Group-34

Group Members:
Aryan Saluja(180143)
Mota Jitendra(180434)
Siripuram ahul(180766)
Yuvraj(180899)

Source Language: Python
Target Language : MIPS Assembly
Implementation Language : C++

1. Print(I/O function)
2. Basics
3. Identifiers
4. Variables
5. Data Types
6. Input
7. Operators
8. If-else
9. Loops
10. Lists
11. Strings
12. Tuples
13. Functions

1. Print(I/O function)

Syntax:

```
print("string")
```

Description:

The function prints the string inside quotes(single or double) to output file.
It can also be used to print variables except that no enclosing quotes required.
Multiple strings and variables can be used to print a sentence in this way:

```
print("Jupiter","has", x,"moons")
```

Where x is an integer variable, it can be a string as well.
Also we can append multiple strings using a plus symbol and output an arithmetic expression value as follows:

```
print("a"+"b")
print((x*y)/z)
```

2. Statements, Comments and Keywords

Statements in python ends with an ‘end of line’ character.

Comments are made using the hash symbol for single line comments

```
#This comment is ignored by compiler
```

Multiline comments are made as follows:

```
'''This is a
multiline comment'''
```

Keywords:

```
and break continue else for if not
or return while true false none
```

3. Identifiers

An Identifier is a name given to variables, functions and classes etc.
Example: a = 10 ; here a is an identifier for integer 10.
Rules for writing identifiers in Python:

1) These identifiers can be combination of lowercase(a - z) and uppercase(A - Z) letters , digits (0 -9) or an underscore (_).
   Example: min_num, Max1.
2) These identifiers shouldn”t contain special characters (!, @, #, $, %, &, etc.).
   Example: num# , a% are invalid identifiers.
3) An Identifier shouldn’t start with a number.
   Example: 1a, 2min are invalid identifiers.
4) Identifiers are case sensitive and keywords can’t be used as an Identifier.
   Example: a and A are two different identifiers , if, else etc. those types of keywords can’t be used as identifiers.

4. Variables

In python, variables are initialized using equal symbol:

```
var = 1
```

Initialized variables can be re-assigned a different datatype by using equal symbol.

Two variables can swap their values or reassigned a new value as follows

```
x = 10
y = 20
x,y = y,x
```

In python a variable is similar to a pointer in C. A variable points to a memory location. Thus using assigning an new variable with existing variable will not create a new memory but two ponters pointing to same location or data.

Following function is used to get the address value that a variable points to:

```
id(var_name)
```

5. Data types

Since a variable can be assigned any value, following function is used to check the data type of a variable

```
type(variable)
```

Different data types:

1. Numbers:As of now, we have just included integers and not float or double.
2. Boolean: keywords `false` and `true` are used.
3. Strings
4. List
5. Tuple

6. Input

Scanning and storing a user input is done in following way:

```
variable = input()
```

Different data types:

1. Numbers:As of now, we have just included integers and not float or double.
2. Boolean: keywords `false` and `true` are used.
3. Strings
4. List
5. Tuple

7. Operators

Symbols used to perform various operations on variables are called Operators.

Different type of Operators:

- Arithmetic Operators
- Logical Operators
- Assignment Operators
- Identity Operators

1)Arithmetic Operators: These Operators take numerical values as operands and return numerical value.

+	Adds operands
-	Subtracts right operand from left operand
●	Multiplies both operands
/	Quotient of division of left operand by right operand
//	Quotient of division of left operand by right
%	Remainder of division of left operand by right operand
* *	Left operand raised to the power of right operand

```
#Example:
x = 4
y = 2
print(“sum :”, a + b)
```

Output: sum :6
Similarly we can get difference, product etc. by using other arithmetic operands.

2)Logical Operators: In this case and, or and not are operands.

3)Assignment Operators: Assignment Operators are used to assign values from its **right side operands to its left side operands.**
Example: a+=4 ; a = a + 4

4)Identity Operators: is and is not are two Identity operands.

