**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **29-05-2020** | **Name:** | **Dhanya Shetty** |
| **Course:** | **LOGIC DESIGN** | **USN:** | **4AL17EC026** |
| **Topic:** | **1.ANALYSIS OF CLOCKED SEQUENTIAL CIRCUITS.**  **2.DIGITAL CLOCK DESIGN.** | **Semester & Section:** | **6th A** |
| **Github Repository:** | **Dhanya Shetty\_026** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| C:\Users\Hp\Desktop\report\Screenshot_20200529-185747.png  **C:\Users\Hp\Desktop\report\Screenshot_20200529-185842.png**  C:\Users\Hp\Desktop\report\20200529_195525.jpg  C:\Users\Hp\Desktop\report\20200529_195536.jpg |
| |  |  |  | | --- | --- | --- | | **Date:29May2020** |  | **Name: Dhanya Shetty** | | **Course: Python** |  | **USN:4AL17EC026** | | **Topic: section 20**  **Build a personal website with Python** |  | **Semester & Section:6th A** | |
|  |

|  |
| --- |
|  |
|  |
|  |
| **AFTERNOON SESSION DETAILS** | |
| **Image of sessionsC:\Users\Hp\Desktop\report\29maypy1.PNG**  C:\Users\Hp\Desktop\report\29maypy3.PNG  C:\Users\Hp\Desktop\report\29maypy2.PNG  Syntax and semantics  Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use [curly brackets](https://en.wikipedia.org/wiki/Curly_bracket_programming_language) to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than [C](https://en.wikipedia.org/wiki/C_(programming_language)) or [Pascal](https://en.wikipedia.org/wiki/Pascal_(programming_language)).  **Indentation**  Python uses [whitespace](https://en.wikipedia.org/wiki/Whitespace_character) indentation, rather than [curly brackets](https://en.wikipedia.org/wiki/Curly_bracket_programming_language) or keywords, to delimit [blocks](https://en.wikipedia.org/wiki/Block_(programming)). An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block.[[65]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-53-65) Thus, the program's visual structure accurately represents the program's semantic structure.[[1]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-guttag-1) This feature is sometimes termed the [off-side rule](https://en.wikipedia.org/wiki/Off-side_rule), which some other languages share, but in most languages indentation doesn't have any semantic meaning.  **Statements and control flow**  Python's [statements](https://en.wikipedia.org/wiki/Statement_(computer_science)) include :   * The assignment statement (token '=', the equals sign). This operates differently than in traditional [imperative programming](https://en.wikipedia.org/wiki/Imperative_programming) languages, and this fundamental mechanism (including the nature of Python's version of *variables*) illuminates many other features of the language. Assignment in [C](https://en.wikipedia.org/wiki/C_(programming_language)), e.g., x = 2, translates to "typed variable name *x* receives a copy of numeric value 2". The (right-hand) value is copied into an [allocated storage location](https://en.wikipedia.org/wiki/Memory_allocation) for which the (left-hand) [variable name](https://en.wikipedia.org/wiki/Variable_(computer_science)) is the symbolic address. The memory allocated to the variable is large enough (potentially quite large) for the declared [type](https://en.wikipedia.org/wiki/Type_system). In the simplest case of Python assignment, using the same example, x = 2, translates to "(generic) name x receives a [reference](https://en.wikipedia.org/wiki/Pointer_(computer_programming)) to a separate, dynamically allocated [object](https://en.wikipedia.org/wiki/Object_(computer_science)) of numeric (int) type of value 2." This is termed *binding* the name to the object. Since the name's storage location doesn't *contain* the indicated value, it is improper to call it a *variable*. Names may be subsequently rebound at any time to objects of greatly varying types, including strings, procedures, complex objects with data and methods, etc. Successive assignments of a common value to multiple names, e.g., x = 2; y = 2; z = 2 result in allocating storage to (at most) three names and one numeric object, to which all three names are bound. Since a name is a generic reference holder it is unreasonable to associate a fixed [data type](https://en.wikipedia.org/wiki/Type_system) with it. However at a given time a name will be bound to *some* object, which **will** have a type; thus there is [dynamic typing](https://en.wikipedia.org/wiki/Dynamic_type). * The [if](https://en.wikipedia.org/wiki/If-then-else) statement, which conditionally executes a block of code, along with else and elif (a contraction of else-if). * The [for](https://en.wikipedia.org/wiki/Foreach#Python) statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block. * The [while](https://en.wikipedia.org/wiki/While_loop#Python) statement, which executes a block of code as long as its condition is true. * The [try](https://en.wikipedia.org/wiki/Exception_handling_syntax#Python) statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses; it also ensures that clean-up code in a finally block will always be run regardless of how the block exits. * The raise statement, used to raise a specified exception or re-raise a caught exception. * The class statement, which executes a block of code and attaches its local namespace to a [class](https://en.wikipedia.org/wiki/Class_(computer_science)), for use in [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming). * The def statement, which defines a [function](https://en.wikipedia.org/wiki/Function_(computing)) or [method](https://en.wikipedia.org/wiki/Method_(computing)). * The with statement, from Python 2.5 released in September 2006,[[66]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-66) which encloses a code block within a context manager (for example, acquiring a [lock](https://en.wikipedia.org/wiki/Lock_(computer_science)) before the block of code is run and releasing the lock afterwards, or opening a [file](https://en.wikipedia.org/wiki/Computer_file) and then closing it), allowing [Resource Acquisition Is Initialization](https://en.wikipedia.org/wiki/Resource_acquisition_is_initialization) (RAII)-like behavior and replaces a common try/finally idiom.