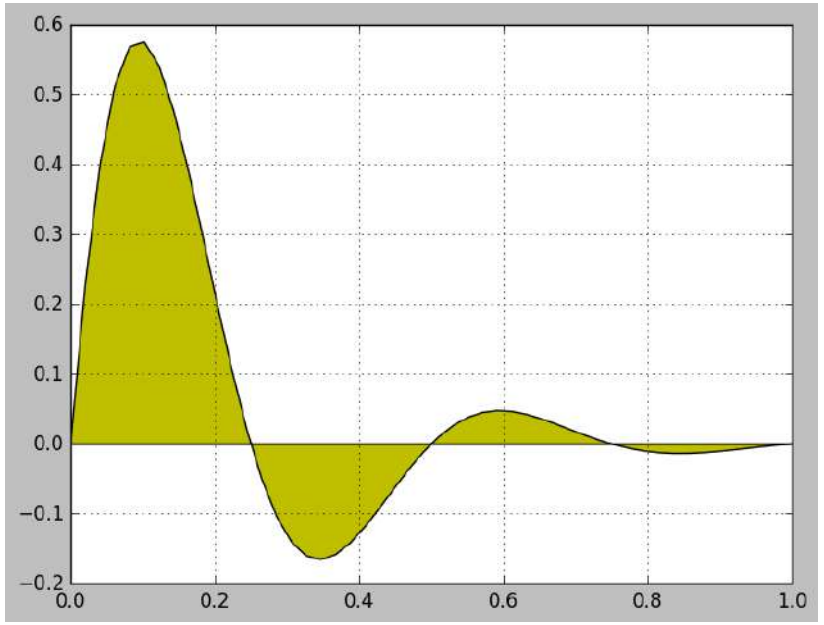
New functions:

`numpy.sin()`- Trigonometric sine, element-wise

`numpy.exp()`- Calculate the exponential of all elements in the input array

`numpy.pi()` - π mathematical constant

- **Output:**



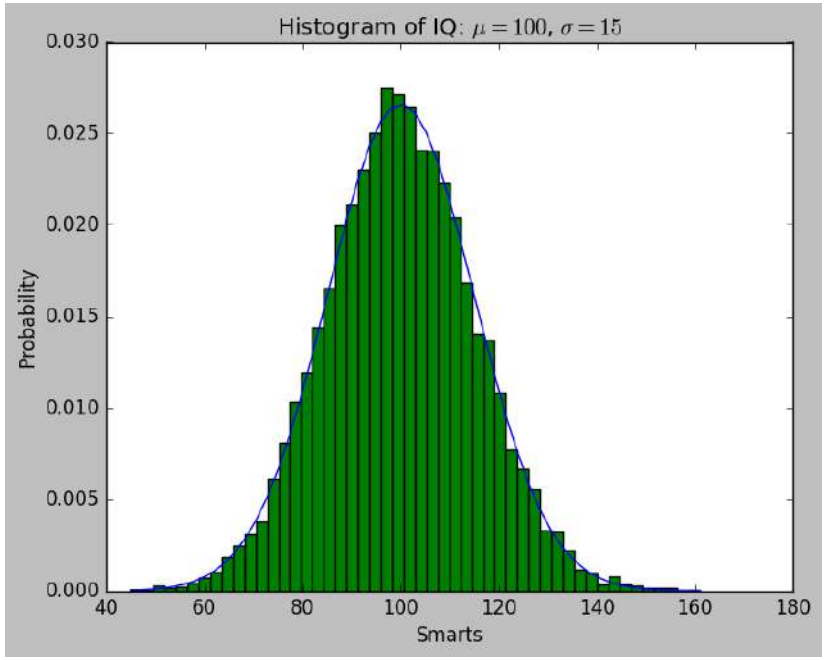
Histogram Plotting

A *histogram* is a graphical representation of the distribution of numerical data.