6. if-else statements

Syntax for if statement:

```
If condition:
    Statement
    Statement
    ...
```

We first write if condition:. This **condition** is based on the decision that we will be making. If the **condition** is True, then only the body of if is executed

All the statements after colon (:) which are indented using space are body of the if statement.
We can use space or tab to provide indentation.
a statement after if block which is not indented using tab or space will not be a part of if.

If-else statement

```
If condition:
    Statement
    Statement
    …
else:
    Statement
    Statement
    …
```

Description:

The **body of *if*** consists of all the indented statements following if, unless some unindented statement is written.The **body of *else*** consists of all the indented statements following else, unless some unindented statement is written. If the **condition** is True, then the body of if is executed, otherwise the body of else is executed.

Indentation:

Statements written inside the body of *if* or *else* must be indented equally from the left margin. It means that all the statements written inside the body of *if* must be equally spaced from the left

7. Loops

Syntax for while loop:

```
while condition:
    Statement
    Statement
    …
```

Description:

The **body of the while loop** consists of all the ***indented*** statements below while condition. While loop checks whether the **condition** is True or not. If the condition is True, the statements written in the body of the while loop are executed. Then **again the condition is checked**, and if found True again, the statements in the body of the while loop are executed again. This process continues until the condition becomes False.

Therefore, the while loop repeats the statements inside its body till its condition becomes False.

8. Lists

Creating lists :

```
mylist = [2, "Hello", True, 100, 2]
```

Accessing Elements from List: To access any element of a list, we write name_of_list[index]

Lists are mutable: Which means we can update the values in list.

Adding Elements to Python List:

```
list_name.append(var)
```

Concatenation (+) Operation- Two or more lists can be concatenated or combined using the + operator.

```
mylist1 = [1, 2, 3]
mylist2 = [4, 5]
mylist3 = mylist1+mylist2
```

Adding a Single Element to Python List at last position:

```
mylist = [1, 2, 3, 4]
mylist.append(5)
```

11. Strings

Syntax:

```
print("string")
```

Description:

A string is created by enclosing the sequence of characters within single quotes or double quotes. It is assigned to a variable using assignment operator.

```
x=53
print("Jupiter","has", x,"moons")
Output:
Jupiter has 53 moons
```

Where x is an integer value, it can be a string as well.

```
print("Harry"+"Potter")
x=2,y=5
print(x*y)
Output:
Harry Potter
10
```

Also we can append multiple strings using a plus symbol and output an arithmetic expression value as well.

12. Tuples

Syntax:

```
T = (20, ‘tommy’, 0.75, [20,30,38])
print(T)
Output:
(20, ‘tommy’, 0.75, [20,30,38])
```

Description:

A Tuple is a collection in python just like lists, the difference being that tuples are immutable i.e., they can not be changed once made.
The elements can not be added, modified or deleted in tuples. You can only access data in tuples. Only methods that can be used with tuples are count and index.

```
print(T[0], T[1], T[3])
Output:
20
Tommy
[20,30,38]
```

```
T = (1,’zoo’,0.76,1)
print(T.count(1))
Output:
2
# count() method returns the number of times a specified value appears in the tuple
```

13. Functions

Syntax:

```
def function_name(arguments):
    statement1
    statement2
    statement3
    ............
    return[expression]

# Function call
function_name(values)
```

Description:

A function is a block of code which only runs when it is called.
You can pass data, known as parameters, into a function.
A function can return data as a result.

Keyword ***def*** marks the start of the function header. An unique ***function name*** is given to identify the function (follows same rules as that of naming identifiers). ***Arguments*** are specified in parentheses (these are optional) and the ***colon (:)*** marks the end of the function header.
One or more python statements make up the ***body of function***.
The ***optional return statement*** is used to return any expression or value from the function.

Arguments *are values that we pass into the function as its input.*

Examples:

```
def first_fun(x,y):
    print('This is my first function')
    print(x*y)

first_fun(2,8)

Output:
This is my first function
16
```

```
a=10
def change_val(a):
      a=20
      return
print("Before calling function a=",a)
change_val(a)
print("After calling function a=",a)

Output:
Before calling function a=10
After calling function a=20
```