**Instructions**: You must read the material and create an outline of the topics in your OWN words.  Do not copy the text from the tutorials into your notes. Make sure your outline contains notes for each subsection of the reading assignment. Thoroughly cover each topic to show you have a firm understanding of the programming concept or construct.

| **Ques** | **NOTES:** |
| --- | --- |
| Blocks | Input and Output in Python  All computing systems can collect information (input process) and send out information (output process), which differs between programming languages.  We will now look at ways Python does this.  **Input**  When a user supplies information to a program, it is always stored in a variable of any name and as a “string” type. Once stored, it is easily recalled when needed for further operations. Numbers are also stored as strings, making it impossible to do mathematical operations on them, such as addition, subtraction, multiplication, and division, without first having to “cast” it to either an “int” or “float” type.  An example of how to prompt a user for input:  number = input(“How many cheeseburgers would you like?”)  Then we cast the received value to an int:  number = int(number)  A more elegant way would be to cast the value as it is received:  name = int(input(“How many cheeseburgers would you like?”)  **Output**  We use the “print” keyword to output information to the console:  name = input(“Hello! What is your name?”)  print(“Hello,”, name)  The syntax is the arrangement and spelling of keywords to create a well-formed and proper Python statement. Hence, the syntax of the print command shown above is quite simple and uses two literal strings (a string enclosed in quotes) together with a variable holding the user's name. Order matters, which is why the “Hello!” comes first. Here, the variable's value is “dumped out” or “printed” at the point in the sequence where it was placed.  Other ways of printing exist, but the above way is considered the easiest. Another way is to join or concatenate pieces of your output.  name = input(“Hello! What is your name?”)  print(“Hello, “ + name + “!”)  The result would be this:  What is your name? Bill  Hello,Bill!  Notice here that there is no space between the “comma” and the name Bill. This is the case because in the first form of print, the “comma” forces print to add spaces, while in the second form, we must add these spaces ourselves.  Another consideration when concatenating variables is converting variable types and data types. The error shown above tells us we cannot concatenate (i.e., join together) different data types. Here, we are trying to join(+) a string (str) with two different numbers: the (int) number of cheeseburgers AND the total cost (float) of those burgers. Python doesn’t like mixing data types!  But if we cast the integer and the float to strings using “str(number) and str(number\*2.50),” then things will work out. |
|  |  |
| Expressions | Formatting Output  The example above shows that the final answer is “$5.0” rather than “$5.00,” as we want, but this can be fixed using formatting.  Python uses the “%” character as a format specifier. It serves as a placeholder for receiving data; here, the data will replace the format specifier, and the format specifier will dictate its appearance. We also place an additional “%” sign to separate the presentation string from the data.  print(“My favourite ice cream is %s for sure!’ % favourite)  We use the following format specifiers from the image above: %s, %15s, and %-15s. We see an invisible positive sign (+) and negative sign (-) before the number 15, which determines where spaces are added, before or after the string variable.  The same rules apply to numbers. We use %d for integers, and for float (a decimal), the specifier is %f. We can add decimal precision using a decimal number such as %0.2f. This will round off the number to two decimal places and %0.3f would round off to three decimal places.  The initial 0 indicates no padding (i.e., blank spaces) preceding the number. Any number over the length of the numeric expression (e.g., four in the case of 3.14:  three numbers and a decimal point) will be the number of spaces added to the output.  Python Blocks:  In Python, blocks can be looked at as “paragraphs” of code. Here, indents are used to connect related code.  for I in range(5):  print(i)  print(i \* 2)  The same applies to functions:  def my\_code():  x = 5 # Same block  x = x + 4 # same block  print(x) # same block  Python Expressions:  An expression combines operators (-, +, /, \*) and operands (2, q, 3) that will work together to produce some other value.  An expression is evaluated as per the precedence of its operators. Types of expressions in Python included the following:  Constant Expression:  These types have constant values.  