[[67]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-67) * The [break](https://en.wikipedia.org/wiki/Break_statement) statement, exits from the loop. * The continue statement, skips this iteration and continues with the next item. * The pass statement, which serves as a [NOP](https://en.wikipedia.org/wiki/NOP_(code)). It is syntactically needed to create an empty code block. * The [assert](https://en.wikipedia.org/wiki/Assertion_(programming)) statement, used during debugging to check for conditions that ought to apply. * The yield statement, which returns a value from a [generator](https://en.wikipedia.org/wiki/Generator_(computer_programming)#Python) function. From Python 2.5, yield is also an operator. This form is used to implement [coroutines](https://en.wikipedia.org/wiki/Coroutine" \o "Coroutine). * The import statement, which is used to import modules whose functions or variables can be used in the current program. There are three ways of using import: import <module name> [as <alias>] or from <module name> import \* or from <module name> import <definition 1> [as <alias 1>], <definition 2> [as <alias 2>] * The print statement was changed to the print() function in Python 3.   Python does not support [tail call](https://en.wikipedia.org/wiki/Tail_call) optimization or [first-class continuations](https://en.wikipedia.org/wiki/First-class_continuations), and, according to Guido van Rossum, it never will.[[68]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-55-68)[[69]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-56-69) However, better support for [coroutine](https://en.wikipedia.org/wiki/Coroutine)-like functionality is provided in 2.5, by extending Python's [generators](https://en.wikipedia.org/wiki/Generator_(computer_programming)).[[70]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-57-70) Before 2.5, generators were [lazy](https://en.wikipedia.org/wiki/Lazy_evaluation) [iterators](https://en.wikipedia.org/wiki/Iterator); information was passed unidirectionally out of the generator. From Python 2.5, it is possible to pass information back into a generator function, and from Python 3.3, the information can be passed through multiple stack levels.[[71]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-58-71)  **Expressions**[[edit](https://en.wikipedia.org/w/index.php?title=Python_(programming_language)&action=edit&section=6)]  Some Python [expressions](https://en.wikipedia.org/wiki/Expression_(computer_science)) are similar to languages such as [C](https://en.wikipedia.org/wiki/C_(programming_language)) and [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), while some are not:   * Addition, subtraction, and multiplication are the same, but the behavior of division differs. There are two types of divisions in Python. They are floor division (or integer division) // and floating point/division.[[72]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-72) Python also added the \*\* operator for exponentiation. * From Python 3.5, the new @ infix operator was introduced. It is intended to be used by libraries such as [NumPy](https://en.wikipedia.org/wiki/NumPy) for [matrix multiplication](https://en.wikipedia.org/wiki/Matrix_multiplication).[[73]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-PEP465-73)[[74]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-Python3.5Changelog-74) * From Python 3.8, the syntax :=, called the 'walrus operator' was introduced. It assigns values to variables as part of a larger expression.[[75]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-Python3.8Changelog-75) * In Python, == compares by value, versus Java, which compares numerics by value[[76]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-76) and objects by reference.[[77]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-77) (Value comparisons in Java on objects can be performed with the equals() method.) Python's is operator may be used to compare object identities (comparison by reference). In Python, comparisons may be chained, for example a <= b <= c. * Python uses the words and, or, not for its boolean operators rather than the symbolic &&, ||, ! used in Java and C. * Python has a type of expression termed a [*list comprehension*](https://en.wikipedia.org/wiki/List_comprehension#Python). Python 2.4 extended list comprehensions into a more general expression termed a [*generator*](https://en.wikipedia.org/wiki/Generator_(computer_programming))*expression*.