- **Example:**

```
1 import numpy as np
2 import matplotlib.mlab as mlab
3 import matplotlib.pyplot as plt
4
5 mu = 100  # mean of distribution
6 sigma = 15  # standard deviation of distribution
7 x = mu + sigma * np.random.randn(10000)
8
9 num_bins = 50
10 # the histogram of the data
11 n, bins, patches = plt.hist(x, num_bins, normed=1, facecolor='green')
12 # add a 'best fit' line
13 y = mlab.normpdf(bins, mu, sigma)
14 plt.plot(bins, y, 'b-')
15 plt.xlabel('Smarts')
16 plt.ylabel('Probability')
17 plt.title(r'Histogram of IQ: $\mu=100$, $\sigma=15$')
18
19 # Tweak spacing to prevent clipping of ylabel
20 plt.subplots_adjust(left=0.15)
21 plt.show()
```

- **Output:**



New function:
`numpy.random.randn(dimension)`

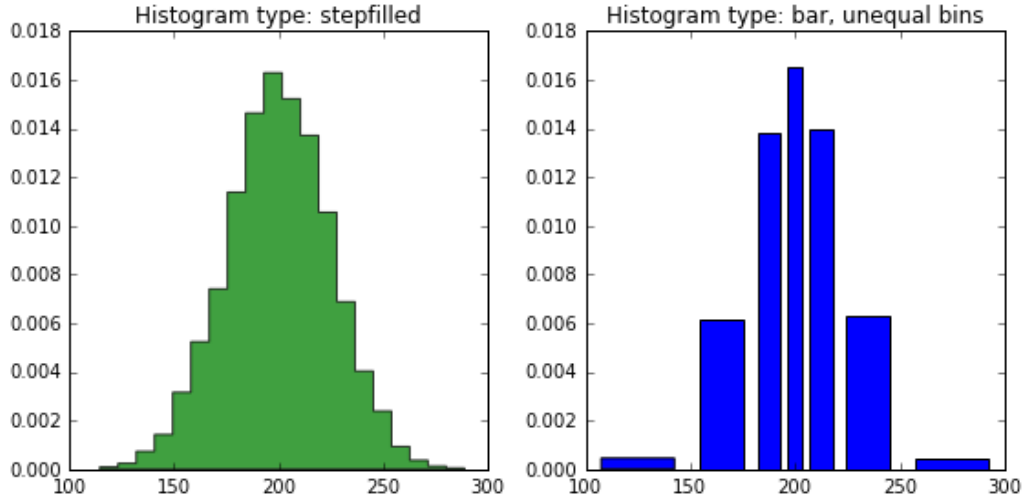
Returns a sample (or samples) from the "standard normal" distribution

Histogram Plotting Continued (Subplots)

