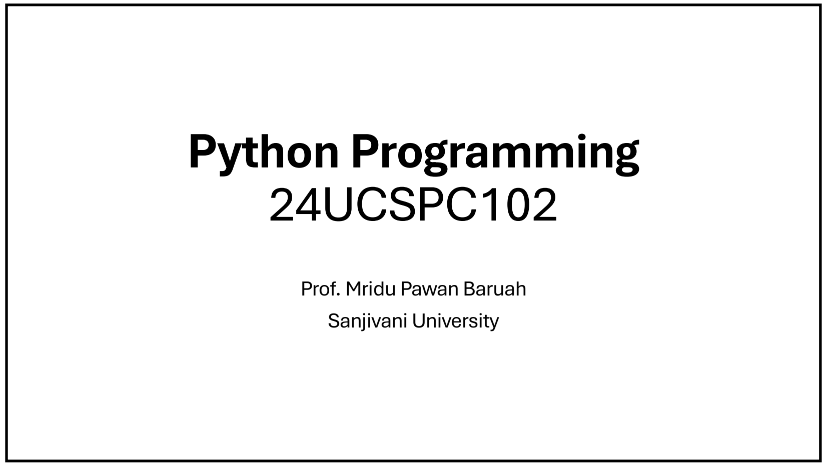
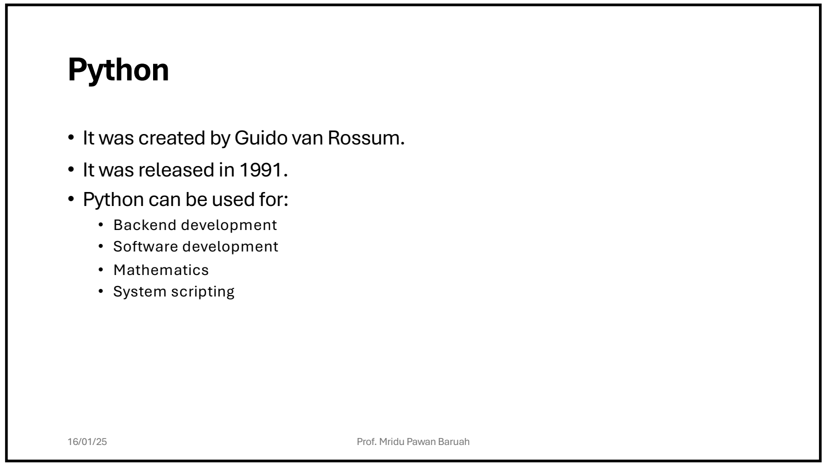


Python Programming
24UCSPC102

Prof. Mridu Pawan Baruah
Sanjivani University

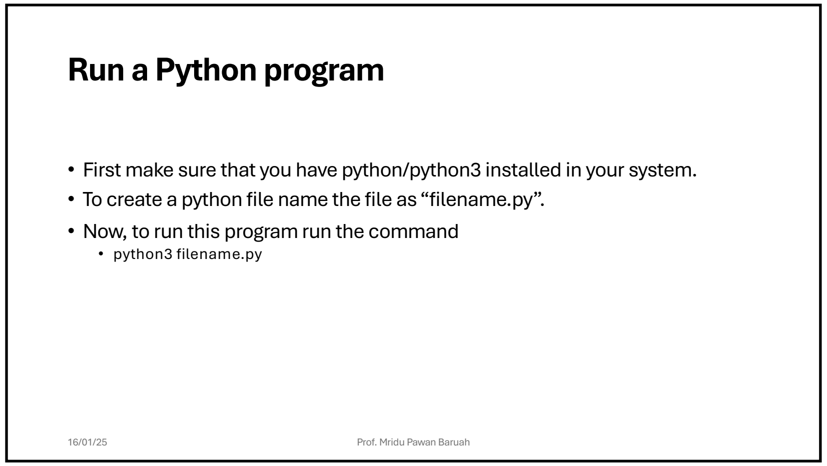
1



Python

- It was created by Guido van Rossum.
- It was released in 1991.
- Python can be used for:
  - Backend development
  - Software development
  - Mathematics
  - System scripting

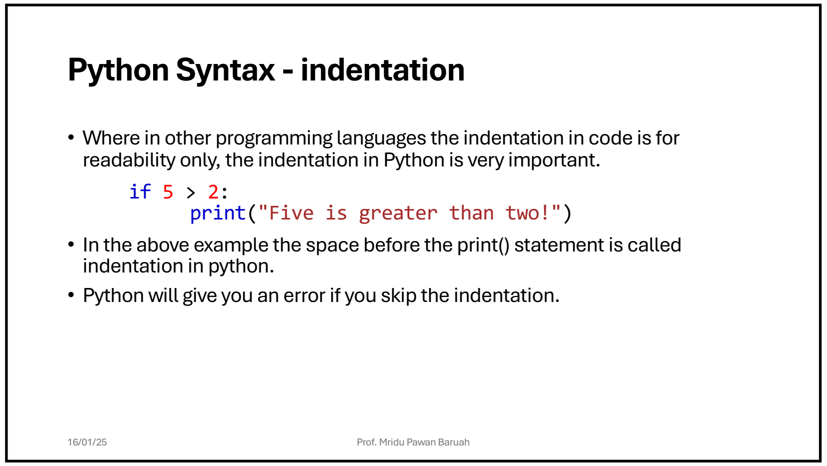
2



Run a Python program

- First make sure that you have python/python3 installed in your system.
- To create a python file name the file as “filename.py”.
- Now, to run this program run the command
  - python3 filename.py

3



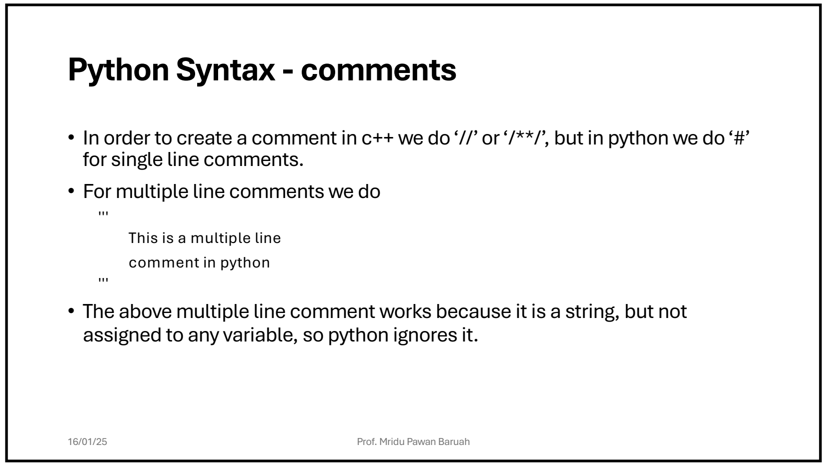
Python Syntax - indentation

- Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

```
if 5 > 2:
        print("Five is greater than two!")
```

- In the above example the space before the print() statement is called indentation in python.
- Python will give you an error if you skip the indentation.