[[54]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-59-54) * [Anonymous functions](https://en.wikipedia.org/wiki/Anonymous_function) are implemented using [lambda expressions](https://en.wikipedia.org/wiki/Lambda_(programming)); however, these are limited in that the body can only be one expression. * Conditional expressions in Python are written as x if c else y[[78]](https://en.wikipedia.org/wiki/Python_(programming_language)" \l "cite_note-AutoNT-60-78) (different in order of operands from the [c ? x : y](https://en.wikipedia.org/wiki/%3F:) operator common to many other languages). * Python makes a distinction between [lists](https://en.wikipedia.org/wiki/List_(computer_science)) and [tuples](https://en.wikipedia.org/wiki/Tuple). Lists are written as [1, 2, 3], are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be [immutable](https://en.wikipedia.org/wiki/Immutable) in Python). Tuples are written as (1, 2, 3), are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The + operator can be used to concatenate two tuples, which does not directly modify their contents, but rather produces a new tuple containing the elements of both provided tuples. Thus, given the variable t initially equal to (1, 2, 3), executing t = t + (4, 5) first evaluates t + (4, 5), which yields (1, 2, 3, 4, 5), which is then assigned back to t, thereby effectively "modifying the contents" of t, while conforming to the immutable nature of tuple objects. Parentheses are optional for tuples in unambiguous contexts.[[79]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-79) * Python features *sequence unpacking* wherein multiple expressions, each evaluating to anything that can be assigned to (a variable, a writable property, etc.), are associated in the identical manner to that forming tuple literals and, as a whole, are put on the left hand side of the equal sign in an assignment statement. The statement expects an *iterable* object on the right hand side of the equal sign that produces the same number of values as the provided writable expressions when iterated through, and will iterate through it, assigning each of the produced values to the corresponding expression on the left.[[80]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-80) * Python has a "string format" operator %. This functions analogous to [printf](https://en.wikipedia.org/wiki/Printf_format_string" \o "Printf format string) format strings in [C](https://en.wikipedia.org/wiki/C_(programming_language)), e.g. "spam=%s eggs=%d" % ("blah", 2) evaluates to "spam=blah eggs=2". In Python 3 and 2.6+, this was supplemented by the format() method of the str class, e.g. "spam={0} eggs={1}".format("blah", 2). Python 3.6 added "f-strings": blah = "blah"; eggs = 2; f'spam={blah} eggs={eggs}'.[[81]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-pep-0498-81) * Python has various kinds of [string literals](https://en.wikipedia.org/wiki/String_literal):   + Strings delimited by single or double quote marks. Unlike in [Unix shells](https://en.wikipedia.org/wiki/Unix_shell), [Perl](https://en.wikipedia.org/wiki/Perl) and Perl-influenced languages, single quote marks and double quote marks function identically. Both kinds of string use the backslash (\) as an [escape character](https://en.wikipedia.org/wiki/Escape_character). [String interpolation](https://en.wikipedia.org/wiki/String_interpolation) became available in Python 3.6 as "formatted string literals".   + Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like [here documents](https://en.wikipedia.org/wiki/Here_document) in shells, Perl and [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)).   + [Raw string](https://en.wikipedia.org/wiki/Raw_string) varieties, denoted by prefixing the string literal with an r. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as [regular expressions](https://en.wikipedia.org/wiki/Regular_expression) and [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows)-style paths. Compare "@-quoting" in [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)). * Python has [array index](https://en.wikipedia.org/wiki/Array_index) and [array slicing](https://en.wikipedia.org/wiki/Array_slicing) expressions on lists, denoted as a[key], a[start:stop] or a[start:stop:step]. Indexes are [zero-based](https://en.wikipedia.org/wiki/Zero-based), and negative indexes are relative to the end. Slices take elements from the *start* index up to, but not including, the *stop* index. The third slice parameter, called *step* or *stride*, allows elements to be skipped and reversed. Slice indexes may be omitted, for example a[:] returns a copy of the entire list. Each element of a slice is a [shallow copy](https://en.wikipedia.org/wiki/Shallow_copy).   In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as [Common Lisp](https://en.wikipedia.org/wiki/Common_Lisp), [Scheme](https://en.wikipedia.org/wiki/Scheme_(programming_language)), or [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)). This leads to duplicating some functionality. For example:   * [List comprehensions](https://en.wikipedia.org/wiki/List_comprehensions) vs. for-loops * [Conditional](https://en.wikipedia.org/wiki/Conditional_(programming)) expressions vs. if blocks * The eval() vs. exec() built-in functions (in Python 2, exec is a statement); the former is for expressions, the latter is for statements.   Statements cannot be a part of an expression, so list and other comprehensions or [lambda expressions](https://en.wikipedia.org/wiki/Lambda_(programming)), all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as a = 1 cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator = for an equality operator == in conditions: if (c = 1) { ... } is syntactically valid (but probably unintended) C code but if c = 1: ... causes a syntax error in Python. | |
|  | |