x = 15 + 1.3  print(x)  Arithmetic Expressions:  This combination of numeric values, operators, and sometimes parenthesis results in a numeric value. Examples of operators used here are:   |  |  |  | | --- | --- | --- | | **Operators** | **Syntax** | **Functioning** | | + | x + y | Addition | | – | x – y | Subtraction | | \* | x \* y | Multiplication | | / | x / y | Division | | // | x // y | Quotient | | % | x % y | Remainder | | \*\* | x \*\* y | Exponentiation |   # Arithmetic Expressions  x **=** 40  y **=** 12    add **=** x **+** y  sub **=** x **-** y  pro **=** x **\*** y  div **=** x **/** y    print(add)  print(sub)  print(pro)  print(div)  Note here that the values of x and y are dynamic.  Integral Expression:  These only produce integer results after all computations and type conversions.  # Integral Expressions  a **=** 13  b **=** 12.0    c **=** a **+** int(b)  print(c)  Output:  25  Floating Expressions:  These produce floating point numbers as a result of all computations and type conversions.  # Floating Expressions  a **=** 13  b **=** 5    c **=** a **/** b  print(c)  Output:  2.6  Relational Expressions:  Here, arithmetic expressions are written on both sides of the relational operator (>, <, >=, <=). The arithmetic expressions are evaluated first, and then compared as per relational operator, and a boolean output is produced in the end. They are also called Boolean expressions.  # Relational Expressions  a **=** 21  b **=** 13  c **=** 40  d **=** 37    p **=** (a **+** b) >**=** (c **-** d)  print(p)  Output:  True  Logical Expressions  These kinds of expressions result in either True or False, it basically specifies one or more conditions. For example, (10 == 9) is a condition if 10 equals 9.   |  |  |  | | --- | --- | --- | | **Operator** | **Syntax** | **Functioning** | | and | P and Q | It returns true if both P and Q are true otherwise returns false | | or | P or Q | It returns true if at least one of P and Q is true | | not | not P | It returns true if condition P is false | |
|  | Examples:  P **=** (10 **==** 9)  Q **=** (7 > 5)    # Logical Expressions  R **=** P **and** Q  S **=** P **or** Q  T **=** **not** P    print(R)  print(S)  print(T) |
|  | Output:  False  True  True  Bitwise Expressions:  These are the kinds in which computations are performed at the bit level.  # Bitwise Expressions  a **=** 12    x **=** a >> 2  y **=** a << 1    print(x, y)  Output:  3 24  Combinational Expressions:  Here, we use different types of expressions in a single expression, which we call a combinational expression.  # Combinational Expressions  a **=** 16  b **=** 12    c **=** a **+** (b >> 1)  print(c)  Output:  22  Multiple Operators in Expression (Operator Precedence)  This defines the priority of operators and which operator is to be executed first. In Python, the operator precedence is as seen below:   |  |  |  | | --- | --- | --- | | **Precedence** | **Name** | **Operator** | | 1 | Parenthesis | ( ) [ ] { } | | 2 | Exponentiation | \*\* | | 3 | Unary plus or minus, complement | -a , +a , ~a | | 4 | Multiply, Divide, Modulo | /  \*  //  % | | 5 | Addition & Subtraction | +  – | | 6 | Shift Operators | >>  << | | 7 | Bitwise AND | & | | 8 | Bitwise XOR | ^ | | 9 | Bitwise OR | | | | 10 | Comparison Operators | >=  <=  >  < | | 11 | Equality Operators | ==  != | | 12 | Assignment Operators | =  +=  -=  /=  \*= | | 13 | Identity and membership operators | is, is not, in, not in | | 14 | Logical Operators | and, or, not |   If we have more than one operator in an expression, it is evaluated as per operator precedence.  # Multi-operator expression    a **=** 10 **+** 3 **\*** 4  print(a)    b **=** (10 **+** 3) **\*** 4  **print**(b)    c **=** 10 **+** (3 **\*** 4)  print(c)  Output:  22  52  22  Python Mathematical Functions  The “math module” is standard in Python and is always available to use via an import statement and gives access to the underlying C library functions for example:  # Square root calculation  Import math  math.sqrt(4)  Functions in Python Math Module   |  |  | | --- | --- | | List of Functions in Python Math Module | | | Function | Description | | ceil(x) | Returns the smallest integer greater than or equal to x. | | copysign(x, y) | Returns x with the sign of y | | fabs(x) | Returns the absolute value of x | | factorial(x) | Returns the factorial of x | | floor(x) | Returns the largest integer less than or equal to x | | fmod(x, y) | Returns the remainder when x is divided by y | | frexp(x) | Returns the mantissa and exponent of x as the pair (m, e) | | fsum(iterable) | Returns an accurate floating point sum of values in the iterable | | isfinite(x) | Returns True if x is neither an infinity nor a NaN (Not a Number) | | isinf(x) | Returns True if x is a positive or negative infinity | | isnan(x) | Returns True if x is a NaN | | ldexp(x, i) | Returns x \* (2\*\*i) | | modf(x) | Returns the fractional and integer parts of x | | trunc(x) | Returns the truncated integer value of x | | exp(x) | Returns e\*\*x | | expm1(x) | Returns e\*\*x - 1 | | log(x[, b]) | Returns the logarithm of x to the base b (defaults to e) | | log1p(x) | Returns the natural logarithm of 1+x | | log2(x) | Returns the base-2 logarithm of x | | log10(x) | Returns the base-10 logarithm of x | | pow(x, y) | Returns x