COMP10001 Foundations of Computing
Semester 1, 2022

Tutorial Questions: Week 4

— VERSION: 1657, DATE: MARCH 21, 2022 —

Discussion

1. What is "Boolean"? What values does it store? Can other types be converted to it?

   A: *Boolean is a data type which stores a truth value: either True or False. Any other type can be converted to it: for numbers, zero converts to False and any non-zero number converts to True; and for strings and lists an empty sequence converts to False and any non-empty sequence converts to True.*

2. For each of the following, identify whether it is: (a) a Boolean value; (b) a relational operator; or (c) a logical operator.

   A: *The relational operators compare two values to produce a truth (boolean) result. They include less than (<), greater than (>), less than or equal to (<=), greater than or equal to (>=), equal (==) and not equal (!=) The* `in` *operator is also a kind of relational operator.*
   The logical operators combine Boolean values to return a single truth value. They are `and`*,* `or` *and* `not`*.*
   Order of precedence is Relational operators then `not`*,* `and`*, and finally* `or`*. Brackets can clarify order of operations and you are encouraged to use them.*

`==`	Relational	`>`	Relational	`False`	Boolean
`!=`	Relational	`and`	Logical	`<=`	Relational
`or`	Logical	`>=`	Relational	`not`	Logical
`True`	Boolean	`<`	Relational		

3. How do we use an if statement? What are the variants? How do we know what is contained inside it and what is after?

   A: *An if statement is of the form* `if <condition>:` *where* `condition` *can be formed of relational and logical operators, or anything else you like which can be converted to a boolean value. If the condition expression evaluates to True, the code "inside" (indented after) the if statement is run. If not, it is skipped. Variants include* `elif` *and* `else` *statements following if, which catch further conditions if the first conditional statement is not fulfilled. Indentation tells us what code belongs inside an if statement and what code follows it.*

   Now try Exercises 1–4

4. What is a "Sequence"? What sequences have we seen so far? What operators can we use with sequences?

   A: *A sequence is a data type which allows us to store a series of objects in a particular order. Strings store sequences of characters while lists and tuples store sequences of any object. We can use the * operator to create copies of the sequence and concatenate them. We can use the + operator to concatenate sequences together.*

5. What is indexing? How can you do it?

   A: *Indexing means accessing the item stored in a particular (integer) position in a sequence. You index using square brackets containing the index* `[i]` *at the end of the variable name or object literal. You can index with positive integers where 0 corresponds to the first item or negative integers where -1 corresponds to the last item.*

6. What is slicing? How can you do it?

   A: *Slicing is like taking an index but rather than only one item, a slice will give you a subsection of the sequence consisting typically of more than one item (though you can have an empty slice). Slicing works with two indices* `[i:j]` *and takes items from the first to* ***one before*** *the second. A slice can take three integers: start, end and step [i:j:k]. The step integer controls the slice step size; Whether every character is included in the slice (k=1), every second character (k=2), and so on. A negative step makes the slice go backwards.*

   Now try Exercise 5

7. What is a "function"? How do we call (use) one? How do we define one ourselves?

   A: *A function is a block of reusable code which is defined once and can be reused wherever you would like in a program. A function is called by writing its name and then a pair of brackets. Depending on the function, you can list arguments inside the brackets, separated by commas. To define a function, we use the def keyword followed by the function name, parameter(s) in brackets, a colon and then the function body, indented.*

8. What does it mean to "return" a value from a function and why would we want to? Does a function always need a return value?

   A: *Returning a value from a function makes it available to the line of code which called the function, so that it can be, for example, assigned to a variable. This return value could be the result of a calculation or a status message or something else. A function doesn't need to have a return value: it could just perform an operation such as* `print()`*. You return with the return keyword, followed by the value you want to pass. At this point, the function execution terminates.*

9. Why should we use functions? Could we live without functions?

   A: *Functions reduce code duplication by allowing us to run the same block of code multiple times. This saves our code from becoming long and repetitive, and makes it easier to edit in the future. It also allows code we write to be used elsewhere, creating more modular code which we can use in more places. We could theoretically live without functions, but code would be very messy and error-prone without them.*

10. Why are brackets important when calling a function? Are they needed even if it takes no arguments?

    A: *Brackets are the difference between a reference to the name of the function and an actual call of the function. Even if there are no arguments, brackets are still needed to call the function (they will simply be empty).*

Now try Exercise 6

Exercises

1. Evaluate the following truth expressions:

   (a) `True or False`
   A: `True`

   (b) `True and False`
   A: `False`

   (c) `False and not False or True`
   A: `True`

   (d) `False and (not False or True)`
   A: `False`

2. For each of the following if statements, give an example of a value for `var` which will trigger it and one which will not.

