**CSSE1001: Question Type Revision**

[**UQAttic**](http://uqattic.net)

[**Other exam papers**](https://drive.google.com/open?id=0B_gnBD5zp3rqT2hrLWhvd2Rkc1U)

Please **contribute** to these documents.

If you're looking for an effective way to familiarise yourself with the course material, you can't go past collaborating with fellow students. We have laboured to put these up, and so at the very least point out where you think we are wrong!

You'll get more out of the course, you'll do better in the exam, and other students will benefit from your input as well.

To get editing permissions, visit [UQAttic.net](http://uqattic.net) and provide us with your Google Account address.

**Style.**

Type answers in blue beneath each question.

If you're unsure of your answer, highlight your answer text then hit Ctrl+Alt+M to create a comment beside the text. Once you're satisfied with the answer, click the "Resolve" button on the comment.

If you want some extra explanation from someone else on their answer, highlight the other person's answer and repeat the procedure above.

**Communicate.**

Head over to [uqattic.net](http://uqattic.net/) and click "Chat Now!". You'll find a chatroom full of students just like you. Talk about a revision document (like this one) or swap prep tips. If you have your own IRC client, point it to irc.uqattic.net, port 6667, channel #attic. uwu

# Chances of each type of question appearing

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | (Estimate) Number of questions appeared | | | | | | | | | |
|  | Basic calculation | Type operation | Value of variable | Output | Describe code | Best simplify | Code fragment | Polymorphism | Tkinter | Big O Notation |
| 2019-1 | 2 | 2 | 8 | 4 | 1 | 2 | 6 | 4 | 2 | 3 |
| 2018-2 | 4 | 2 | 7 | 1 | 1 | 0 | 6 | 5 | 2 | 2 |
| 2018-1 | 4 | 2 | 6 | 1 | 2 | 0 | 6 | 5 | 2 | 2 |
| 2017-2 | 1 | 4 | 17 | 0 | 0 | 0 | 6 | 5 | 0 | 2 |
| 2017-1 | 3 | 2 | 6 | 0 | 0 | 1 | 4 | 5 | 2 | 3 |
| 2016-2 | 1 | 4 | 15 | 0 | 0 | 0 | 6 | 5 | 2 | 2 |
| 2016-1 | 1 | 3 | 11 | 0 | 0 | 0 | 6 | 5 | 2 | 0 |
| Chance of appearing (%) | 100% | 100% | 100% | 90%(look at dat trend) | 42% | 29% | 100% | 100% | 85% | 90% (trend) |

**Anyone can help** fill out definitions or give an example (+answer) to any of the question types, in any colour other than black. Please DO NOT modify any of the original definitions/examples unless you have found a fundamental flaw to it. This is the edit link: https://docs.google.com/document/d/19unVcFsOx5KAjz2jHmI\_OfbQiufg5jibzp2HXVeWFBo/edit?usp=sharing

## Basic Calculation

|  |  |
| --- | --- |
| Python can do a few operations with integers and floats. Such as   1. + 2. - 3. \* 4. \*\* 5. / 6. // 7. %   Keep in mind python follows BOMDAS.  For any operation other than / operation,  If **at least one side** of the expression is a **float**, then the **resulting** **type** will also be a **float**. Otherwise **if** **both** a and b are **integers**, then the **result** would also be an **integer**.  For a / b:   * The result would ALWAYS BE float   For a // b:   * The result would only be an integer if both sides are integer   For example:  (2.0 \*\* 3) % 3 = 8.0 % 3  = 2.0 (float) |  |

## Type Operation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Strings:  They can only add amongst themselves, or multiply with **integers**. These are the only 2 conditions you can perform an operation on a string.  Condition 1 (string + string):   * “abc” + “def” = “abcdef”   Condition 2 (string \* integer):   * “abc” \* 3 = “abcabcabc”   ANY OTHER OPERATIONS WILL RESULT IN ERROR   * “abc” - “b” = ERROR * “dog” \* 2.0 = ERROR * “cat”\*\*1 = ERROR * “okboomer” / 8 = ERROR   List:  Same as string, either add amongst themselves, or multiply with integers.   * [a, b] + [c, d] = [a, b, c, d]   Tuple:  Same as string and list, either add amongst themselves, or multiply with integers.   * (a,) + (b, c) = (a, b, c) * a, + b, + c, = (a, b, c) * (a, b) \* 2 = (a, b, a, b) | Sequence Splicing:  We are going to focus on strings, lists and tuples. Please think string as a sequence made up by many characters.  In order to splice a sequence, we need to first type the original value (variable or data) then followed by a pair of [ ].  And the order of splicing goes like this:  Sequence[start:end:step]  Starting at “start” index, ending at “end” (but no including end) and choosing every “step” index.  Examples:   |  |  | | --- | --- | | [1, 2, 3][0] = 1  [1, 2, 3][2] = 3  [1, 2, 3][-2] = 2  a = [1, 2, 3, 4, 5, 6, 7]  a[2:] = [3, 4, 5, 6, 7]  a[:4] = [1, 2, 3, 4] | a[2:4] = [3, 4]  a[1:5:2] = [2,4]  a[:] = [1, 2, 3, 4, 5, 6, 7]  a[::] = [1, 2, 3, 4, 5, 6, 7]  a[::-1] = [7, 6, 5, 4, 3, 2, 1]] |   Note for for both “value[:]” and “value[::]”, it will create a new set of values which is the exact copy of the original value. But if you assign it to another variable, it will not gain the access of this variable.   |  |  | | --- | --- | | a = [1,2,3]  b = a  b[0] = 5  (now “variable a” == [5, 2, 3]) | a = [1, 2, 3]  b = a[:]  b[0] = 5  (“variable a” is not changed) | |