4



Python Syntax - comments

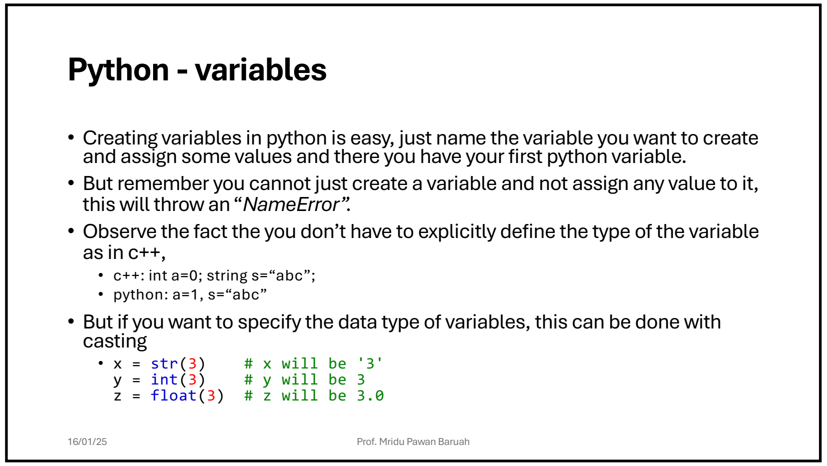
- In order to create a comment in c++ we do '//' or '/**/', but in python we do '#' for single line comments.
- For multiple line comments we do

```
'''
    This is a multiple line
    comment in python
'''
```

- The above multiple line comment works because it is a string, but not assigned to any variable, so python ignores it.

16/01/25 Prof. Mridu Pawan Baruah

5



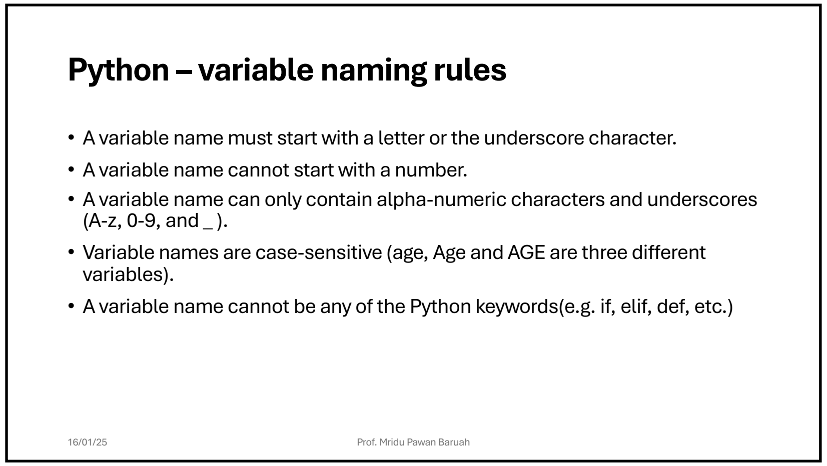
Python - variables

- Creating variables in python is easy, just name the variable you want to create and assign some values and there you have your first python variable.
- But remember you cannot just create a variable and not assign any value to it, this will throw an "*NameError"*.
- Observe the fact the you don't have to explicitly define the type of the variable as in c++,
  - c++: int a=0; string s="abc";
  - python: a=1, s="abc"
- But if you want to specify the data type of variables, this can be done with casting

  -
    ```
    x = str(3)    # x will be '3'
    y = int(3)    # y will be 3
    z = float(3)  # z will be 3.0
    ```

16/01/25 Prof. Mridu Pawan Baruah

6

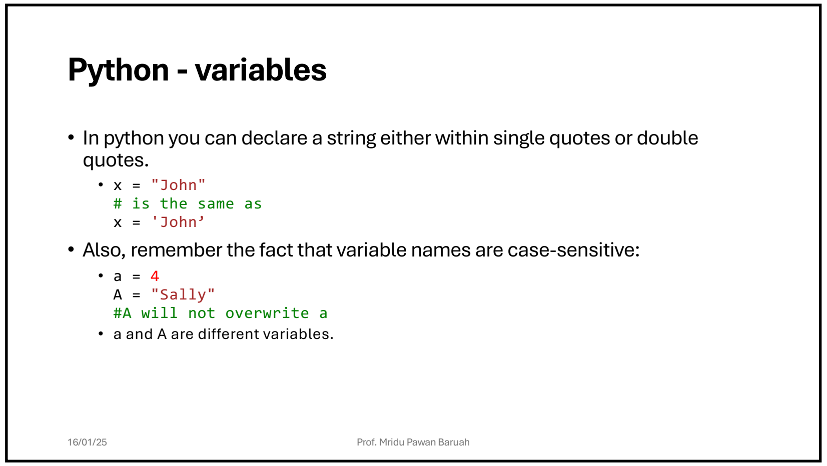


Python – variable naming rules

- A variable name must start with a letter or the underscore character.
- A variable name cannot start with a number.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
- Variable names are case-sensitive (age, Age and AGE are three different variables).
- A variable name cannot be any of the Python keywords(e.g. if, elif, def, etc.)

16/01/25 Prof. Mridu Pawan Baruah

7



Python - variables

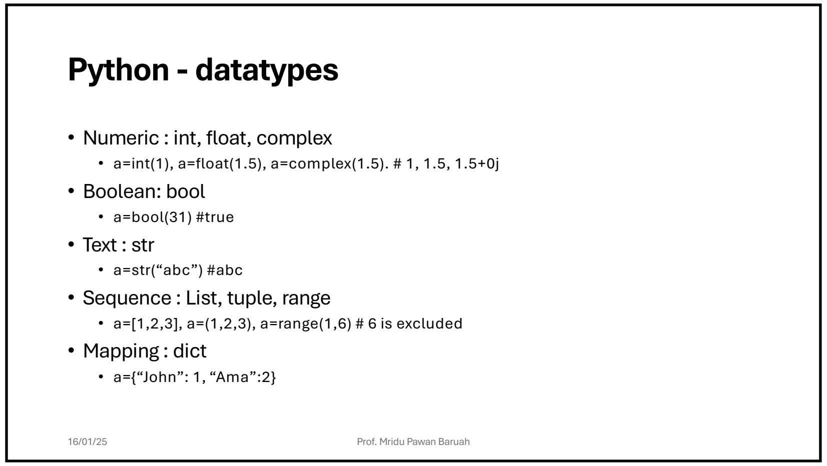
- In python you can declare a string either within single quotes or double quotes.

