INDIAN INSTITUTE OF TECHNOLOGY MADRAS

सिद्धिर्भवति कर्मजा

IIT Madras

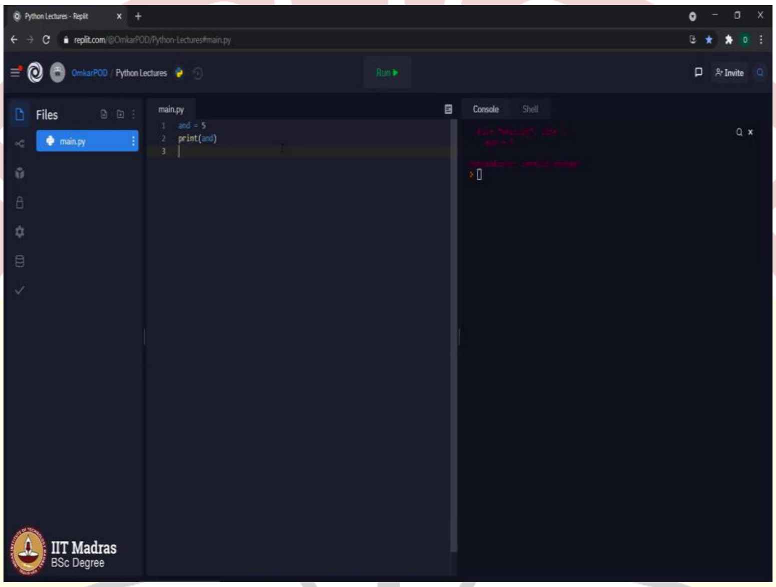
ONLINE DEGREE

Programming in Python

Professor Sudarshan Iyengar
Department of Computer Science and Engineering
Indian Institute of Technology Ropar
Omkar Joshi
Course Instructor
Indian Institute of Technology Madras Online Degree Program

More on Variables, Operators and Expressions

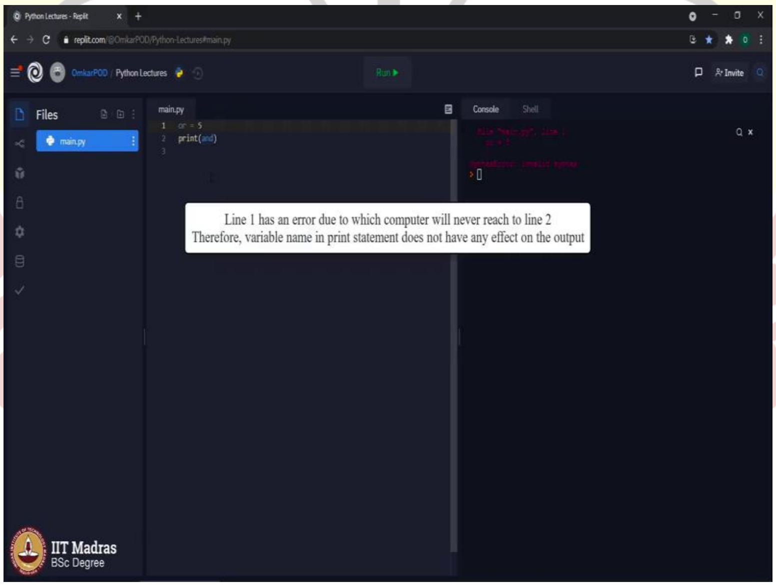
(Refer Slide Time: 00:16)



```
and = 5
print(and)
```

```
File "main.py", line 1
  and = 5

SyntaxError: invalid syntax
>
```

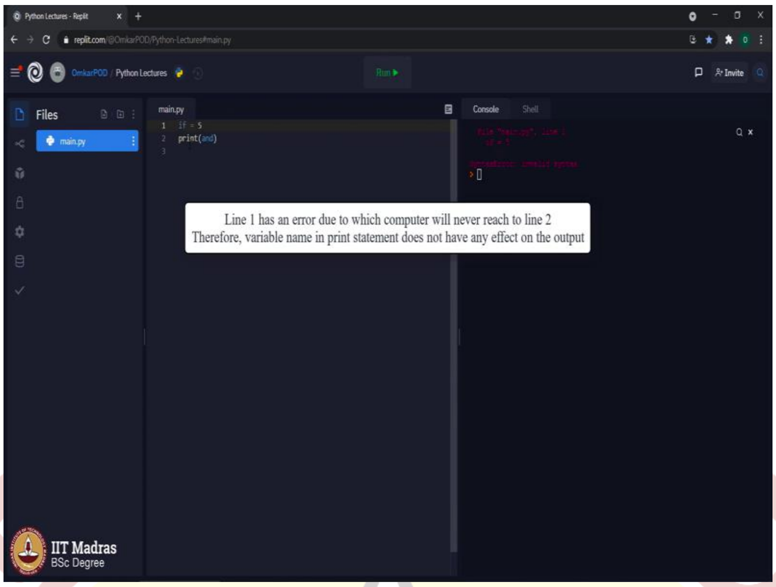


```
or = 5
print(and)
```

```
File "main.py", line 1
  or = 5

SyntaxError: invalid syntax
>
```

Line 1 has an error due to which computer will never reach to line 2
Therefore, variable name in print statement does not have any effect on the output