## Value of Variable

|  |  |
| --- | --- |
| Variable names in Python can contain letters (both upper and lowercase), numbers and the underscore (\_) symbol with the restriction they cannot begin with a number.  The type of a variable returned from `input()` is a string and therefore if you want to use it as a number it must be cast as such. For an integer this could be `int(input())`.  To assign an integer value to a variable it is enough to write `variable\_name = 5`. To assign values to a dictionary the dictionary must be declared first:  *d = {}*  *d[“key”] = “value”*  However if we already have a dictionary object we can copy it.  *old\_dict = {1: 2}*  *new\_dict = old\_dict*  However this copy points to the same object that the original points to and thus any changes made to the old or new dictionary will be reflected in the other. To make a separate copy we can run `new\_dict = dict(old\_dict)` or `new\_dict = old\_dict.copy()`. | Lists:  .append() and .extend() does not return anything, it simply does the job.   * print([].append(1), [].extend(“abc”))   Output would be None None.  Copy by reference:  Types such as lists and dictionaries can be copied by reference.  Which means if you assign it to a new variable, and you make changes to the new variable, the old variable will be affected as well. But if the new variable loses the reference, it will stop modifying the old variable. |

## Output

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| There are a few traps to determine the output of the program. Such as the distinction between no output and a new line, and the type of data that the question is trying to do operation on. For example:    This is data type kind of question, where you kind of have to know what variable type is x and y. In this case, both of them are strings, so the answer is “2” + “5” = “25”.  ------------> | This is more of a trick question rather than counting the actual amount of stars. As 0 > 0 == False, we will not enter a loop, therefore no print statement will be called, so there will not be any output  For the sake of learning, let x = 5 for line 1 and try to work out the answer.  We know there will be 5 iterations of the same loop.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Before loop | Loop 1  x = 5 | Loop 2  x = 4 | Loop 3  x = 3 | Loop 4  x = 2 | Loop 5  x = 1 | | x = 5  stars = “\*” | print(star)  stars = “\*” + “\*”  x = 5 - 1 | print(star)  stars = “\*\*” + “\*”  x = 4 - 1 | print(star)  stars = “\*\*\*” + “\*”  x = 3 - 1 | print(star)  stars = “\*\*\*\*” + “\*”  x = 2 - 1 | print(star)  stars = “\*\*\*\*\*” + “\*”  x = 1 - 1 |   So by the end of the loop:   * x = 0 * stars = “\*\*\*\*\*\*”   And our output would look a little like this:  \*  \*\*  \*\*\*  \*\*\*\*  \*\*\*\*\*  Notice that there are only at max 5 stars printed, because the print statement execute before the string addition. |

## 

## Describe Code

|  |  |
| --- | --- |
| Just a general rule of thumb for describing code, use the method of elimination rather than trying to deduce the real answer. I’ll just give example to this type of question.    It cannot be (A) because we are able to enter the loop if the input is anything but 0.  It cannot be (B) because we can exit the loop by inputting 0.  It cannot be (C) because it would not have an if condition.  At this point we can see that it must be (E) because r % 2 == 0 is only True when r is even.  So the answer is (E) |  |

## Best Simplify

|  |  |
| --- | --- |
| This type of question picks a few options out of these.   * For loop * While loop * If statement * Function * Class   There is not much to this other than just really understanding the advantages and disadvantages of them.  For loops:   * Good for iterating through sequences * Executing a set of commands with each iteration   While loops:   * Execute command as long as condition is true * Good for dealing with unknown length (user input)   If statement:   * Decision making, pretty self-explanatory   Function:   * A stored block of code which accepts input (and can give output) and executes commands * Good for dealing with repeated action/block of code.   Class:   * “A Class is like an object constructor, or a "blueprint" for creating objects.” (w3schools) * Some common objects:   + Students   + Vehicles   + Employer/employee   + Animal | I am not going to finish this guide in time \*)!(@\*)(!\* Ree ree ree ree ree |

## Code fragment

## Dictionary

## Polymorphism

## Tkinter

## Big O Notation