  -
    ```
    x = "John"
    # is the same as
    x = 'John'
    ```
- Also, remember the fact that variable names are case-sensitive:

  -
    ```
    a = 4
    A = "Sally"
    #A will not overwrite a
    ```
  - a and A are different variables.

16/01/25 Prof. Mridu Pawan Baruah

8

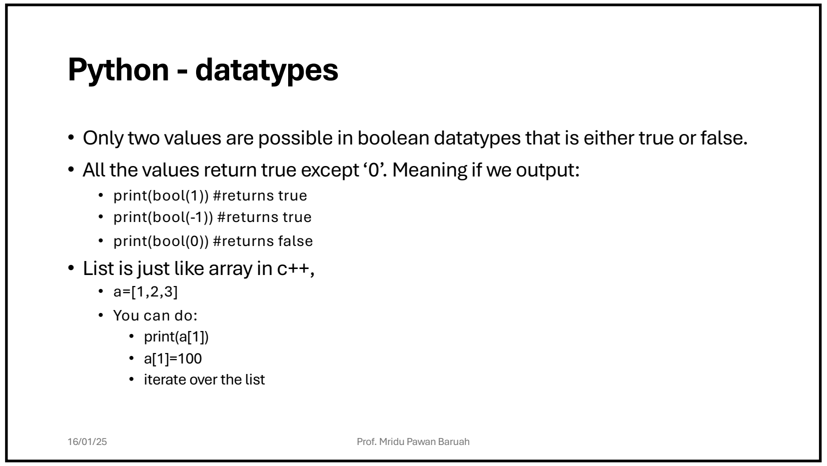


Python - datatypes

- Numeric : int, float, complex
  - a=int(1), a=float(1.5), a=complex(1.5). # 1, 1.5, 1.5+0j
- Boolean: bool
  - a=bool(31) #true
- Text : str
  - a=str("abc") #abc
- Sequence : List, tuple, range
  - a=[1,2,3], a=(1,2,3), a=range(1,6) # 6 is excluded
- Mapping : dict
  - a={"John": 1, "Ama":2}

16/01/25 Prof. Mridu Pawan Baruah

9

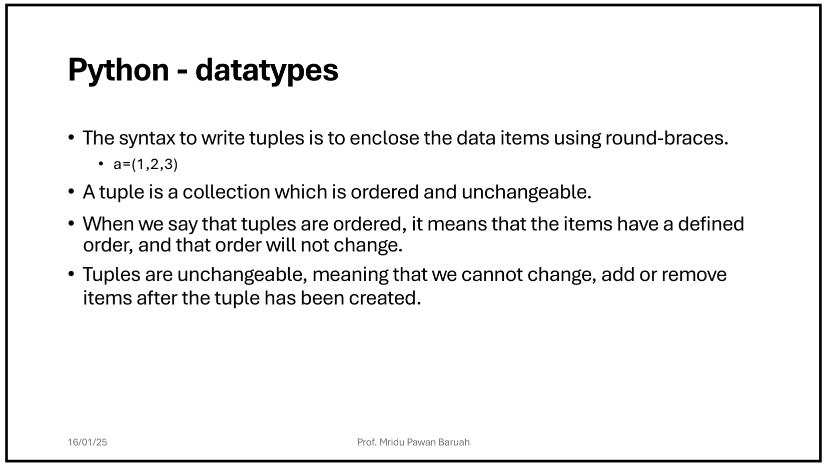


Python - datatypes

- Only two values are possible in boolean datatypes that is either true or false.
- All the values return true except '0'. Meaning if we output:
  - print(bool(1)) #returns true
  - print(bool(-1)) #returns true
  - print(bool(0)) #returns false
- List is just like array in c++,
  - a=[1,2,3]
  - You can do:
    - print(a[1])
    - a[1]=100
    - iterate over the list

16/01/25 Prof. Mridu Pawan Baruah

10

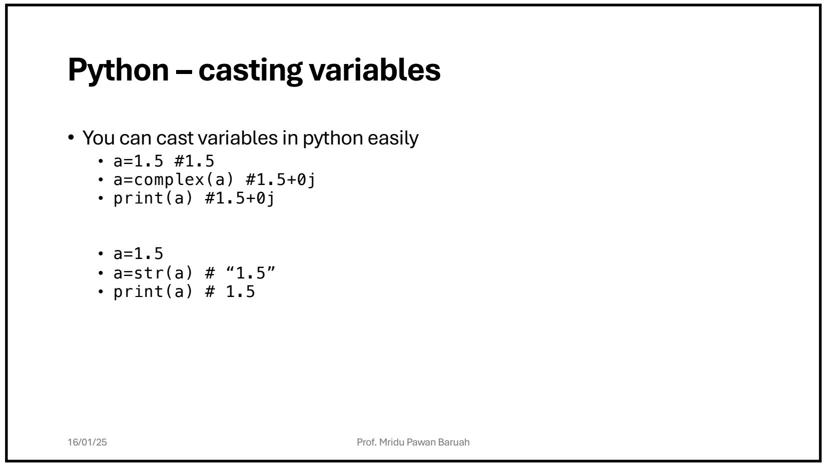


Python - datatypes

- The syntax to write tuples is to enclose the data items using round-braces.
  - a=(1,2,3)
- A tuple is a collection which is ordered and unchangeable.
- When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.
- Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

16/01/25 Prof. Mridu Pawan Baruah

11



Python – casting variables

- You can cast variables in python easily
  - `a=1.5 #1.5`
  - `a=complex(a) #1.5+0j`
  - `print(a) #1.5+0j`

  - `a=1.5`
  - `a=str(a) # "1.5"`
  - `print(a) # 1.5`

16/01/25 Prof. Mridu Pawan Baruah

12

Python - operators

- Python divides the operators in the following groups:
  - Arithmetic operators
  - Assignment operators
  - Comparison operators
  - Logical operators
  - Identity operators
  - Membership operators
  - Bitwise operators

13

Python – arithmetic operators

Operator	Name	Example
+	Addition	x+y
-	Subtraction	x-y
*	Multiplication	x*y
/	Divide	x/y
%	Modulus	x%y
**	Exponentiation	x**y
//	Floor Division	x//y

14

Python – assignment operators

Operator	Example	Equivalent to
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3

- Same can be for the rest of the operators like /, **, %, etc.
- Also, remember the fact that there is no pre-increment and post-increment operators in python.s

15

Python – comparison operators

Operator	Name	Example
==	Equal	x==y
!=	Not equal	x!=y
>	Greater than	x>y
<	Less than	x<y
>=	Greater than or equals to	x>=y
<=	Less than or equals to	x<=y

16

Python – Logical operators

Operator	Description	Example
and	Returns true if both statements are true	x and y
or	Returns true if anyone of the statements is true	x or y
not	Returns the opposite of the result	x not y

17

Python – identity operators

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
Is not	Returns True if both variables are not the same object	x is not y

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x
print(x is z) # returns True because z is the same object as x
print(x is y) # returns False because x is not the same object as y, even if they have the same content
print(x == y) # to demonstrate the difference between "is" and "==": this comparison returns True because
x is equal to y
```

Output: True False True

18

Python – membership operators

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

```
x = ["apple", "banana"]

print("banana" in x)
# returns True because a sequence with the value "banana" is in the list
```

Output: True

19

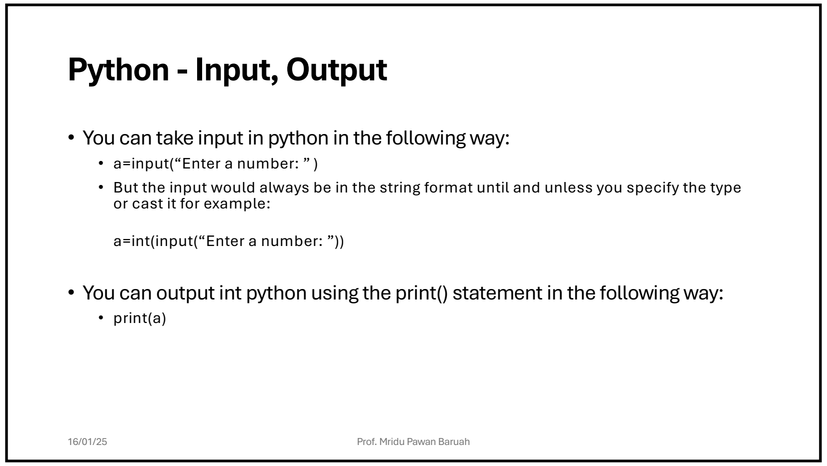
Python - bitwise operators

Operator	Name	Description	Example
&	AND	sets each bit to 1 if both bits are 1	x & y
|	OR	sets each bit to 1 if one of two bits is 1	x | y
^	XOR	sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	inverts all the bits	~x
<<	Left Shift	shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Right Shift	shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x>>2

Example:

3 >> 1, (11>>1), (01—right shifted 1 time),
1

20



Python - Input, Output

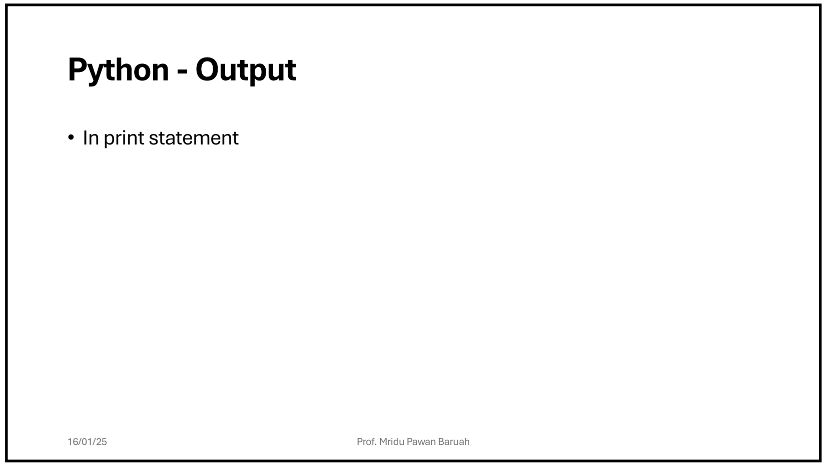
- You can take input in python in the following way:
  - a=input(“Enter a number: ”)
  - But the input would always be in the string format until and unless you specify the type or cast it for example:

    a=int(input(“Enter a number: ”))

- You can output int python using the print() statement in the following way:
  - print(a)

16/01/25 Prof. Mridu Pawan Baruah

21

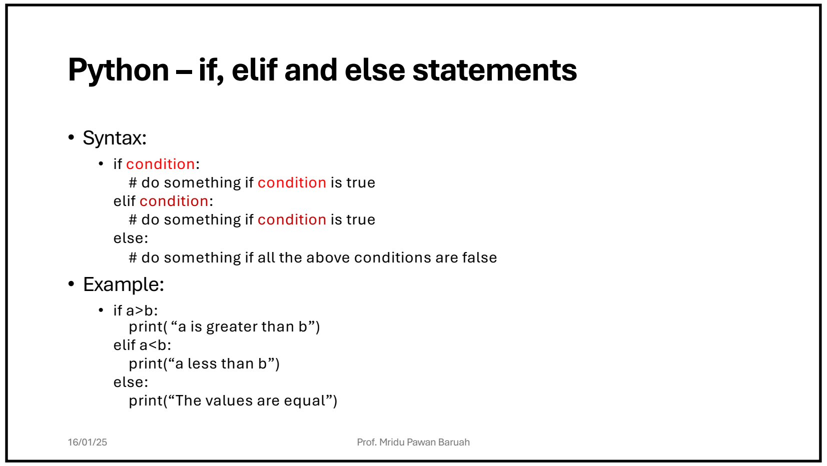


Python - Output

- In print statement

16/01/25 Prof. Mridu Pawan Baruah

22



Python – if, elif and else statements

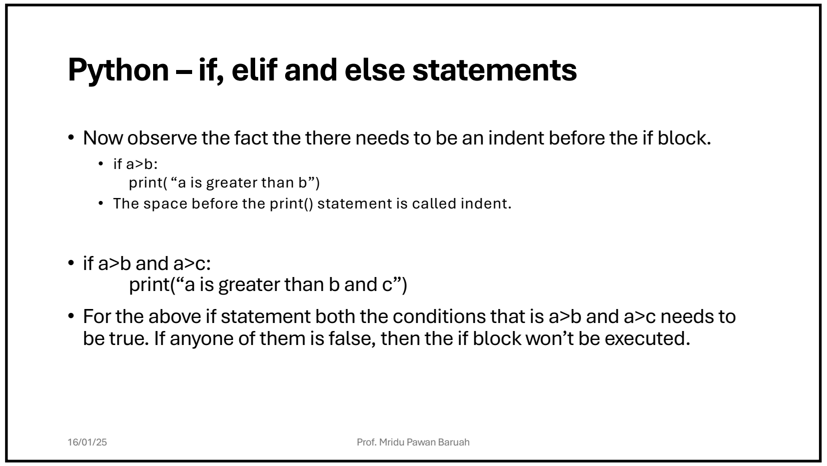
- Syntax:

  -
    ```
    if condition:
       # do something if condition is true
    elif condition:
       # do something if condition is true
    else:
       # do something if all the above conditions are false
    ```
- Example:

  -
    ```
    if a>b:
       print( “a is greater than b”)
    elif a<b:
       print(“a less than b”)
    else:
       print(“The values are equal”)
    ```

16/01/25 Prof. Mridu Pawan Baruah

23



Python – if, elif and else statements

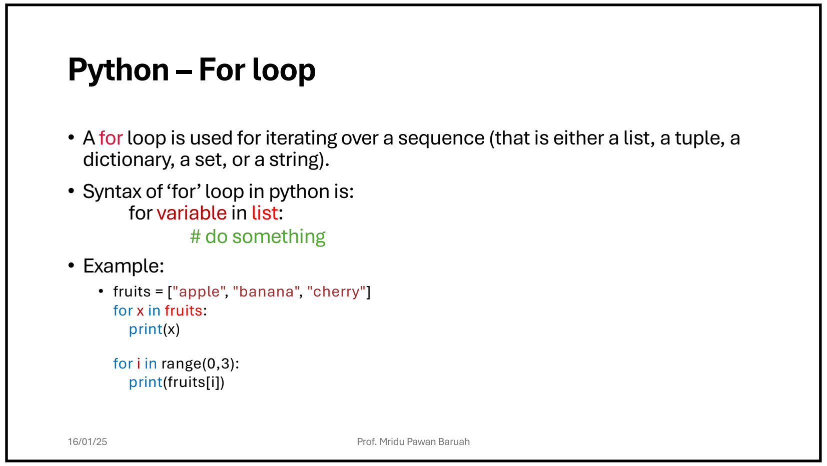
- Now observe the fact the there needs to be an indent before the if block.

  -
    ```
    if a>b:
       print( “a is greater than b”)
    ```
  - The space before the print() statement is called indent.

-
  ```
  if a>b and a>c:
        print(“a is greater than b and c”)
  ```
- For the above if statement both the conditions that is a>b and a>c needs to be true. If anyone of them is false, then the if block won’t be executed.

16/01/25 Prof. Mridu Pawan Baruah

24



Python – For loop

- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- Syntax of ‘for’ loop in python is:

```
for variable in list:
        # do something
```

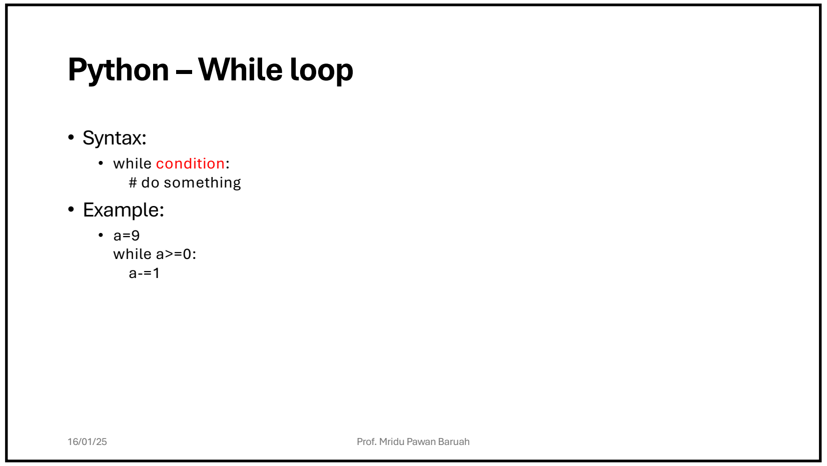
- Example:

  -
    ```
    fruits = ["apple", "banana", "cherry"]
    for x in fruits:
      print(x)

    for i in range(0,3):
      print(fruits[i])
    ```

16/01/25 Prof. Mridu Pawan Baruah

25



Python – While loop

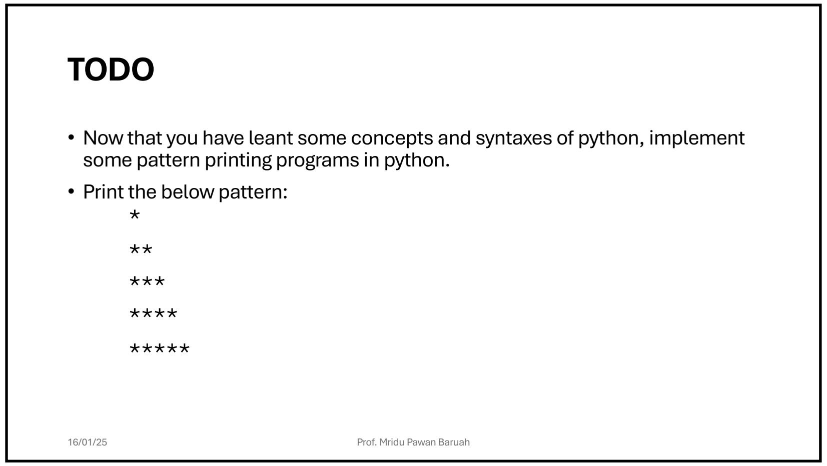
- Syntax:

  -
    ```
    while condition:
      # do something
    ```
- Example:

  -
    ```
    a=9
    while a>=0:
      a-=1
    ```

16/01/25 Prof. Mridu Pawan Baruah

26



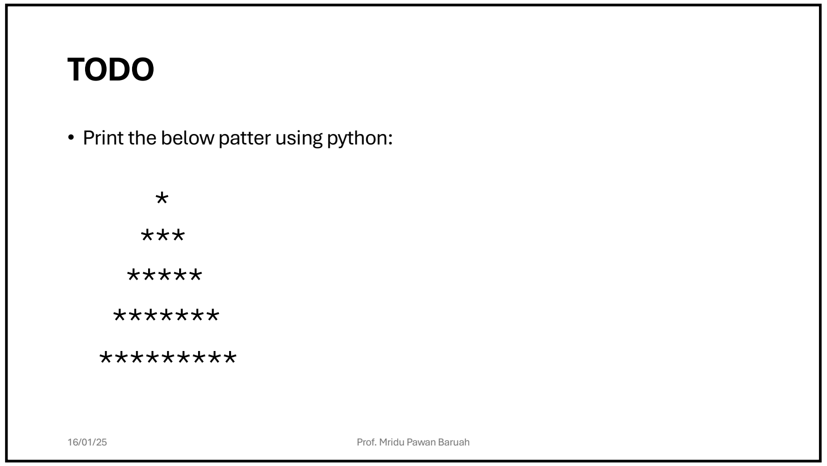
TODO

- Now that you have leant some concepts and syntaxes of python, implement some pattern printing programs in python.
- Print the below pattern:

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

16/01/25 Prof. Mridu Pawan Baruah

27



TODO

- Print the below patter using python:

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

16/01/25 Prof. Mridu Pawan Baruah

28

Solution

- Solution 1:

```
for i in range(0,6):
    for j in range (0,i+1):
        print("*", end="")
    print()
```

- Solution 2:

```
    for I in range(1,6):
        for j in range(1, 6-i):
            print(" ",end="")
        for j in range(1,2*i):
            print("*",end="")
        print()
```

  - OR

```
    for i in range(1,5+1):
        print(" "*(5-i)+"*"*(2*i-1))
```

Python - functions

- A function is a block of code which only runs when it is called.
- We may pass parameters into a function.
- A function can return data as a result.
- In order to create a function in python we use the keyword def.

Python- function creation

- Function creating:
- def function_name(parameters):
  # do something_ write your logic
- Example of a function adding to numbers:

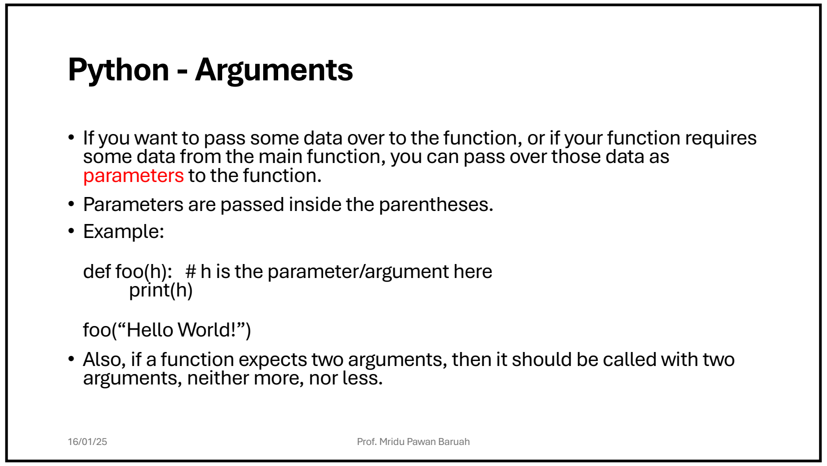
```
def addToNumbers():
    print("hello World!")
```

Python - function calling

- To call a function, use the function name followed by parenthesis:

```
def function_name():
    print("Hello World!")

function_name() #calling a function
```



Python - Arguments

- If you want to pass some data over to the function, or if your function requires some data from the main function, you can pass over those data as parameters to the function.
- Parameters are passed inside the parentheses.
- Example:

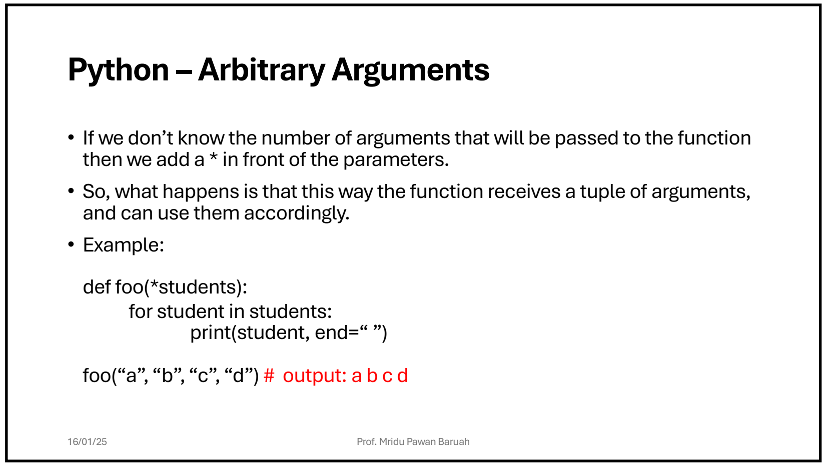
```
def foo(h):   # h is the parameter/argument here
        print(h)

foo("Hello World!")
```

- Also, if a function expects two arguments, then it should be called with two arguments, neither more, nor less.