raised to the power y | | sqrt(x) | Returns the square root of x | | acos(x) | Returns the arc cosine of x | | asin(x) | Returns the arc sine of x | | atan(x) | Returns the arc tangent of x | | atan2(y, x) | Returns atan(y / x) | | cos(x) | Returns the cosine of x | | hypot(x, y) | Returns the Euclidean norm, sqrt(x\*x + y\*y) | | sin(x) | Returns the sine of x | | tan(x) | Returns the tangent of x | | degrees(x) | Converts angle x from radians to degrees | | radians(x) | Converts angle x from degrees to radians | | acosh(x) | Returns the inverse hyperbolic cosine of x | | asinh(x) | Returns the inverse hyperbolic sine of x | | atanh(x) | Returns the inverse hyperbolic tangent of x | | cosh(x) | Returns the hyperbolic cosine of x | | sinh(x) | Returns the hyperbolic cosine of x | | tanh(x) | Returns the hyperbolic tangent of x | | erf(x) | Returns the error function at x | | erfc(x) | Returns the complementary error function at x | | gamma(x) | Returns the Gamma function at x | | lgamma(x) | Returns the natural logarithm of the absolute value of the Gamma function at x | | pi | Mathematical constant, the ratio of circumference of a circle to it's diameter (3.14159...) | | e | mathematical constant e (2.71828...) | |  |  | |
|  | Python math.ceil() Method:  Round a number upward to its nearest integer:  # Import math library import math  # Round a number upward to its nearest integer print(math.ceil(1.4)) print(math.ceil(5.3)) print(math.ceil(-5.3)) print(math.ceil(22.6)) print(math.ceil(10.0))  Output:  2 6 -5 23 10  Definition and Usage:  It rounds a number UP to the nearest integer, if necessary, and returns the result. Use the “math.floor()” method to round DOWN.  Syntax:  math.ceil(x)  Parameter Values:   |  |  | | --- | --- | | **Parameter** | **Description** | | *x* | Required. Specifies the number to round up |   Technical Details   |  |  | | --- | --- | | **Return Value:** | An int value, representing the rounded number. | | **Change Log:** | Python 3+ : Returns an int value Python 2.x : Returns a float value. |   Logical Operators in Python with Examples:  These are used to perform operations on values and variables. They are the special symbols that carry out arithmetic and logical computations. The value operated on is known as the Operand.  In Python, logical operators are used on conditional statements (either True or False). They perform Logical AND, Logical OR, and Logical NOT operations.   |  |  |  |  | | --- | --- | --- | --- | | **OPERATOR** | **DESCRIPTION** | **SYNTAX** | **Example** | | and | Returns True if both the operands are true | x and y | x>7 and x>10 | | or | Returns True if either of the operands is true | x or y | x<7 or x>15 | | not | Returns True if the operand is false | not x | not(x>7 and x> 10) |   Truth Table for Logical Operators in Python  Logical AND operator in Python:  It returns True if both the operands are True else it returns False. |
|  |  |
|  | Logical AND operators in Python Examples:  # Python program to demonstrate  # logical and operator  a **=** 10  b **=** 10  c **=** **-**10  **if** a > 0 **and** b > 0:  **print**("The numbers are greater than 0")  **if** a > 0 **and** b > 0 **and** c > 0:      print("The numbers are greater than 0")  **else**:      print("Atleast one number is not greater than 0")  Output:  **The numbers are greater than 0** **Atleast one number is not greater than 0**  # Python program to demonstrate  # logical and operator  a **=** 10  b **=** 12  c **=** 0  **if** a **and** b **and** c:      print("All the numbers have boolean value as True")  **else**:      print("Atleast one number has boolean value as False")  Output:  **Atleast one number has boolean value as False**  Note: if the first expression is True while using the and operator, then the further expressions are not evaluated.  Logical OR operator in Python:  Logical OR operator returns True if either of the operator is True: |
| Conditionals | Logical OR operator in Python Examples:  # Python program to demonstrate  # logical or operator  a **=** 10  b **=** **-**10  c **=** 0  **if** a > 0 **or** b > 0:  **print**("Either of the number is greater than 0")  **else**:  **print**("No number is greater than 0")  **if** b > 0 **or** c > 0:      print("Either of the number is greater than 0")  **else**:      print("No number is greater than 0")  Output:  **Either of the number is greater than 0** **No number is greater than 0**  # Python program to demonstrate  # logical and operator  a **=** 10  b **=** 12  c **=** 0  **if** a **or** b **or** c:      print("Atleast one number has boolean value as True")  **else**:      print("All the numbers have boolean value as False")  Output:  **Atleast one number has boolean value as True**  Note: if the first expression is True while using the or operator, then the further expressions are not evaluated.  