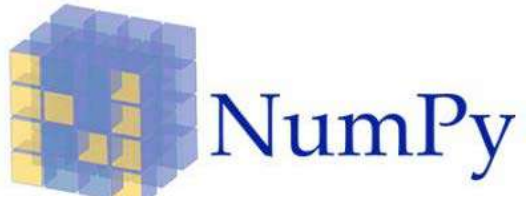
- **Example:**

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 mu = 200
5 sigma = 25
6 x = mu + sigma*np.random.randn(10000)
7 print(x)
8 fig, (ax0, ax1) = plt.subplots(ncols=2, figsize=(8, 4))
9
10 ax0.hist(x, 20, normed=1, histtype='stepfilled', facecolor='g', alpha=0.75)
11 ax0.set_title('Histogram type: stepfilled')
12
13 #Create a histogram by providing the bin edges (unequally spaced).
14 bins = [100, 150, 180, 195, 205, 220, 250, 300]
15 ax1.hist(x, bins, normed=1, histtype='bar', rwidth=0.7)
16 ax1.set_title('Histogram type: bar, unequal bins')
17
18 plt.tight_layout()
19 plt.show()
```

- **Output:**



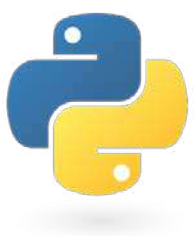
2D Plotting and Scientific Computing in Python



- **For more matplotlib examples:** http://matplotlib.org/examples/index.html
- **Plotting Commands Summary:** http://matplotlib.org/api/pyplot_summary.html
- **NumPy Manual:** https://docs.scipy.org/doc/numpy/index.html

Hacettepe University

Computer Engineering Department

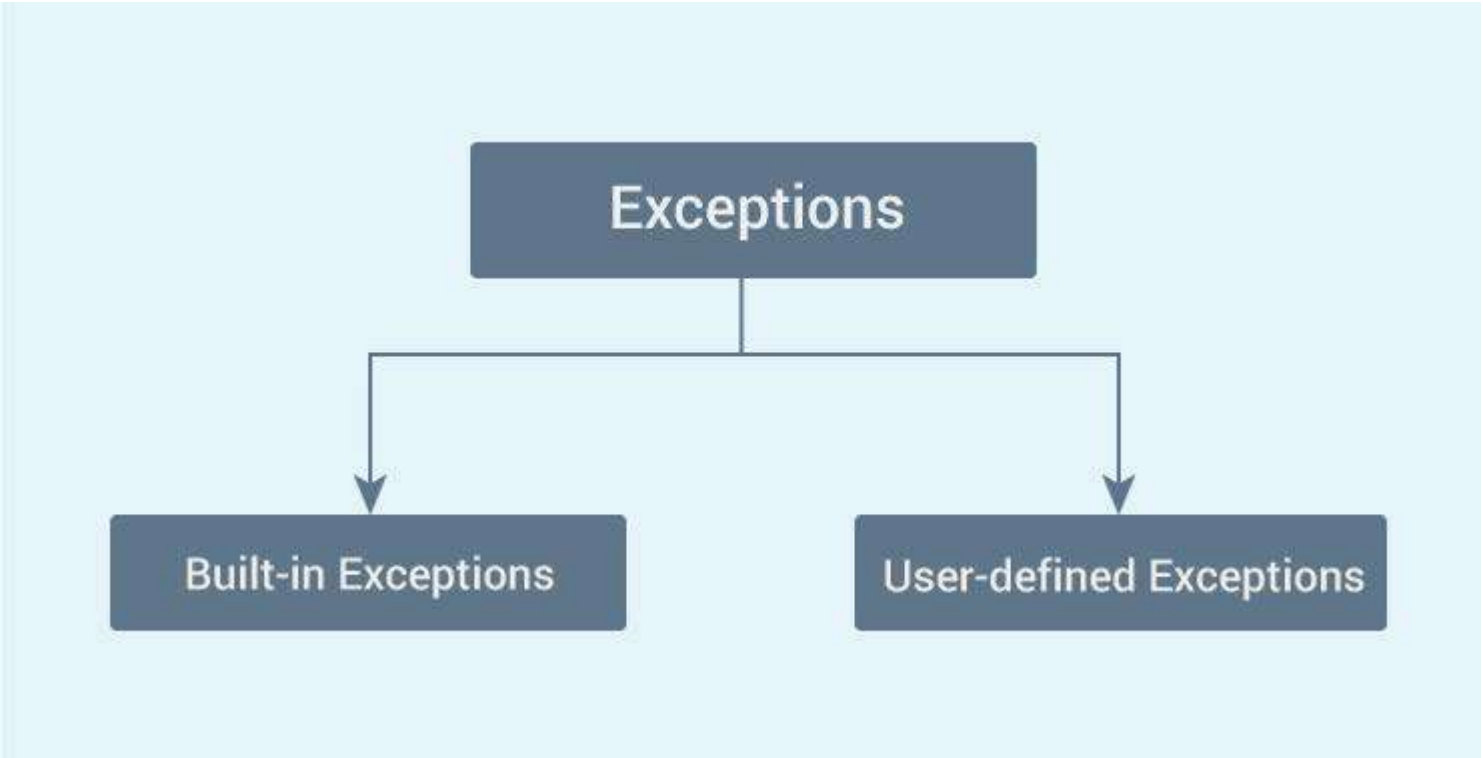
Programming in  python

BBM103 Introduction to Programming Lab 1

Week 10

Fall 2017

Exceptions



Built-in Exceptions

The simplest way to handle exceptions is with a "try-except" block:

Example 1:

```
(x,y) = (5,0)
try:
    z = x/y
except ZeroDivisionError:
    print ("divide by zero")
```

Output: divide by zero

Example 2: except ValueError:

```
first_number = input("First number: ")
second_number = input("Second number: ")
try:
    number1 = int(first_number)
    number2 = int(second_number)
    print(number1, "/", number2, "=", number1 / number2)
except ValueError:
    print("Error! Please enter number!")
```

Example 3: except ZeroDivisionError:

```
first_number = input("First number: ")
second_number = input("Second number: ")
try:
    number1 = int(first_number)
    number2 = int(second_number)
    print(number1, "/", number2, "=", number1 / number2)
except ValueError:
    print("Error! Please enter number!")
except ZeroDivisionError:
    print("You can't divide a number to 0!")
```

Example 4: except (ValueError, ZeroDivisionError)

```
first_number = input("First number: ")
second_number = input("Second number: ")
try:
    number1 = int(first_number)
    number2 = int(second_number)
    print(number1, "/", number2, "=", number1 / number2)
except (ValueError,ZeroDivisionError):
    print("Error!")
```

Example 5: try... except... as...

```
first_number = input("First number: ")
second_number = input("Second number: ")
try:
    number1 = int(first_number)
    number2 = int(second_number)
    print(number1, "/", number2, "=", number1 / number2)
except (ValueError,ZeroDivisionError) as error:
    print("Error!")
    print("Original error message: ", error)
```

Example 6: try... except... else...

```
for arg in sys.argv[1:]:
    try:
        f = open(arg, 'r')
    except IOError:
        print('cannot open', arg)
    else:
        print(arg, 'has', len(f.readlines()), 'lines')
        f.close()
```

Example 7: try... except... finally...

```
try:
    file = open("dosyaadı", "r")
except IOError:
    print("error!")
finally:
    file.close()
```

Some Examples using Exceptions

```
except IOError:
        print('An error occured trying to read the file.')

except ValueError:
        print('Non-numeric data found in the file.')

 except ImportError:
        print ("NO module found«)

except EOFError:
        print('Why did you do an EOF on me?')

except KeyboardInterrupt:
         print('You cancelled the operation.')

except:
        print('An error occured.')
```

raise

Example 8:

```
tr_character = "şçğüöıİ"

password = input("Enter your password: ")

for i in password:
    if i in tr_character:
        raise TypeError("Yo can't use Turkish characters in password!")
    else:
        pass

print("Password is excepted!")
```

Example 9:

```
try:
    while True:
        if int(input('Guess a number: ')) == 5
            raise ZeroDivisionError
except ZeroDivisionError:
    print ('You got it!')
```

Example 10:

```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except OSError as err:
    print("OS error: {0}".format(err))
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error:", sys.exc_info()[0])
    raise
```

User-Defined Exceptions

Example 11:

```
class MyException(Exception):
    def __init__(self, t=0):
        self.numtries = t
try:
    for tries in range(1, 6):
        if int(input('Guess a number: ')) == 5:
            raise MyException(tries)
except MyException as e:
    print ('You got it in only %d tries!'  % e.numtries)
else:
    print ('Too bad, you ran out of tries!')
```

Example 12 user-defined exceptions

```
class Error(Exception):
    """Base class for other exceptions"""
    pass

class ValueTooSmallError(Error):
    """Raised when the input value is too small"""
    pass

class ValueTooLargeError(Error):
    """Raised when the input value is too large"""
    pass

# our main program
# user guesses a number until he/she gets it right

# you need to guess this number
number = 10
```

This example continues in the next slide ⟶

Example 12 continued

```
while True:
    try:
        i_num = int(input("Enter a number: "))
        if i_num < number:
            raise ValueTooSmallError
        elif i_num > number:
            raise ValueTooLargeError
        break
    except ValueTooSmallError:
        print("This value is too small, try again!")
        print()
    except ValueTooLargeError:
        print("This value is too large, try again!")
        print()

print("Congratulations! You guessed it correctly.")
```

Assert

```
assert <some_test>, <message>
```

Example 13:

```
def test_set_comparison():
    set1 = set("1308")
    set2 = set("8035")
    assert set1 == set2

test_set_comparison()
```

Output:

```
C:\Users\necva\Desktop>py deneme.py
Traceback (most recent call last):
  File "deneme.py", line 8, in <module>
    test_set_comparison()
  File "deneme.py", line 4, in test_set_comparison
    assert set1 == set2
AssertionError
```

Example 14:

```
array = [0,1, 2, 3, 4, 5, 6, 7, 8, 9]

def number(input):
        assert (input in array)


number(10)
number(5)
```

Output:

```
C:\Users\necva\Desktop>py deneme.py
Traceback (most recent call last):
  File "deneme.py", line 7, in <module>
    number(10)
  File "deneme.py", line 4, in number
    assert (input in array)
AssertionError
```

Example 15:

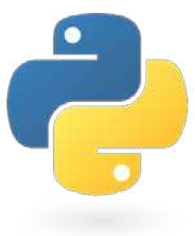
```
def func (a,b):
    max= 0
    if a < b: max= b
    if b < a: max= a
    print(max)
    assert (max == a or max == b) and max >= a and max >= b

func(10,15)
```

Output:

```
C:\Users\necva\Desktop>py deneme.py
15
```

Hacettepe University

Computer Engineering Department

Programming in  python

BBM103 Introduction to Programming Lab 1

Week 10

Fall 2017

Lab Exercise

- Write a Python program `myExercise.py` that reads input file named `student`.
- Every line of the file contains student records (*name*, *University*, and *department*).

student <File Name>

```
Emre:Hacettepe University,Computer Engineering
Kerem:METU,Architecture
Leyla:Ankara University,Physics
Sami:Bilkent University,Civil Engineering
```

- Your program should print the student name and record from the file for each name provided as command-line argument (seperated by commas).
- If a student name is not found in the student records, your program should **throw an exception** and print an error message as stated below.

```
python myExercise.py Emre,Ahmet
```

output

```
Name: Emre, University: Hacettepe University,Computer Engineering
No record of 'Ahmet' was found!
```

Hacettepe University

Computer Engineering Department

Programming in  python

BBM103 Introduction to Programming Lab 1

Week 11

Fall 2017

Debugging

- ***Debugging*** is the process of identifying and removing errors that prevent correct operation of computer software or a system.

- PyCharm provides a full range of facilities for debugging your source code:
  - Breakpoints in Python.
  - Customizable breakpoint properties: conditions, pass count, etc.
  - Frames, variables, and watches views in the debugger UI.
  - Runtime evaluation of expressions.

- For detailed explanation of the debugging process in PyCharm: https://www.jetbrains.com/help/pycharm/2016.1/debugging.html

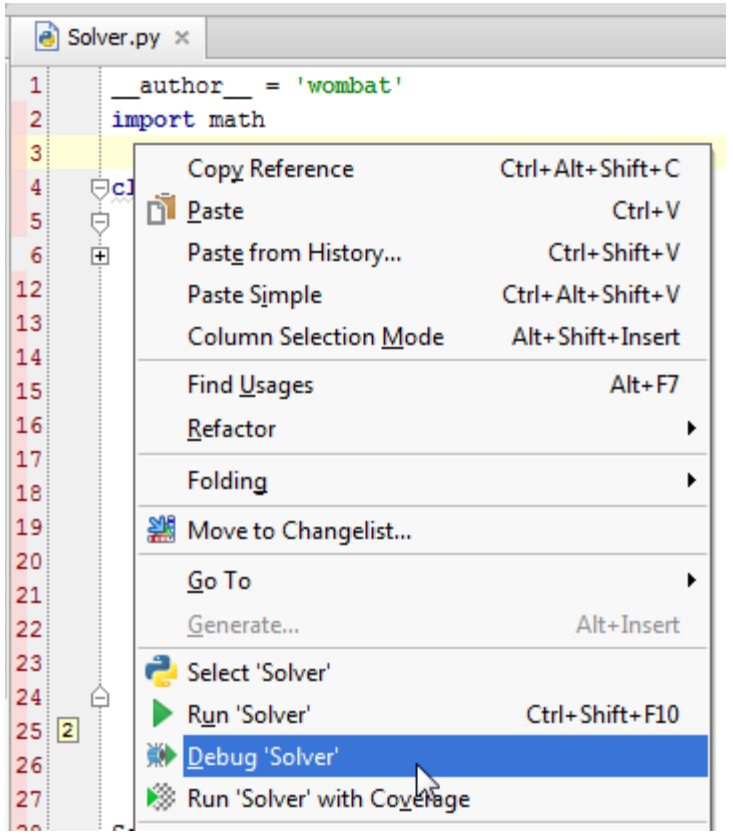
Debugging Cont.

- **General debugging steps:**

1. Configure the debugger options.
2. Define a run/debug configuration.
3. Create breakpoints in the source code.
4. Launch a debugging session.
5. Pause or resume the debugging session as required.
6. During the debugger session, step through the breakpoints, evaluate expressions, change values on-the-fly , examine suspended program, explore frames, and set watches .

Starting the Debugger Session

- Set breakpoints in the source code.
- Open the desired Python script in the editor, or select it in the Project tool window.
- On the context menu, choose Debug <script name>:



PyCharm Debug Tool Window

View | Tool Windows | Debug

Alt+5

- The Debug tool window becomes available when you start debugging.
- It displays the output generated by the debugging session for your application.
- For Toolbars and Items descriptions: https://www.jetbrains.com/help/pycharm/2016.1/debug-tool-window.html

Lab Exercise - 1

- Could you write a function that tells us that "120" is "5!"?
- **Hint:** The strategy is pretty straightforward, just divide the term by successively larger terms until you get to "1" as the result: (use recursion)