16/01/25 Prof. Mridu Pawan Baruah

33



Python – Arbitrary Arguments

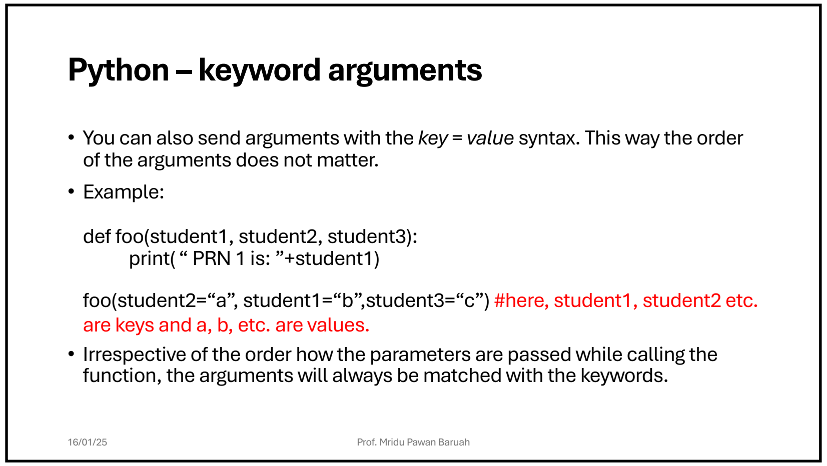
- If we don't know the number of arguments that will be passed to the function then we add a * in front of the parameters.
- So, what happens is that this way the function receives a tuple of arguments, and can use them accordingly.
- Example:

```
def foo(*students):
        for student in students:
                print(student, end=" ")

foo("a", "b", "c", "d") #  output: a b c d
```

16/01/25 Prof. Mridu Pawan Baruah

34



Python – keyword arguments

- You can also send arguments with the *key* = *value* syntax. This way the order of the arguments does not matter.
- Example:

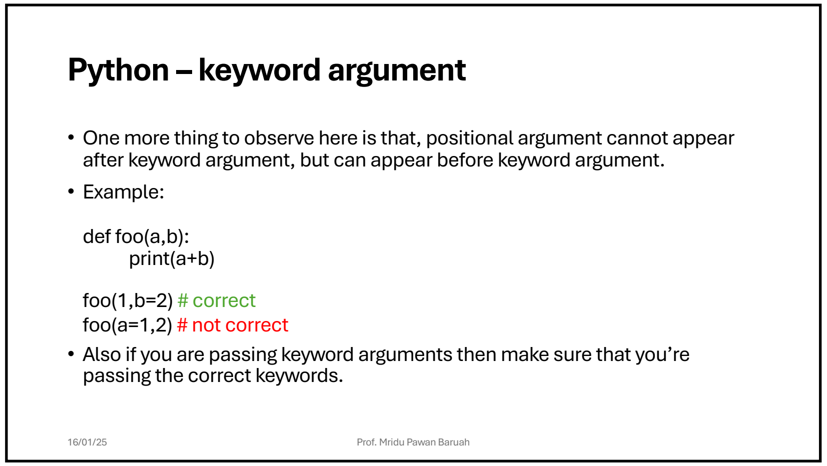
```
def foo(student1, student2, student3):
        print( " PRN 1 is: "+student1)

foo(student2="a", student1="b",student3="c") #here, student1, student2 etc. are keys and a, b, etc. are values.
```

- Irrespective of the order how the parameters are passed while calling the function, the arguments will always be matched with the keywords.

16/01/25 Prof. Mridu Pawan Baruah

35



Python – keyword argument

- One more thing to observe here is that, positional argument cannot appear after keyword argument, but can appear before keyword argument.
- Example:

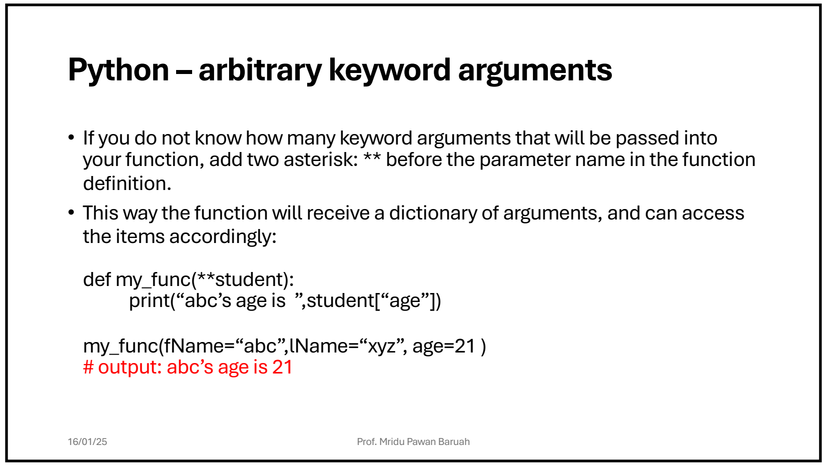
```
def foo(a,b):
        print(a+b)

foo(1,b=2) # correct
foo(a=1,2) # not correct
```

- Also if you are passing keyword arguments then make sure that you're passing the correct keywords.

16/01/25 Prof. Mridu Pawan Baruah

36



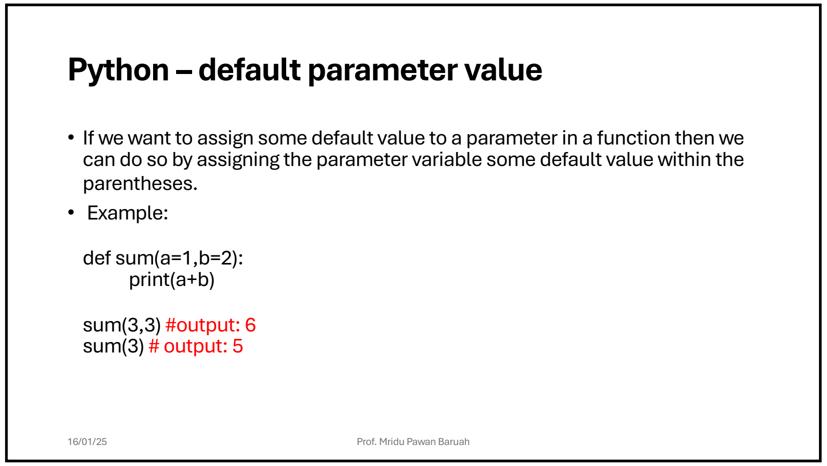
Python – arbitrary keyword arguments

- If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.
- This way the function will receive a dictionary of arguments, and can access the items accordingly:

```
def my_func(**student):
        print(“abc’s age is  ”,student[“age”])

my_func(fName=“abc”,lName=“xyz”, age=21 )
# output: abc’s age is 21
```

37



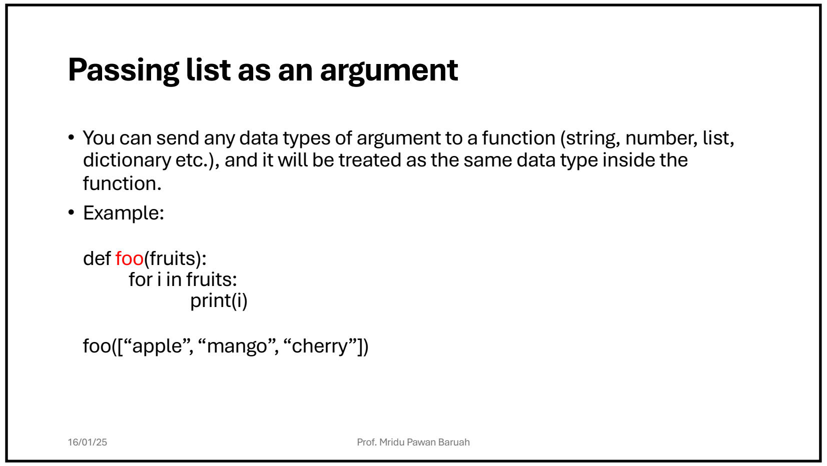
Python – default parameter value

- If we want to assign some default value to a parameter in a function then we can do so by assigning the parameter variable some default value within the parentheses.
- Example:

```
def sum(a=1,b=2):
        print(a+b)

sum(3,3) #output: 6
sum(3) # output: 5
```