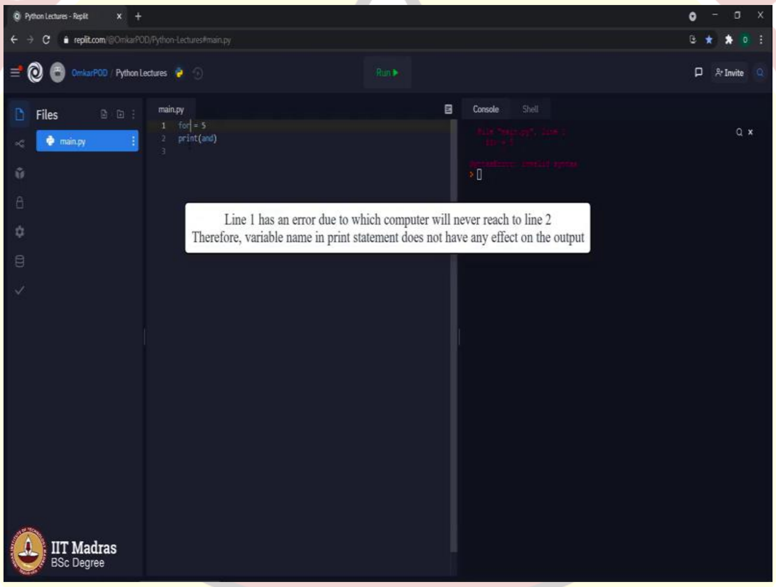


```python
if = 5
print(and)
```

```
File "main.py", line 1
    if = 5

SyntaxError: invalid syntax
>
```

Line 1 has an error due to which computer will never reach to line 2
Therefore, variable name in print statement does not have any effect on the output



```python
for = 5
print(and)
```

```
File "main.py", line 1
    for = 5

SyntaxError: invalid syntax
>
```

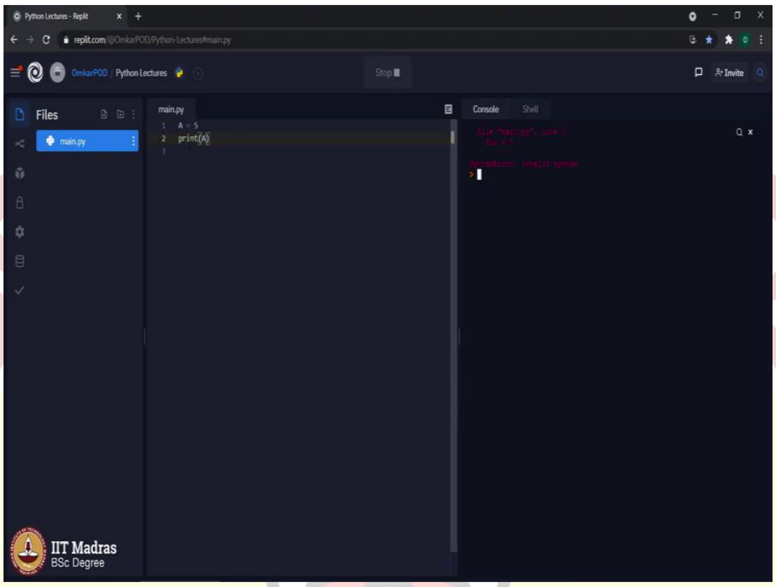
Line 1 has an error due to which computer will never reach to line 2
Therefore, variable name in print statement does not have any effect on the output

Hello Python students, earlier we have discussed about variables, operators and expressions. We will continue on the same lines and discuss few more concepts related to those same topics. So, the first concept is keywords and naming rules. We have already seen couple of operators like and, or, not, all these terms are considered as keywords. Along with these keywords, there are a few other keywords which we may not have seen yet in Python, but definitely you must be remembering those from computational thinking course like if, else, for, while and so on.

All these terms are referred as keywords as in, these terms or these words are defined as part of the programming language itself, because of that we cannot use these keywords as variable names. For example, you cannot have a variable name and is equal to 5. This will give an error; error says invalid syntax because the term and is predefined, it is a keyword in Python programming language.

Hence, it cannot be used as a variable name. Similarly, let us try or, same output. Maybe let us try if, same output; for, same output. So, all these keywords cannot be used as variable names in the Python language. There are another set of rules, which defines what can be used as a variable name.

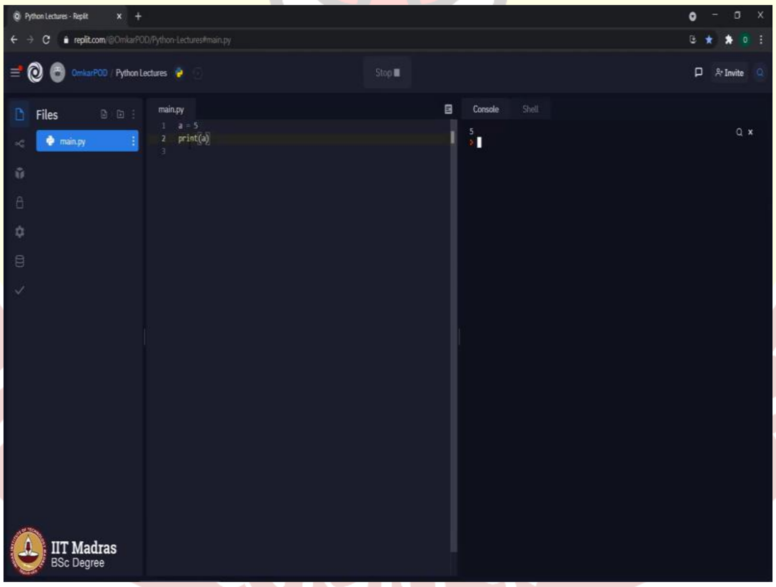
(Refer Slide Time: 02:12)



```
A = 5
print(A)
```

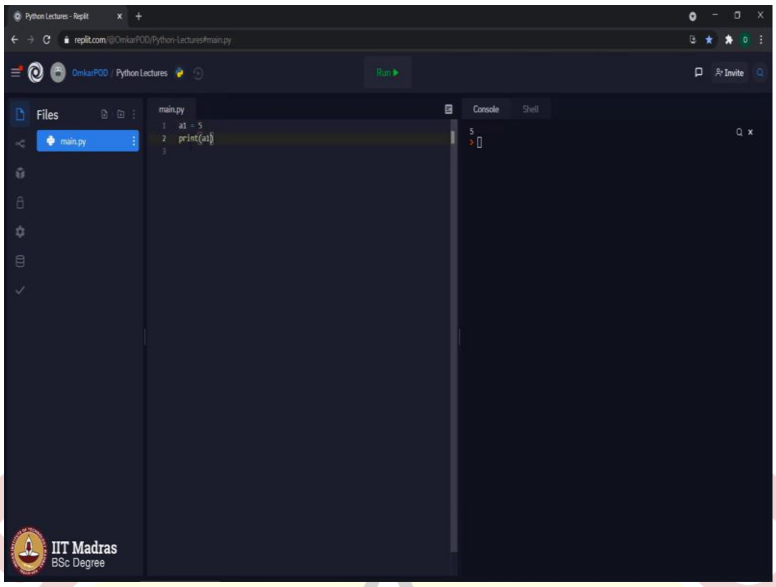
```
  File "main.py", line 1
    for = 5

SyntaxError: invalid syntax
> 
```