Logical NOT operator in Python Examples  This works with a single Boolean value. If the Boolean value is True it returns False and vice-versa  # Python program to demonstrate  # logical not operator  a **=** 10    **if** **not** a:  **print**("Boolean value of a is True")  **if** **not** (a**%**3 **==** 0 **or** a**%**5 **==** 0):      print("10 is not divisible by either 3 or 5")  **else**:  **print**("10 is divisible by either 3 or 5")  Output:  **10 is divisible by either 3 or 5**  Order of Precedence of Logical Operators  Python always evaluates the expression from left to right.  # Python program to demonstrate  # order of evaluation of logical  # operators  **def** order(x):  **print**("Method called for value:", x)  **return** True **if** x > 0 **else** False  a **=** order  b **=** order  c **=** order  **if** a(**-**1) **or** b(5) **or** c(10):  **print**("Atleast one of the number is positive")  Output:  **Method called for value: -1** **Method called for value: 5** **Atleast one of the number is positive**  Conditions:  Python uses Boolean logic to evaluate conditions, here a True or False value is returned when the expression is compared or evaluated.  x = 2  print(x == 2) # prints out True  print(x == 3) # prints out False  print(x < 3) # prints out True  Note here that unlike in assignments where one “=” is used we use two “==” to make the comparison.  Boolean Operators  The “and” and “or” Boolean operators allow building complex Boolean expressions:  name = "John"  age = 23  if name == "John" and age == 23:  print("Your name is John, and you are also 23 years old.")  if name == "John" or name == "Rick":  print("Your name is either John or Rick.")  The “in” operator  This is used to check if a specified object exists within an iterable object container, such as a list:  name = "John"  if name in ["John", "Rick"]:  print("Your name is either John or Rick.")  Python uses indentation instead of brackets to define code blocks. Standard indentation in Python is four spaces, but we can use tabs also, and it needs to be consistent.  statement = False  another\_statement = True  if statement is True:  # do something  pass  elif another\_statement is True: # else if  # do something else  pass  else:  # do another thing  Pass  A statement is evaulated as true if one of the following is correct: 1. The "True" boolean variable is given, or calculated using an expression, such as an arithmetic comparison. 2. An object which is not considered "empty" is passed.  Here are some examples for objects which are considered as empty: 1. An empty string: "" 2. An empty list: [] 3. The number zero: 0 4. The false boolean variable: False  The “is” operator  Here we match the “instance” of the operands unlike the “==” that matches the values.  x = [1,2,3]  y = [1,2,3]  print(x == y) # Prints out True  print(x is y) # Prints out False  The “not” operator:  Using “not” before a Boolean expression inverts it:  print(not False) # Prints out True  print((not False) == (False)) # Prints out False  Python if...else Statement:  The if statement is a conditional statement that is used to execute a block of code only when a specific condition is met.  Take these case for example:   1. If a student scores above **90**, assign grade **A** 2. If a student scores above **75**, assign grade **B** 3. If a student scores above **65**, assign grade **C**   Python if Statement  Syntax:  If condition:  # body of if statement  Here if the condition evaluates to True the body is executed else the body is skipped.  Take note of the indentation when running if blocks.  number = 10  # check if number is greater than 0  if number > 0:  print('Number is positive')  print('This statement always executes') |
|  | Python if…else Statement  In an if statement we use the else clause to execute a body of code if the if condition returns False.  if condition:  # body of if statement  else:  # body of else statement  number = 10  if number > 0:  print('Positive number')  else:  print('Negative number')  print('This statement always executes')  Python if..elif…else Statement  this is used when we need to make a choice between more than two alternative block of code:  if condition1:  # code block 1  elif condition2:  # code block 2  else:  # code block 3  number = 0  if number > 0:  print('Positive number')  elif number <0:  print('Negative number')  else:  print('Zero')  print('This statement is always executed')  Python Nested if Statements:  It is possible to include an if statement inside another if statement.  number = 5  # outer if statement  if number >= 0:  # inner if statement  if number == 0:  print('Number is 0')    # inner else statement  else:  print('Number is positive')  # outer else statement  else:  print('Number is negative') |

From my reading, I looked at Python's general structure and syntax as a programming language. I looked at Python expressions and their types: constant, arithmetic, integral, floating, relational, logical, bitwise, and combinational. I looked at the math module, how to import and use it, and all the available methods in that module. Then, I looked at the math module ceil method for rounding up numbers. I looked at logical operators and their precedence, and then I looked at conditions and how statement blocks are used to manage them.