   (a) `if 10 > var >= 5:`

   A: *A number 5 or greater and lower than 10 will trigger this if statement. A number outside this range will not. A value other than a number will cause a TypeError.*

   (b) `if var[0] == "A"and var[-1] == "e":`

   A: *A string which begins with "A" and ends with "e", for example* `"Apple"`*,* `"Antelope"` *or* `"Ae"` *(with that capitalisation). A list or tuple with* `"A"` *and* `"e"` *as the first and last items will also pass. Other strings and sequences will not trigger the conditional, and any sequence which does not have at least one one item will give an IndexError. A non-sequence type will produce a TypeError.*

   (c) `if var in ["VIC", "NSW", "ACT"]:`

   A: *Only strings "VIC", "NSW" and "ACT" will trigger this condition. Any other value of* `var` *will result in a False result.*

   (d) `if var:`

   A: *This condition will convert* `var` *into a boolean value, so if it is non-zero/non-empty, the if statement will be triggered.*

3. What's wrong with this code? How can you fix it?

```
letter = input("Enter a letter: ")
if letter == 'a' or 'e' or 'i' or 'o' or 'u':
    print("vowel")
else:
    print("consonant")
```

   A: *Logical operators separate conditions, so* `letter == 'a'` *will be evaluated completely separately from* `'e'` *and the rest of the conditions. Strings with one character will always evaluate to True so the logical statement* `letter == 'a'or True or True or True or True` *will always evaluate to True which is undesired. This can be fixed by writing out* `letter ==` *for every condition; or by using the* `in` *operator as* `if letter in 'aeiou':` *or better,* `if letter in ('a', 'e', 'i', 'o', 'u'):`.

4. What's wrong with this code? How can you fix it?

```
eggs == 3
if eggs = 5:
    print("spam")
else:
    print("not spam")
```

A: *This programmer has confused the assignment (=) and equality (==) operators. This can be fixed by swapping them (`eggs = 3`, `if eggs == 5:`).*

5. Evaluate the following given the assignment `s = "python"`

(a) `s[1]`
A: `'y'` *(Indexing starts from 0)*

(b) `s[-1]`
A: `'n'` *(Negative indexing starts from -1)*

(c) `s[1:3] + s[3:5]`
A: `'ytho'`

(d) `s[10]`
A: `IndexError: string index out of range`

(e) `s[10:]`
A: `''` *(Empty String)*

(f) `s[-4:-2]`
A: `'th'` *(Note that even with negative indices, the index of the left-most character is first)*

(g) `s[:-4]`
A: `'py'`

(h) `s[::2]`
A: `'pto'` *(skips every second letter)*

(i) `s[::-1]`
A: `'nohtyp'` *(backwards)*

A: *Note that both d and e ask for a portion of the string which doesn't exist, but while d (indexing) gave an error, e (slicing) returned an empty sequence.*

6. What's wrong with this code? How can you fix it?

```
def calc(n1, n2):
    answer = n1 + (n1 * n2)
    print(answer)

num = int(input("Enter the second number: "))
result = calc(2, num)
print("The result is:", result)
```

A: *This function prints the answer to the calculation it's performed rather than returning it. This means that the value of `result` will be `None` and the last line will not work as intended.*

Problems

1. Write a function which takes a string as a single argument, and returns a shortened version of the string consisting of its first three letters and then every second letter in the rest of the word. `shorten('uncalled')` should return `'uncle'`.

   A:

   ```
   def shorten(word):
       short_word = word[:3] + word[4::2]
       return short_word
   ```

2. Write a function which takes a sentence as a single argument (in the form of a string), and evaluates whether it is valid based on whether the first letter is capitalised and the last character is a full stop. It should return a Boolean value `True` or `False`.

   A:

   ```
   def check(sentence):
       if "A" <= sentence[0] <= "Z" and sentence[-1] == ".":
            return True
       else:
           return False
   ```

   or

   ```
   def check(sentence):
       return "A" <= sentence[0] <= "Z" and sentence[-1] == "."
   ```

   or

   ```
   def check(sentence):
       return sentence[0].isupper() and sentence[-1] == "."
   ```

3. Write a program which asks the user for two numbers and an operator out of +, −, / and ⋆ and performs that operation on the two numbers, printing the result.

   A:

   ```
   num_1 = float(input("Enter␣the␣first␣number:␣"))
   num_2 = float(input("Enter␣the␣second␣number:␣"))
   op = input("Enter␣operator␣(+␣-␣*␣/)␣")
   if op == "+":
       print(num_1, "+", num_2, "=", num_1 + num_2)
   elif op == "-":
       print(num_1, "-", num_2, "=", num_1 - num_2)
   elif op == "*":
       print(num_1, "*", num_2, "=", num_1 * num_2)
   else:
       print(num_1, "/", num_2, "=", num_1 / num_2)
   ```