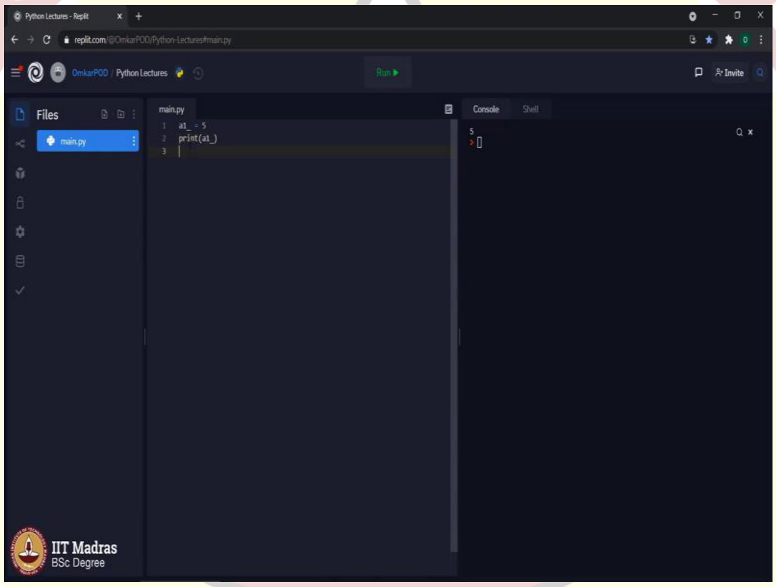
```
a = 5
print(a)
```

```
5
> 
```



```
a1 = 5
print(a1)
```

```
5
>
```



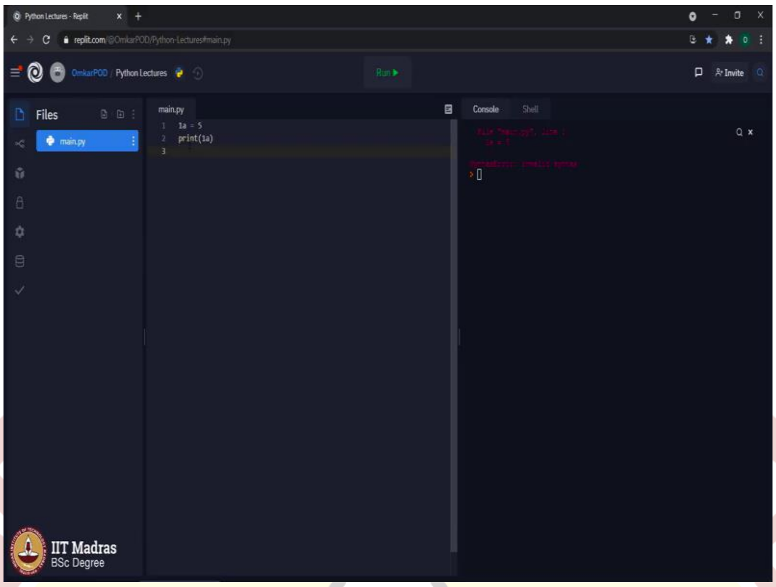
```
a1_ = 5
print(a1_)
```

```
5
>
```

The first rule says, we can use only alphanumeric characters and underscore in variable name. Alphanumeric characters include all alphabets from A to Z in lowercase, as well as uppercase and all numbers from 0 to 9. Python also supports only one special character in the variable name, which is underscore. Capital A can be a variable name. Similarly, small a can be a variable name, a1 is also a valid variable name.

As I have mentioned, we can use underscore along with it. So let us try a1_. This one is also a valid variable name, which means we can use all alphabets from A to Z in uppercase, as well as lower case, all numbers from 0 to 9 and underscore in a variable name.

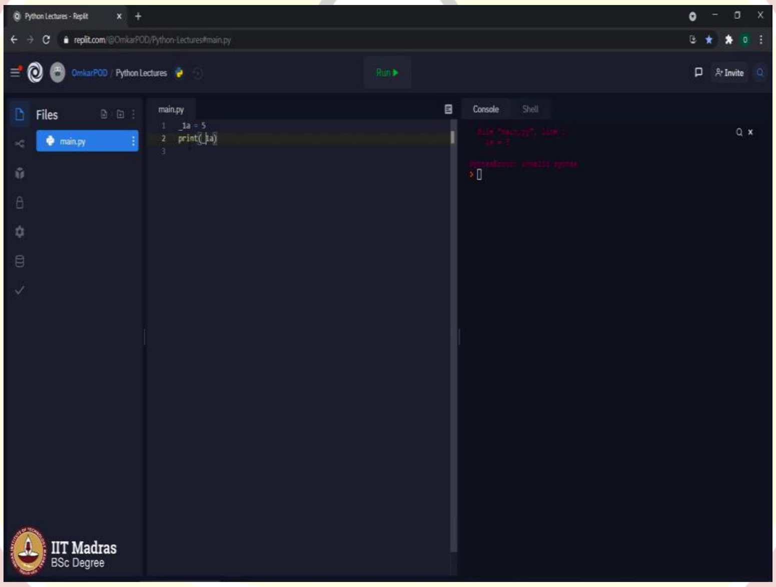
(Refer Slide Time: 03:25)



```
1a = 5
print(1a)
```

```
File "main.py", line 1
  1a = 5

SyntaxError: invalid syntax
> 
```



