Week 4 Video Programs

**What is a string?**

Modify the double\_word function so that it returns the same word repeated twice, followed by the length of the new doubled word. For example, double\_word("hello") should return hellohello10.



**def double\_word(word):**

**return (word+word+str(len(word+word)))**

**print(double\_word("hello")) # Should return hellohello10**

**print(double\_word("abc")) # Should return abcabc6**

**print(double\_word("")) # Should return 0**

**Here is your output:**

**hellohello10**

**abcabc6**

**0**

**Great work! You're getting comfortable using some of the**

**basic string operations. You'll soon be working on even more**

**interesting tasks!**

#### The Parts of a String

Want to give it a go yourself? Be my guest! Modify the first\_and\_last function so that it returns True if the first letter of the string is the same as the last letter of the string, False if they’re different. Remember that you can access characters using message[0] or message[-1]. Be careful how you handle the empty string, which should return True since nothing is equal to nothing.



**def first\_and\_last(message):**

**if not message:**

**return True**

**return message[0] == message[-1]**

**print(first\_and\_last("else"))**

**print(first\_and\_last("tree"))**

**print(first\_and\_last(""))**

**Here is your output:**

**True**

**False**

**True**

**Nice job! You've specified the correct condition for empty**

**strings, and** **used appropriate string indexing to get the**

**work done.**

#### Creating New Strings

Try using the index method yourself now!

Using the index method, find out the position of "x" in "supercalifragilisticexpialidocious".



**word = "supercalifragilisticexpialidocious"**

**print(word.index('x'))**

**Here is your output:**

**21**

**Right on! The index method is a very useful way of working**

**with strings, and not having to do the hard work manually.**

#### More String Methods

Want to try some string methods yourself? Give it a go!

Fill in the gaps in the initials function so that it returns the initials of the words contained in the phrase received, in upper case. For example: "Universal Serial Bus" should return "USB"; "local area network" should return "LAN”.



**def initials(phrase):**

**words = phrase.split(" ")**

**result = ""**

**for word in words:**

**result += word[0]**

**return result.upper()**

**print(initials("Universal Serial Bus")) # Should be: USB**

**print(initials("local area network")) # Should be: LAN**

**print(initials("Operating system")) # Should be: OS**

**RunReset**

**Here is your output:**

**USB**

**LAN**

**OS**

**Awesome! You've filled in the correct string methods and**

**remembered how to use them! We’re getting through all this**

**stuff really fast now -- great job!**

#### Formatting Strings

Modify the student\_grade function using the format method, so that it returns the phrase "X received Y% on the exam". For example, student\_grade("Reed", 80) should return "Reed received 80% on the exam".



**def student\_grade(name, grade):**

**return (str(name)+" "+"received"+" "+str(grade)+"%"+" "+"on the exam")**

**print(student\_grade("Reed", 80))**

**print(student\_grade("Paige", 92))**

**print(student\_grade("Jesse", 85))**

**Here is your output:**

**Reed received 80% on the exam**

**Paige received 92% on the exam**

**Jesse received 85% on the exam**

**You got it! Your string formatting skills are coming along**

**nicely!**

#### What is a list?

Using the "split" string method from the preceding lesson, complete the get\_word function to return the {n}th word from a passed sentence. For example, get\_word("This is a lesson about lists", 4) should return "lesson", which is the 4th word in this sentence. Hint: remember that list indexes start at 0, not 1.



**def get\_word(sentence, n):**

**# Only proceed if n is positive**

**if n > 0:**

**words = sentence.split(" ")**

**# Only proceed if n is not more than the number of words**

**if n <= len(words):**

**return(words[n-1])**

**return("")**

**print(get\_word("This is a lesson about lists", 4)) # Should print: lesson**

**print(get\_word("This is a lesson about lists", -4)) # Nothing**

**print(get\_word("Now we are cooking!", 1)) # Should print: Now**

**print(get\_word("Now we are cooking!", 5)) # Nothing**

**RunReset**

**Here is your output:**

**lesson**

**Now**

**Excellent! You're getting comfortable with string**

**conversions into lists. Now we are really cooking!**

#### Modifying the Contents of a List

The skip\_elements function returns a list containing every other element from an input list, starting with the first element. Complete this function to do that, using the for loop to iterate through the input list.



**def skip\_elements(elements):**

**# Initialize variables**

**new\_list = []**

**i = 0**

**# Iterate through the list**

**for i in range(0,len(elements),2):**

**# Does this element belong in the resulting list?**

**if i%2==0:**

**# Add this element to the resulting list**

**new\_list.append(elements[i])**

**return new\_list**

**print(skip\_elements(["a", "b", "c", "d", "e", "f", "g"])) # Should be ['a',**

**'c', 'e', 'g']**

**print(skip\_elements(['Orange', 'Pineapple', 'Strawberry', 'Kiwi', 'Peach']))**

**# Should be ['Orange', 'Strawberry', 'Peach']**

**print(skip\_elements([])) # Should be []**

**Here is your output:**

**['a', 'c', 'e', 'g']**

**['Orange', 'Strawberry', 'Peach']**

**[]**

**You nailed it! Isn't all this list manipulation so much fun?**

#### Lists and Tuples

Let's use tuples to store information about a file: its name, its type and its size in bytes. Fill in the gaps in this code to return the size in kilobytes (a kilobyte is 1024 bytes) up to 2 decimal places.