```
120 -> 120/2 -> 60/3 -> 20/4 -> 5/5 -> 1 => 5!
```

- **python3.5 myExercise1.py 120,150**

Output

```
120 = 5!
150 NONE
```

Use recursion

Lab Exercise - 2

- An anagram is a form of word play, where you take a word (or set of words) and form a different word (or different set of words) that use the same letters, just rearranged.
- **Hint :** All words must be valid spelling, and shuffling words around doesn't count.
- Write a function that takes an input and checks if two words in each line are anagrams.

input.txt

```
Clint Eastwood ? Old West Action
parliament ? partial man
```

python3.5 myExercise2.py input.txt

Output:

```
Clint Eastwood is an anagram of Old West Action
parliament is NOT an anagram of partial man
```

Hacettepe University

Computer Engineering Department

Programming in

&

BBM103 Introduction to Programming Lab 1

Week 12

Fall 2017

First Program with C

```c
#include <stdio.h>
int main()
{
    printf("Hello World");
    return 0;
}
```

The **#include <stdio.h>** is a preprocessor command.

This command tells compiler to include the contents of stdio.h (standard input and output) file in the program.

The stdio.h file contains functions such as scanf() and print() to take input and display output respectively.

- The execution of a C program starts from the main() function
- The printf() is a library function to send formatted output to the screen. In this program, the printf() displays *Hello, World!* text on the screen.
- The return 0; statement is the "Exit status" of the program. In simple terms, program ends with this statement.

printf() and scanf()

```c
#include <stdio.h>
int main()
{
    int number;


    printf("Enter a number to scan it on the screen: ");

    // scanf() reads the formatted input and stores them
    scanf("%d", &number);

    // printf() displays the formatted output
    printf("You entered: %d", number);
    return 0;
}
```

scanf()

the scanf() function reads an integer data from the user and stores in variable *number*.

Basic Mathematical Operations