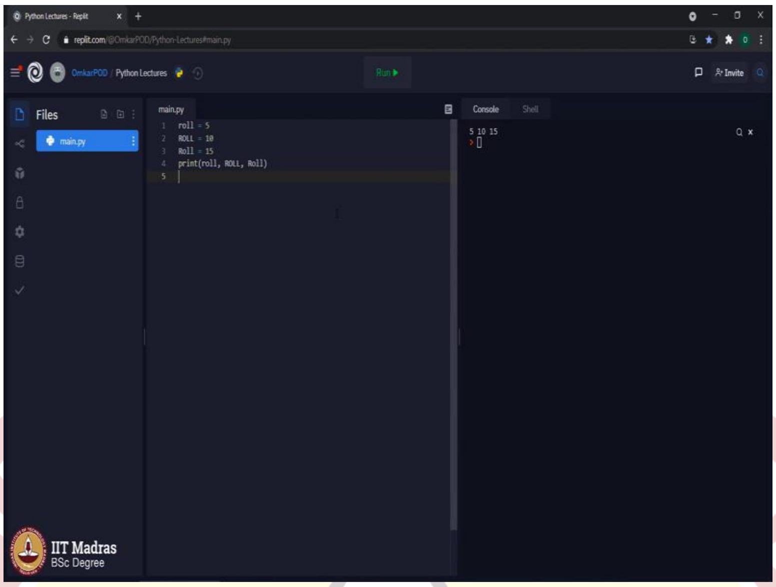
```
_1a = 5
print(_1a)
```

```
File "main.py", line 1
  1a = 5

SyntaxError: invalid syntax
> 
```

In addition to this, there is one more restriction on how a variable name has to be. And the rule says; we must start a variable name with an alphabet or an underscore, but we cannot start it with a number. Which means 1a is not a valid variable name. If we add underscore before it, then it will work as per the rule, which we discussed just now.

(Refer Slide Time: 3:54)



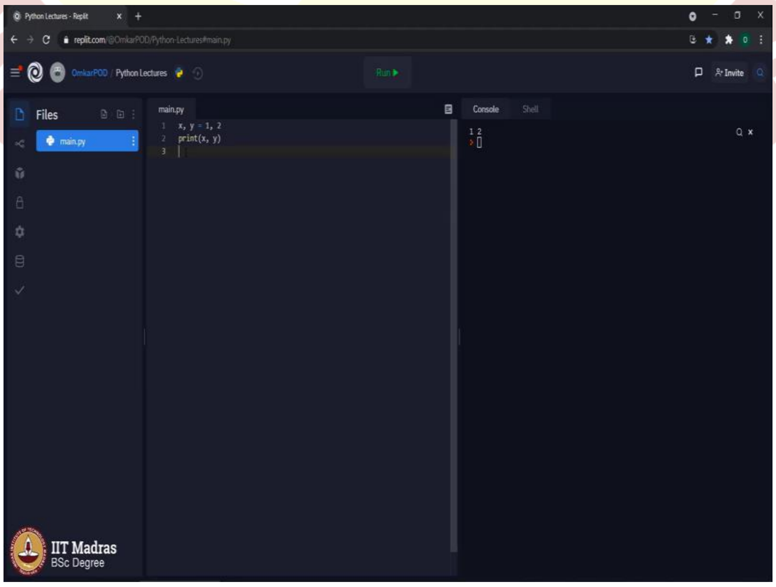
```
roll = 5
ROLL = 10
Roll = 15
print(roll, ROLL, Roll)
```

```
5 10 15
```

The last variable naming rule is variable names are case sensitive. For example, we have a variable to store a roll number; roll is equal to 5 where all characters in the variable name roll are in smaller case. In second instance, all characters are in uppercase. In third instance, only the first character is in uppercase, whereas remaining characters are in lowercase.

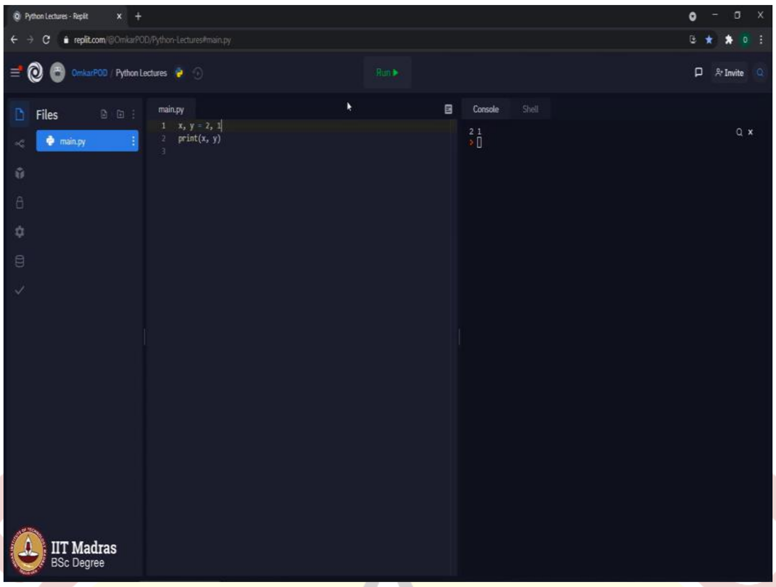
Let us see whether we are getting three different values or it is printing only 15 replacing it with the previous values. As you can see, we are getting 5, 10 and 15 because computer treat all these three variables as unique variables, even though the spelling is same, because variable names in Python are case sensitive, even a small change of uppercase letter or lowercase letter makes that variable a new and separate variable.