**def file\_size(file\_info):**

**fname, ftype, fsize= file\_info**

**return("{:.2f}".format(fsize / 1024))**

**print(file\_size(('Class Assignment', 'docx', 17875))) # Should print 17.46**

**print(file\_size(('Notes', 'txt', 496))) # Should print 0.48**

**print(file\_size(('Program', 'py', 1239))) # Should print 1.21**

**Here is your output:**

**17.46**

**0.48**

**1.21**

**Well done! Aren't tuples handy to keep the information**

**nicely organized for when we need it?**

#### Iterating over Lists and Tuples

Try out the enumerate function for yourself in this quick exercise. Complete the skip\_elements function to return every other element from the list, **this time using the enumerate function** to check if an element is on an even position or an odd position.



**def skip\_elements(elements):**

**new\_list=[]**

**for index,person in enumerate(elements):**

**if index%2==0:**

**new\_list.append(elements[index])**

**return (new\_list)**

**print(skip\_elements(["a", "b", "c", "d", "e", "f", "g"])) # Should be ['a',**

**'c', 'e', 'g']**

**print(skip\_elements(['Orange', 'Pineapple', 'Strawberry', 'Kiwi', 'Peach']))**

**# Should be ['Orange', 'Strawberry', 'Peach']**

**RunReset**

**Here is your output:**

**['a', 'c', 'e', 'g']**

**['Orange', 'Strawberry', 'Peach']**

**Great job! The enumerate function sure makes things easier,**

**doesn't it?**

#### List Comprehensions

The odd\_numbers function returns a list of odd numbers between 1 and n, inclusively. Fill in the blanks in the function, using list comprehension. Hint: remember that list and range counters start at 0 and end at the limit minus 1.



**def odd\_numbers(n):**

**return [x for x in range(1,n+1) if x%2==1]**

**print(odd\_numbers(5)) # Should print [1, 3, 5]**

**print(odd\_numbers(10)) # Should print [1, 3, 5, 7, 9]**

**print(odd\_numbers(11)) # Should print [1, 3, 5, 7, 9, 11]**

**print(odd\_numbers(1)) # Should print [1]**

**print(odd\_numbers(-1)) # Should print []**

**RunReset**

**Here is your output:**

**[1, 3, 5]**

**[1, 3, 5, 7, 9]**

**[1, 3, 5, 7, 9, 11]**

**[1]**

**[]**

#### What is a dictionary?

The "toc" dictionary represents the table of contents for a book. Fill in the blanks to do the following: 1) Add an entry for Epilogue on page 39. 2) Change the page number for Chapter 3 to 24. 3) Display the new dictionary contents. 4) Display True if there is Chapter 5, False if there isn't.



**toc = {"Introduction":1, "Chapter 1":4, "Chapter 2":11, "Chapter 3":25, "Chapter**

**4":30}**

**toc["Epilogue"]=39**

**toc["Chapter 3"]=24**

**print(toc)**

**if toc.get("Chapter 5")!=None:**

**print(True)**

**else:**

**print(False)**

**Here is your output:**

**{'Introduction': 1, 'Chapter 1': 4, 'Chapter 2': 11, 'Chapter 3': 24, 'Chapter 4': 30, 'Epilogue': 39}**

**False**

**Great work! You've made the changes to the dictionary**

**exactly as requested.**

#### Iterating over the Contents of a Dictionary

Now, it's your turn! Have a go at iterating over a dictionary!

Complete the code to iterate through the keys and values of the cool\_beasts dictionary. Remember that the items method returns a tuple of key, value for each element in the dictionary.



**cool\_beasts = {"octopuses":"tentacles", "dolphins":"fins", "rhinos":"horns"}**

**for key,value in cool\_beasts.items():**

**print("{} have {}".format(key,value))**

**Here is your output:**

**octopuses have tentacles**

**dolphins have fins**

**rhinos have horns**

**Nice job! Your dictionary skills are getting stronger and**

**stronger!**

#### Dictionaries vs. Lists

In Python, a dictionary can only hold a single value for a given key. To workaround this, our single value can be a list containing multiple values. Here we have a dictionary called "wardrobe" with items of clothing and their colors. Fill in the blanks to print a line for each item of clothing with each color, for example: "red shirt", "blue shirt", and so on.



**wardrobe = {"shirt":["red","blue","white"], "jeans":["blue","black"]}**

**for key, value in wardrobe.items():**

**for i in value:**

**print("{} {}".format(i, key))**

**Here is your output:**

**red shirt**

**blue shirt**

**white shirt**

**blue jeans**

**black jeans**