**Question 1**

|  |  |  |
| --- | --- | --- |
| Question | Which of the following lines of code will combine the following two dictionaries? |  |
| Option 1 | birth\_places + more\_birth\_places | Incorrect |
| Option 2 | birth\_places.append(more\_birth\_places) | Incorrect |
| Option 3 | birth\_places = {\*birth\_places, \*more\_birth\_places} | Incorrect |
| Option 4 | **birth\_places.update(more\_birth\_places)** | **Correct** |
| Correct Feedback: | Correct! The other options are not valid dictionary operations or methods. | |
| Incorrect Feedback: | Incorrect – please try again. | |
| General Feedback: | While option 4 is the only choice that will combine both dictionaries, Option 3 would produce the same result by adding two \*’s: birth\_places = {\*\*birth\_places, \*\*more\_birth\_places}. | |
| Hint: | Refer to dictionary methods here: <https://docs.python.org/3/library/stdtypes.html#typesmapping> | |

**Question 2**

|  |  |  |
| --- | --- | --- |
| Question | Match the built-in python function to the output it would produce when invoked on arr. |  |
| Image |  |  |
| Option 1 | any(arr) | **True** |
| Option 2 | all(arr) | **False** |
| Option 3 | set(arr) | **{-1, 0, 1}** |
| Option 4 | sorted(arr) | **[-1, 0, 0, 1]** |
| Option 5 | tuple(arr) | **(-1, 0, 1, 0)** |
| Option 6 | sum(arr) | **0** |
| Option 7 | max(arr) | **1** |
| Option 8 | min(arr) | **-1** |
| Option 9 | list(arr) | **[-1, 0, 1, 0]** |
| Option 10 | len(arr) | **4** |
| Correct Feedback: | Correct! |  |
| Incorrect Feedback: | Incorrect, please try again. |  |
| General Feedback: | Python has a number of functions and types built into it that are always available. | |
| Hint: | See description of each function here: <https://docs.python.org/3/library/functions.html> | |

**Question 3**

|  |  |  |
| --- | --- | --- |
| Question | Which of the following statements about the map() function are true? Mark all that apply. | |
| Option 1 | The map() function accepts two arguments. The first is a function and the second is an iterable (the data) | **True** |
| Option 2 | The map() function allows lambda functions as the first parameter. | **True** |
| Option 3 | The map() function returns a list as output | **False** |
| Option 4 | The map() function is useful for applying the same function on every element of an iterable. | **True** |
| Correct Feedback: | Correct! | |
| Incorrect Feedback: | Incorrect – please try again! | |
| General Feedback: | The map() function is a handy tool when you want to apply some function over and over again without maintaining an internal state. Using Python in this way is an example of functional programming: <https://docs.python.org/3/howto/functional.html> | |
| Hint: | Review lesson 3-1 for more on the map() function. The documentation may be helpful as well: <https://docs.python.org/3/library/functions.html#map> | |

**Question 4**

|  |  |  |
| --- | --- | --- |
| Question | Suppose a file named “starwars.csv” is saved in the current working directory. Which of the following lines of code will load the data as a pandas DataFrame and encode the sentinel value “?” as a missing value (np.nan)? | |
| Option 1 | df = pd.read\_csv("starwars.csv") | Incorrect |
| Option 2 | df = pd.read\_csv("starwars.csv", na\_filter=”?”) | Incorrect |
| Option 3 | df = pd.read\_csv("starwars.csv", keep\_default\_na=True) | Incorrect |
| Option 4 |  | Incorrect |
| Option 5 | df = pd.read\_csv("starwars.csv", na\_values=”?”) | **Correct** |
| Correct Feedback: | Correct – the na\_values parameter can be used to add additional strings to recognize as NA/NAN. | |
| Incorrect Feedback: | Incorrect – please try again! See documentation and module 4 for more information on pd.read\_csv. <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html> | |
| General Feedback: | pd.read\_csv is a flexible function for reading data. Using the available parameters can save you a considerable amount of time cleaning data after it’s loaded. | |
| Hint: | See what each of these parameters do in the Pandas docs for pd.read\_csv here: <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html> | |

**Question 5**

|  |  |  |
| --- | --- | --- |
| Question | Assume the Pandas DataFrame below is assigned to the variable df.  Which of the following lines of code will produce a filtered DataFrame with only rows whose ‘username’ is ‘Favio’ and whose ‘fare’ is less than 7 dollars? Mark each line as True or False. | |
| Image |  | |
| Option 1 | df[df[“username”] = “Favio”] | False |
| Option 2 | df[df[“username”] == “Favio”] | **True** |
| Option 3 | df[username == “Favio”] | False |
| Option 4 | df.iloc[df[“username”] == “Favio”, :] | False |
| Option 5 | df.loc[df[“username”] == “Favio”, :] | **True** |
| Correct Feedback: | Correct! | |
| Incorrect Feedback: | Incorrect – please try again! | |
| General Feedback: | There are multiple ways to select data within Pandas DataFrames. The two most common ways are using the column and row labels (.loc) and using the positional indices (.iloc) | |
| Hint: | Review Modules 4 and 6-3 for more on selecting data in Pandas DataFrames. Feel free to reference the documentation here: <https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html> | |