(Refer Slide Time: 05:03)



```
x, y = 1, 2
print(x, y)
```

```
1 2
```



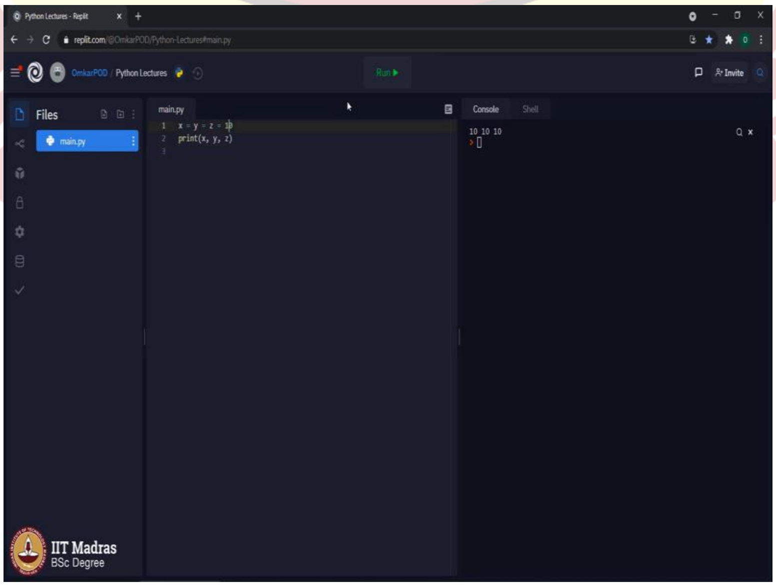
```python
x, y = 2, 1
print(x, y)
```

```
2 1
```

Next concept is multiple assignment; let us look at this particular code x comma y is equal to 1 comma 2, print x comma y and the output 1 and 2. This type of assignment of a value against a variable is called multiple assignment, where two different values are assigned two different variables in a single line. While using such multiple assignment, we have to be bit careful, because the sequence of variables or the sequence of literals on the right hand side of the equal to sign is very important.

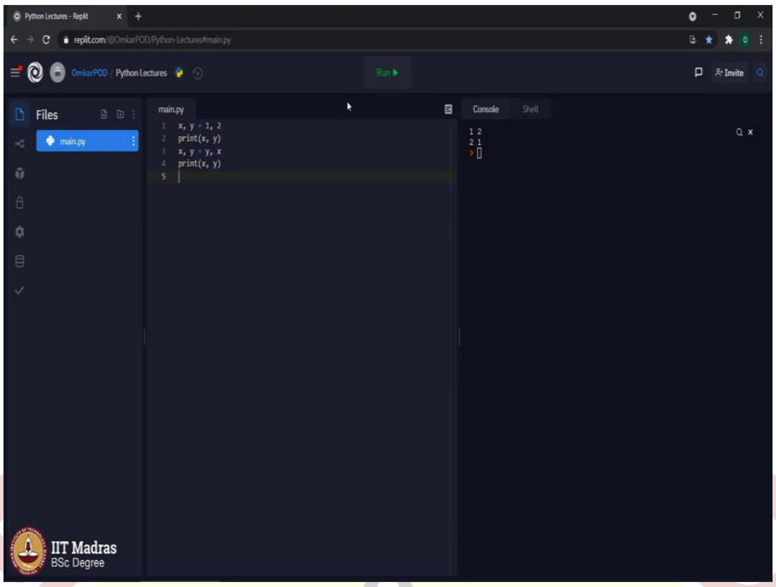
In case if we make it 2 comma 1; it will change the values of x and y. Now x is 2 and y became 1. Along with this, there is one more way in which you can assign a value to multiple variables in a single line.

(Refer Slide Time: 06:13)



```python
x = y = z = 10
print(x, y, z)
```

```
10 10 10
```



```
x, y = 1, 2
print(x, y)
x, y = y, x
print(x, y)
```

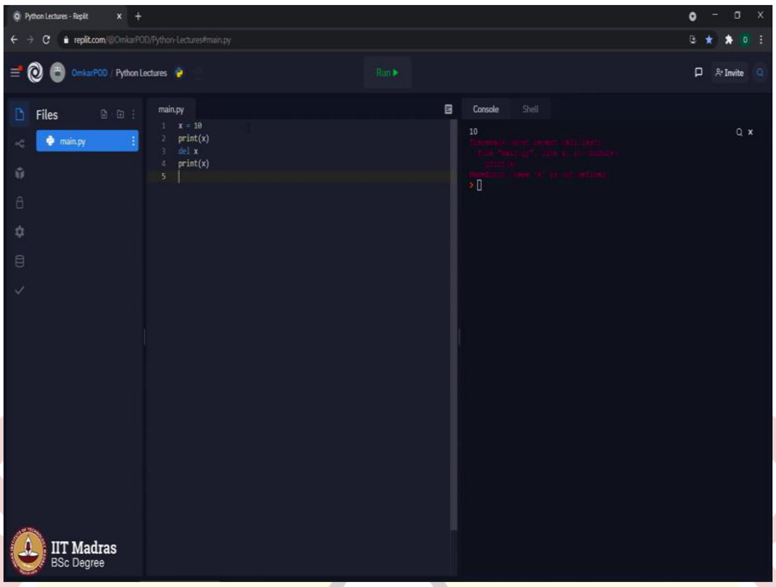
```
1 2
2 1
> 
```

x is equal to, y is equal to, z is equal to let us say 10 and print z also. It will print 10, 10 and 10 because the value 10 is assigned to all three variables x, y and z in a single statement. So far, we have only assigned a literal values to a variable but we can do similar with variables as well.

For example, x comma y is equal to 1 comma 2, print x comma y, then x comma y is equal to y comma x and then print x comma y, can you guess what will be the output? Let us try it 1, 2 and then 2, 1. Here, this kind of statement simply swaps the values of variables x and y with each other. So far, we have created so many variables, assign values to them, did lot of operations, print the values of those variables and so on.

But we never talked about how to remove a variable. As we keep on writing a Python code, we will keep creating lot of variables. So there has to be a mechanism through which we should be able to delete those variables as well and that is our next concept called deleting a variable.