38



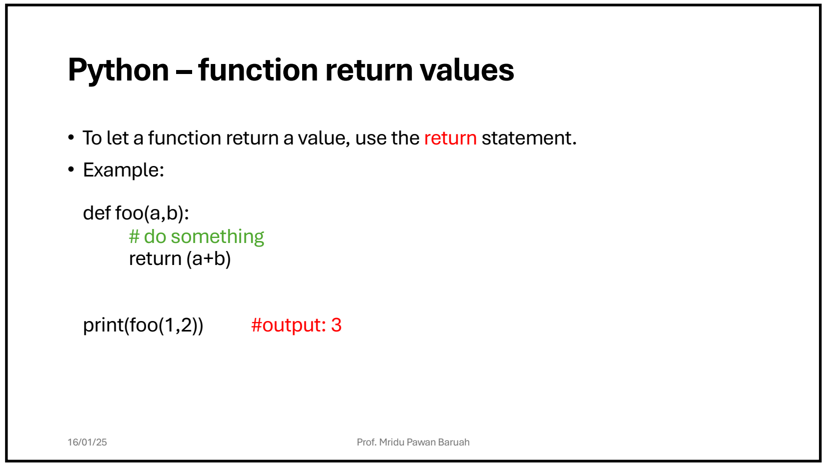
Passing list as an argument

- You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.
- Example:

```
def foo(fruits):
        for i in fruits:
                print(i)

foo([“apple”, “mango”, “cherry”])
```

39



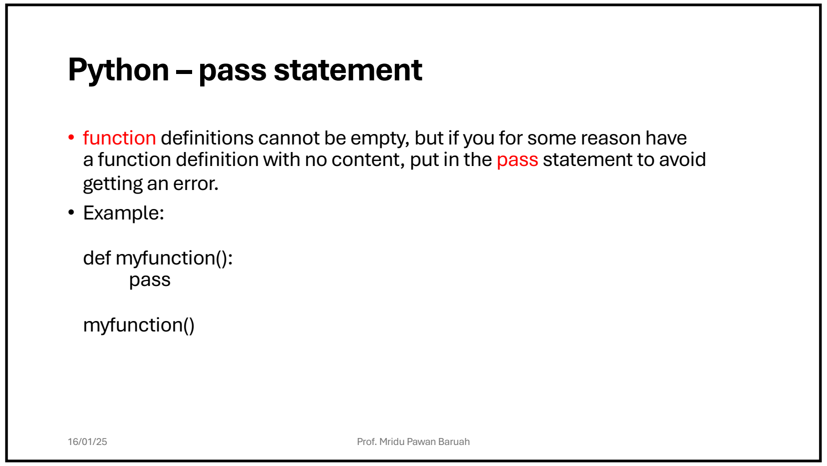
Python – function return values

- To let a function return a value, use the return statement.
- Example:

```
def foo(a,b):
        # do something
        return (a+b)


print(foo(1,2))        #output: 3
```

40



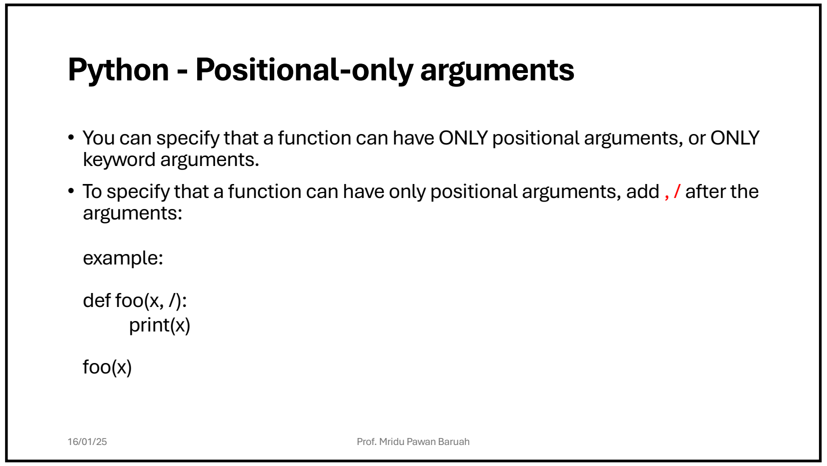
Python – pass statement

- function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.
- Example:

```
def myfunction():
        pass

myfunction()
```

41



Python - Positional-only arguments

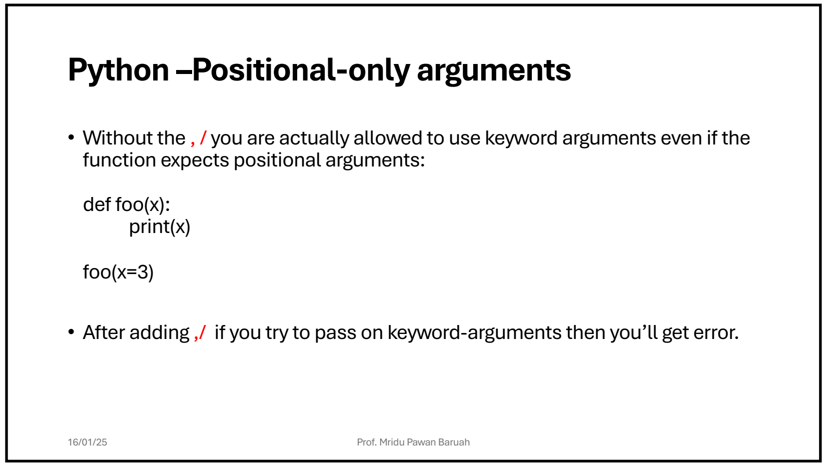
- You can specify that a function can have ONLY positional arguments, or ONLY keyword arguments.
- To specify that a function can have only positional arguments, add , / after the arguments:

example:

```
def foo(x, /):
        print(x)

foo(x)
```

42



Python –Positional-only arguments

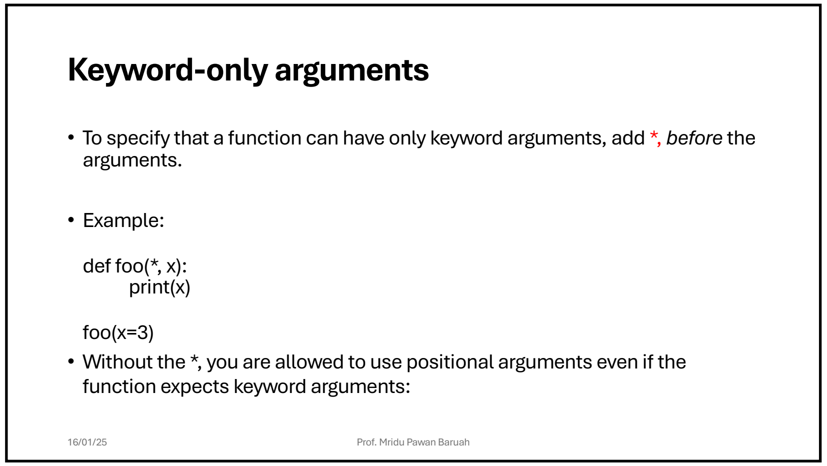
- Without the , / you are actually allowed to use keyword arguments even if the function expects positional arguments:

```
def foo(x):
        print(x)

foo(x=3)
```

- After adding ,/ if you try to pass on keyword-arguments then you'll get error.