```c
#include <stdio.h>

int main()
{
    int first, second, add, subtract, multiply;
    float divide;

    printf("Enter two integers\n");
    scanf("%d%d", &first, &second);

    add = first + second;
    subtract = first - second;
    multiply = first * second;
    divide = first / (float)second;    //typecasting

    printf("Sum = %d\n",add);
    printf("Difference = %d\n",subtract);
    printf("Multiplication = %d\n",multiply);
    printf("Division = %.2f\n",divide);

    return 0;
}
```

In c, as a general rule
integer/integer = integer
float/integer = float
integer/float = float.

So we convert denominator to float in program.
This explicit conversion is known as typecasting.

printf()

Format Specifiers

%i or %d	int
%c	char
%f	float (see also the note below)
%s	string

Format Specifiers

Commonly used escape sequences are:

\n :newline
\t :tab
\v :vertical tab
\f :new page
\b :backspace
\r :carriage return
\n :newline

Example: Formatting

%d :print as a decimal integer
%6d :print as a decimal integer with a width of at least 6 wide
%f :print as a floating point
%4f :print as a floating point with a width of at least 4 wide
%.4f :print as a floating point with a precision of four characters after the decimal point
%3.2f :print as a floating point at least 3 wide and a precision of 2

Example: Formatting

```c
#include<stdio.h>

main()
{
    printf("The color: %s\n", "blue");
    printf("First number: %d\n", 12345);
    printf("Second number: %04d\n", 25);
    printf("Third number: %i\n", 1234);
    printf("Float number: %3.2f\n", 3.14159);
    printf("Hexadecimal: %x\n", 255);
    printf("Octal: %o\n", 255);
    printf("Unsigned value: %u\n", 150);
    printf("Just print the percentage sign %%\n", 10);
}
```

Output:

```
The color: blue
First number: 12345
Second number: 0025
Third number: 1234
Float number: 3.14
Hexadecimal: ff
Octal: 377
Unsigned value: 150
Just print the percentage sign %
```

Arrays in C

Defining array

```
int examplearray[100]; /* This declares an array */
```

Example: Array

```c
#include <stdio.h>

int main()
{
  int x;
  int y;
  int array[8][8]; /* Declares an array like a chessboard */

  for ( x = 0; x < 8; x++ ) {
    for ( y = 0; y < 8; y++ )
      array[x][y] = x * y; /* Set each element to a value */
  }
  printf( "Array Indices:\n" );
  for ( x = 0; x < 8;x++ ) {
    for ( y = 0; y < 8; y++ )
    {
        printf( "[%d][%d]=%d", x, y, array[x][y] );
    }
    printf( "\n" );
  }
  getchar();
}
```

Output:

```
Array Indices:

[0][0]=0
[0][1]=0
[0][2]=0
[0][3]=0
[0][4]=0
[0][5]=0
[0][6]=0
[0][7]=0
[1][0]=0
[1][1]=1
[1][2]=2
[1][3]=3
[1][4]=4
[1][5]=5
[1][6]=6
[1][7]=7
[2][0]=0
[2][1]=2
[2][2]=4
[2][3]=6
[2][4]=8
[2][5]=10
[2][6]=12
[2][7]=14
[3][0]=0
[3][1]=3
[3][2]=6
[3][3]=9
[3][4]=12
[3][5]=15
[3][6]=18
[3][7]=21
[4][0]=0
[4][1]=4
[4][2]=8
[4][3]=12
[4][4]=16
[4][5]=20
[4][6]=24
[4][7]=28
[5][0]=0
[5][1]=5
[5][2]=10
[5][3]=15
[5][4]=20
[5][5]=25
[5][6]=30
[5][7]=35
[6][0]=0
[6][1]=6
[6][2]=12
[6][3]=18
[6][4]=24
[6][5]=30
[6][6]=36
[6][7]=42
[7][0]=0
[7][1]=7
[7][2]=14
[7][3]=21
[7][4]=28
[7][5]=35
[7][6]=42
[7][7]=49
```

Headers in C

```c
#include <stdio.h>
#include <math.h>

int main()
{
    int number;

    printf("Enter an Integer\n");
    scanf("%d", number);
    printf("Square of %d is %d\n", number, sqrt(number));

    return 0;
}
```

Constants in C

Example program using const keyword in C:

```c
#include <stdio.h>
int main()
{

const int height = 100; /*int constant*/
const float number = 3.14; /*Real constant*/
const char letter = 'A'; /*char constant*/
const char letter_sequence[10] = "ABC"; /*string constant*/
const char backslash_char = '\?'; /*special char cnst*/
printf("value of height :%d \n", height );
printf("value of number : %f \n", number );
printf("value of letter : %c \n", letter );
printf("value of letter_sequence : %s \n", letter_sequence);
printf("value of backslash_char : %c \n", backslash_char);

}
```

Output:

```
value of height :100
value of number : 3.140000
value of letter : A
value of letter_sequence :
ABC
value of backslash_char : ?
```

Example program using #define preprocessor directive in C:

```c
#include <stdio.h>
#define height 100
#define number 3.14
#define letter 'A'
#define letter_sequence "ABC"
#define backslash_char '\?'

int main()
{
printf("value of height : %d \n", height );
printf("value of number : %f \n", number );
printf("value of letter : %c \n", letter );
printf("value of letter_sequence : %s \n",letter_sequence);
printf("value of backslash_char : %c \n",backslash_char);
}
```

Output:

```
value of height :100
value of number : 3.140000
value of letter : A
value of letter_sequence :
ABC
value of backslash_char : ?
```

Hacettepe University

Computer Engineering Department

Programming in

&

BBM103 Introduction to Programming Lab 1

Week 12

Fall 2017

Lab Exercise

Write your first C program `myFirstC.c` that examines every elements stored in an array* named `varArray` and returns the number of

- *different element*
- *those divisible to 5*
- *those whose sqaure root is equal to or greater than 4*

Example:

For an array of `varArray = [3, 4, 45, 61, 55, 4, 99, 18, 61, 75]`

```
different element: 8
those divisible to 5: 3
those whose sqaure root is equal to or greater than 4: 7
```

```
compile: gcc myFirstC.c –o myFirstC.o -lm
run    : ./myFirstC.o
```

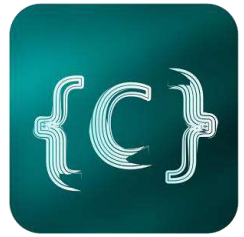
* Only 10 elements are allowed to be stored each of which should be provided from user.

Hacettepe University

Computer Engineering Department

Programming in

&



BBM103 Introduction to Programming Lab 1

Week 13

Fall 2017

C - Command Line Arguments

The command line arguments are handled using main() function arguments:

- **argc** refers to the number of arguments passed,
- **argv[]** is a pointer array which points to each argument passed to the program

Example :

```c
#include <stdio.h>

int main( int argc, char *argv[] )  {

   if( argc == 2 ) {
      printf("The argument supplied is %s\n", argv[1]);
   }
   else if( argc > 2 ) {
      printf("Too many arguments supplied.\n");
   }
   else {
      printf("One argument expected.\n");
   }
}
```

- When the above code is compiled and executed with single argument

```
./a.out testing
```

Output:

```
The argument supplied is testing
```

Example :

```
./a.out "testing1 testing2"

Programm name ./a.out
The argument supplied is testing1 testing2
```

```c
#include <stdio.h>

int main( int argc, char *argv[] )  {

   printf("Program name %s\n", argv[0]);

   if( argc == 2 ) {
      printf("The argument supplied is %s\n", argv[1]);
   }
   else if( argc > 2 ) {
      printf("Too many arguments supplied.\n");
   }
   else {
      printf("One argument expected.\n");
   }
}
```

C - Functions

A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.

Defining a Function

- The general form of a function definition in C programming language is as follows:

```
return_type function_name( parameter list ) {
   body of the function
}
```

Example:

```
/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

Function Declarations

- A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

```
return_type function_name( parameter list );
```

```
int max(int num1, int num2);
```

Calling a Function – Example :

```c
#include <stdio.h>

/* function declaration */
int max(int num1, int num2);

int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;

   /* calling a function to get max value */
   ret = max(a, b);

   printf( "Max value is : %d\n", ret );

   return 0;
}
```

```c
/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

Output:

```
Max value is : 200
```