(Refer Slide Time: 07:53)

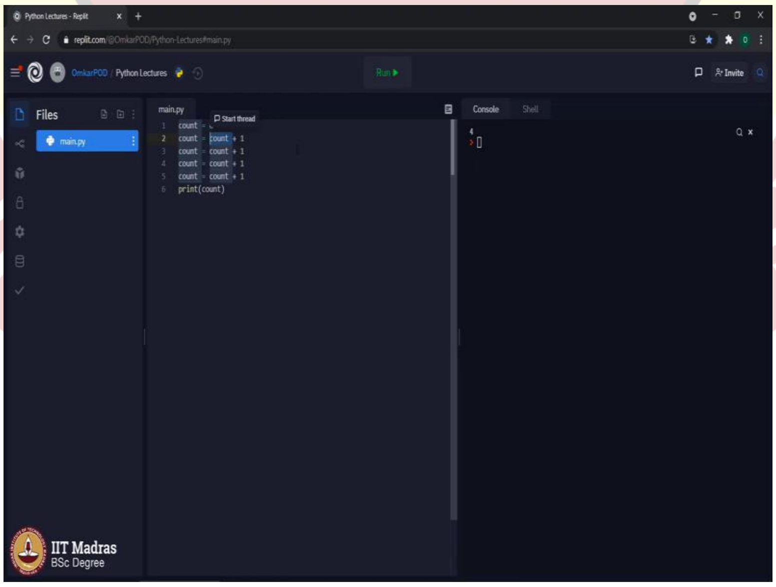


```
1  x = 10
2  print(x)
3  del x
4  print(x)
5
```

```
10
[illegible]
> 
```

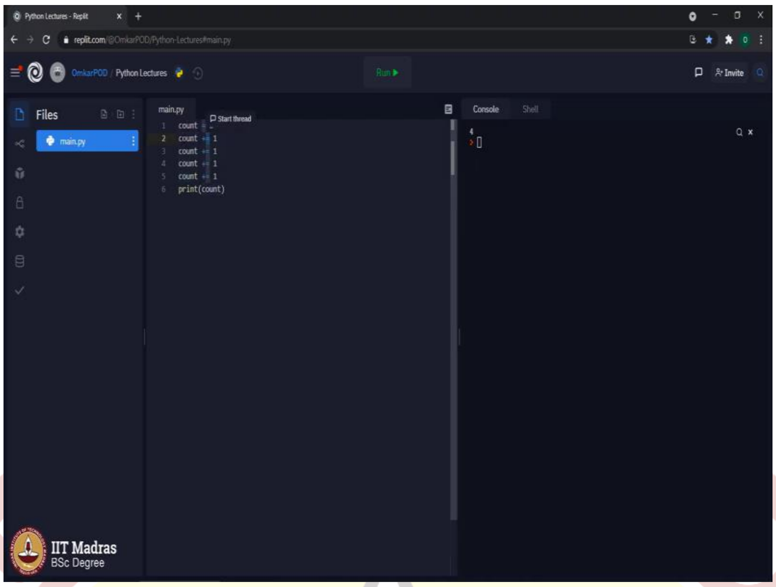
For example, x is equal to 10, print x, del, as in delete x, then print x again. Let us execute this code and see whether the variable x is getting deleted or not. Because of first print statement, line number 2, we got the output as 10 and it is giving us error for line number 4, print x. It says name x is not defined, it says not defined, because we have explicitly told computer to delete that particular variable x from the memory location. Hence, at line number 4 does not exist.

(Refer Slide Time: 8:47)



```
count = [illegible]
count = count + 1
count = count + 1
count = count + 1
count = count + 1
print(count)
```

```
4
> 
```



```
1  count = [illegible]
2  count += 1
3  count += 1
4  count += 1
5  count += 1
6  print(count)
```

```
4
> 
```

Next concept is called shorthand operators. Earlier we have seen arithmetic operators. Shorthand operators are kind of arithmetic operators but they have a unique feature. Let us look at this code. We are counting some values. Hence the initial value for variable count is 0. When we count 1 ideally, we write a code something like count is equal to count plus 1. Then when we count second, we repeat the same step and then once again the same and say, finally, we print the variable count, we get the value as 4, which is absolutely correct.

But writing this name of the variable count again and again, is bit tedious and unnecessary, that is why Python has a solution to this particular problem and the solution is called shorthand operator, which help us to remove a repetitive use of same variable name in the same statement.

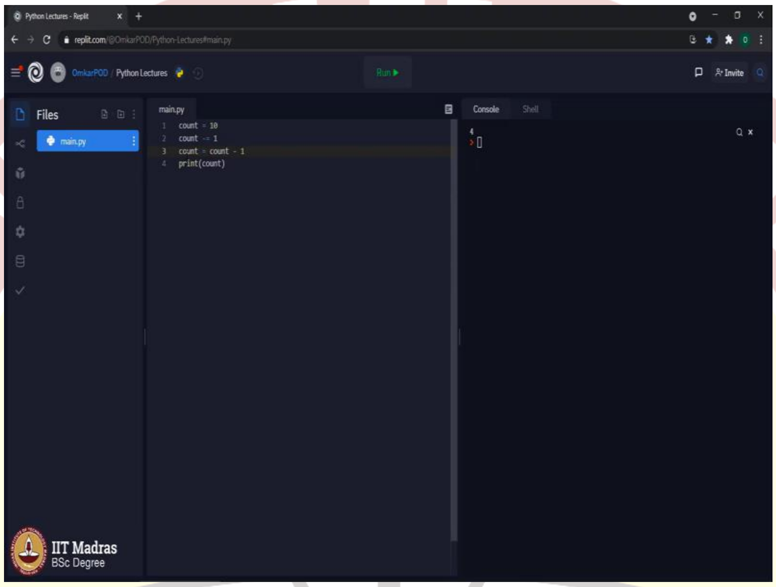
We can remove this count from here and instead of writing it like this, we will write it like this, count plus equal to 1, let us replace count plus equal to 1. Let us execute again, still we are getting the same result, which is 4 but now, we have to write this variable only once, instead of writing it twice in the same line.