43



Keyword-only arguments

- To specify that a function can have only keyword arguments, add *, *before* the arguments.
- Example:

```
def foo(*, x):
        print(x)

foo(x=3)
```

- Without the *, you are allowed to use positional arguments even if the function expects keyword arguments:

44

Combined Positional-only and Keyword-only

- You can combine the two argument types in the same function.
- Any argument before the / , are positional-only, and any argument after the *, are keyword-only.
- Example:

```
def foo(a, b, /, *, c, d):
        print(a + b + c + d)

foo(5, 6, c = 7, d = 8)
```

45

Python - Recursion

- Python also accepts function recursion, which means a defined function can call itself.
- Example:

```
def recur(k):
        if(k==0):
                return
        else:
                print(k)
                recur(k-1)

recur(3) #output: 3 2 1
```

46

Variable scope- Local scope

- A variable is only available from inside the region it is created. This is called **scope**.
- A variable created inside a function belongs to the local *scope* of that function, and can only be used inside that function.
- Example of local scope:

```
def foo():
        x=200
        print(x)

foo()           # output: 200
print(x)        # error
```

47

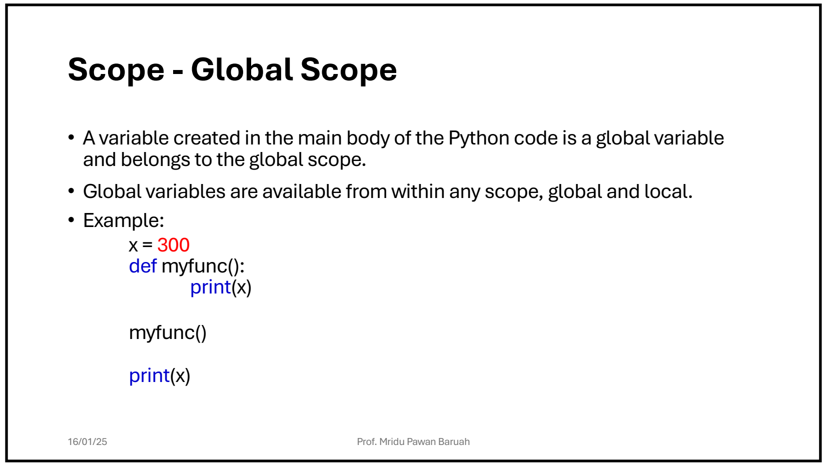
Scope – function inside function

- As explained in the example above, the variable x is not available outside the function, but it is available for any function inside the function:
- Example:

```
def myFunc():
        x=300
        def myInnerFunc():
                print(x)
```

- So, as you can see the inner function named "myInnerFunc()" can access the outer function's variable 'x'.

48



Scope - Global Scope

- A variable created in the main body of the Python code is a global variable and belongs to the global scope.
- Global variables are available from within any scope, global and local.
- Example:

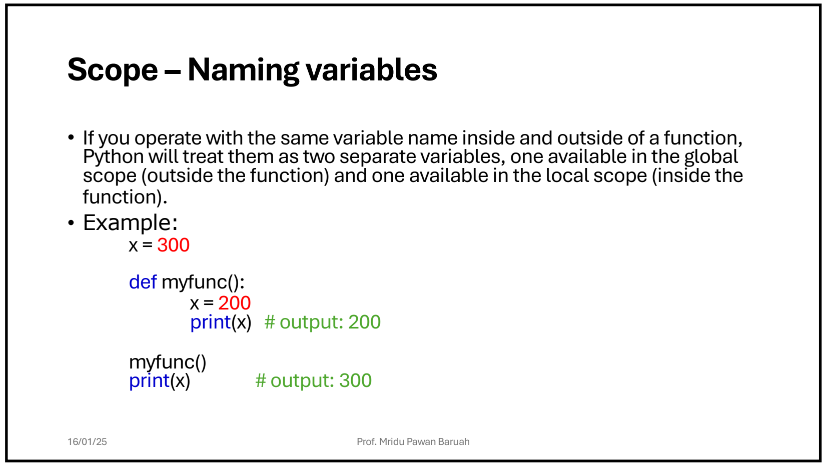
```
x = 300
def myfunc():
    print(x)

myfunc()

print(x)
```

16/01/25 Prof. Mridu Pawan Baruah

49



Scope – Naming variables

- If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function).
- Example:

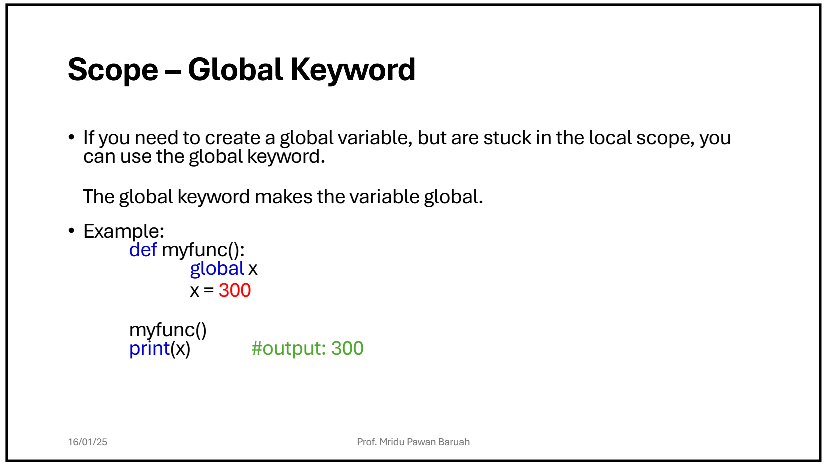
```
x = 300

def myfunc():
    x = 200
    print(x)  # output: 200

myfunc()
print(x)          # output: 300
```

16/01/25 Prof. Mridu Pawan Baruah

50



Scope – Global Keyword

- If you need to create a global variable, but are stuck in the local scope, you can use the global keyword.

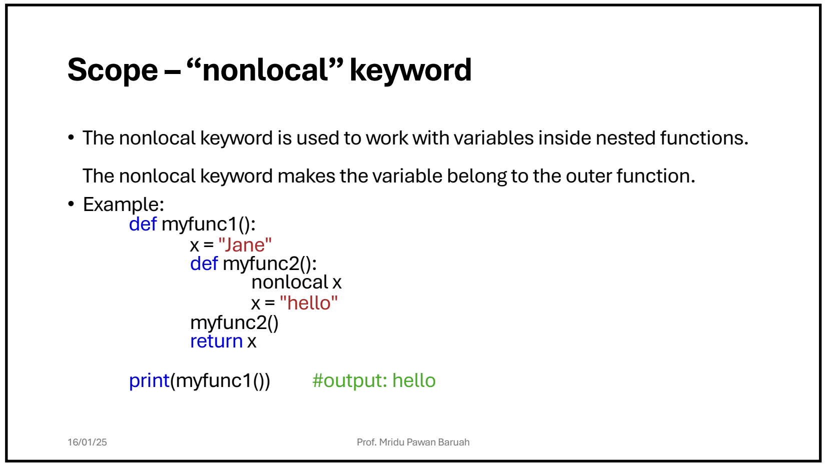
  The global keyword makes the variable global.
- Example:

```
def myfunc():
    global x
    x = 300

myfunc()
print(x)          #output: 300
```

16/01/25 Prof. Mridu Pawan Baruah

51



Scope – “nonlocal” keyword

- The nonlocal keyword is used to work with variables inside nested functions.

  The nonlocal keyword makes the variable belong to the outer function.
- Example:

```
def myfunc1():
    x = "Jane"
    def myfunc2():
        nonlocal x
        x = "hello"
    myfunc2()
    return x

print(myfunc1())        #output: hello
```

16/01/25 Prof. Mridu Pawan Baruah

52