While reading this kind of a statement, we should read it as count is equal to count plus 1, which means at a computer level, the statement in line number 2 and line number 3 are identical, it will give the exact same result, then you must be thinking, why we require shorthand operators?

And the answer is shorthand operators are introduced to make our life, as in programmer's life easier because this kind of incrementation or decrementation operations, we have to do so many times in a typical code and the use of shorthand operators to replace the regular increment operation reduces the time.

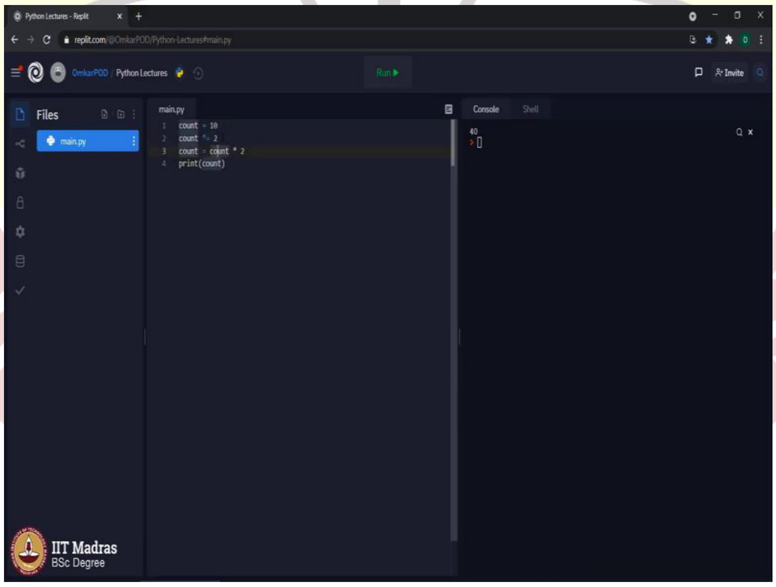
Hence, it helps us code fast. Now, you might be thinking, can we use it only with addition operator or is it possible to use with other arithmetic operators as well? The answer is, yes, it is possible to use such a shorthand operator with all arithmetic operators.

(Refer Slide Time: 11:50)



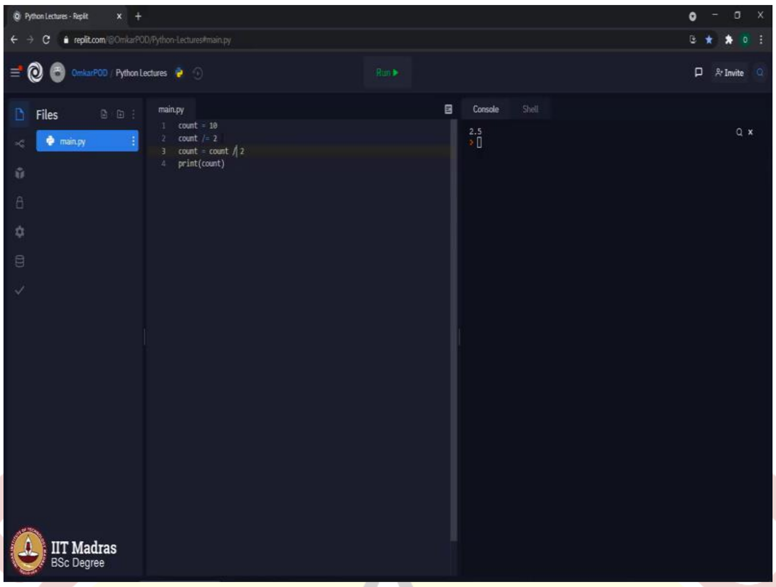
```
count = 10
count -= 1
count = count - 1
print(count)
```

```
4
>
```



```
count = 10
count *= 2
count = count * 2
print(count)
```

```
40
>
```



```
1  count = 10
2  count /= 2
3  count = count / 2
4  print(count)
```

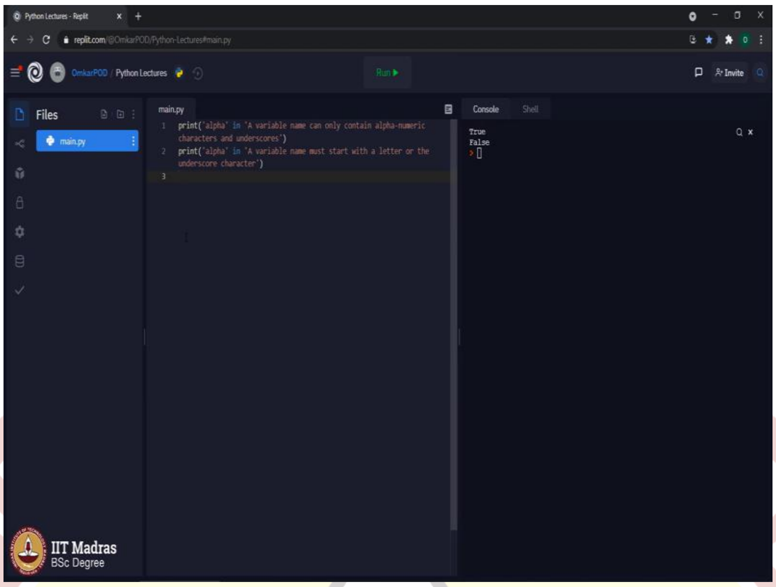
```
2.5
>
```

Let us try to, let us say count is 10 and this is minus equal to and this is minus let us see what output we get. It is 8, because we decremented the value of variable count twice, first with shorthand operator and second with a regular statement, which is count is equal to count minus 1. Similar thing can be done using multiplication operator, let us execute. It says 40 because initially, the value for variable count was 10.

Because of statement in line number 2, the value became 10 into 2 that is 20 and then line number 3, it became 20 into 2 40. Let us try division also, first 10 divided by 2, which is 5 and then 5 divided by 2 which is 2.5.

Next concept is a special operator called in operator. It allows us to perform a similar operation, which usually happens with search engines, where we type some keywords and that particular keyword, is checked against all the possible documents and if there is a match, then we get that document in the search results.

(Refer Slide Time: 13:21)



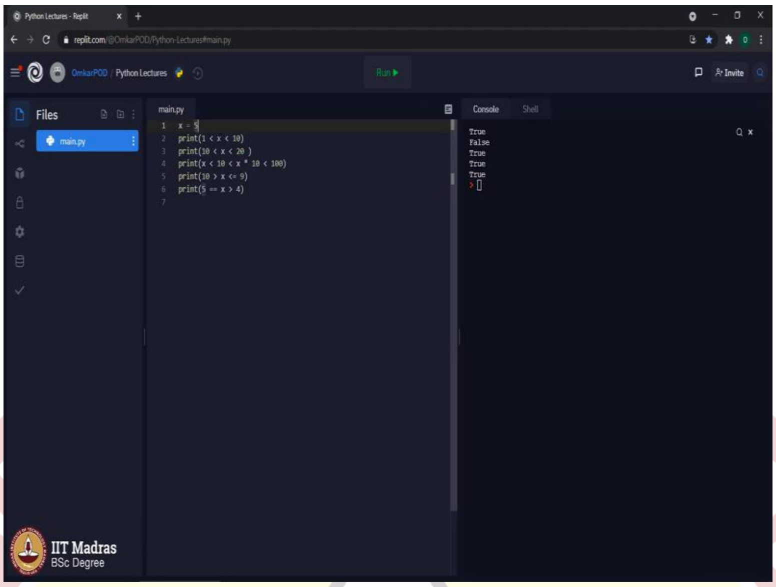
```python
print('alpha' in 'A variable name can only contain alpha-numeric characters and underscores')
print('alpha' in 'A variable name must start with a letter or the underscore character')
```

```
True
False
```

Similar thing can be done in Python, as well. Let us look at this particular example. First print statement, first string alpha then the operator in followed by a second string that says variable name can only contain alphanumeric characters and underscores. Similarly, first string is same in second print statement followed by in a variable name must start with a letter or the underscore character.

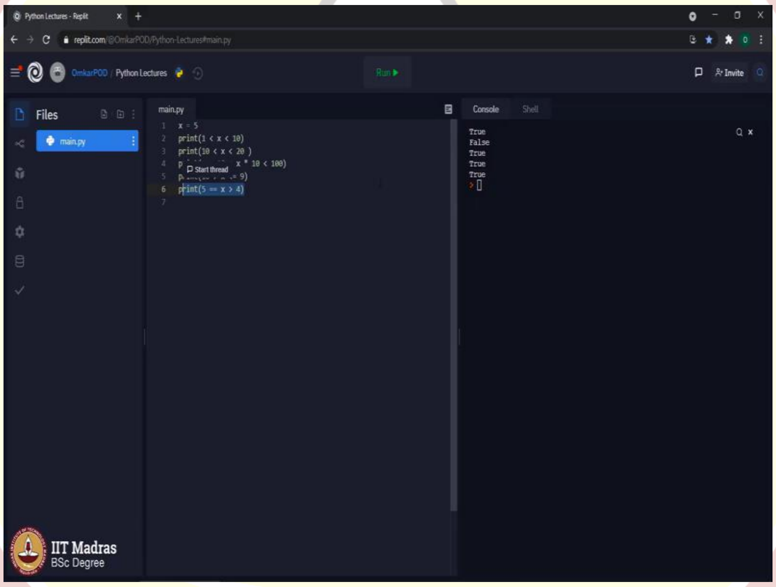
Before explaining this code, let us execute it, then we will see why we are getting a particular output. It says true followed by false. As I explained earlier the in operator checks, if a particular value exists in something else or not, which means a computer is actually checking a string alpha exists inside this particular string or not. If it exists, it returns true that is how this particular in operator works. The result of in operator is always a Boolean value.

(Refer Slide Time: 14:34)



```python
x = 5
print(1 < x < 10)
print(10 < x < 20 )
print(x < 10 < x * 10 < 100)
print(10 > x <= 9)
print(5 == x > 4)
```

```
True
False
True
True
True
>
```



```python
x = 5
print(1 < x < 10)
print(10 < x < 20 )
p            x * 10 < 100)
p              9)
print(5 == x > 4)
```

```
True
False
True
True
True
>
```

Now the next concept is related to expressions and it is called as chaining operators. Let us look at a small Python code and see how chaining operators work in Python. x is equal to 5 print 1 is less than x less than 10. Print 10 is less than x less than 20. Print x is less than 10 less than x multiplied by 10 less than 100; print 10 is greater than x less than equal to 9, print 5 is equal equal to x greater than 4.

Now, if you observe any of these print statements, you will see, we have used two or more relational operators in a single statement. When we use multiple relational operators in the single statement, then it is called as chaining operators. Let us execute the code and see what kind of output we are getting. First, it says true, because x is 5 and 1 less than 5 less than 10 statement is true, because 5 lies between 1 and 10.

Second statement turns out to be false, because 5 less than 20 might be correct, but 10 less than 5 is an incorrect statement. In order to get true as a final output, all possible conditions in that particular statement has to be true. Therefore, the output for second print statement is false. Next print statement, x less than 10 which is 5 less than 10.

That is true, less than x multiplied by 10 which is 5 multiplied by 10 which is 50 then less than 100. That is why we are getting true because we are saying 5 less than 10 less than 50 less than 100. Here all relational operators are giving output which is true. Similarly, in the next print statement 10 is greater than 5 and 5 is less than or equal to 9.

In the last print statement, where 5 is equal equal to x and that is greater than 4. Hence, once again the output is true. When we use these relational operators in this particular manner, then it is referred as chaining operators. Thank you for watching this lecture. Happy learning!