Worksheet #8 More on Strings Answer Key =)

TRUE/FALSE

1. (11 points: 1 point each) The following are true/false questions. **Write either T or F in the boxes at the bottom of page 1.** If there's any counterexample, it's false.

   (a) Characters are a different type from strings in Python.

   (b) If strings s1 and s2 are equivalent, the expression `s1 in s2` will return False because "in" checks for partial matches and not the complete operand length.

   (c) In Python, you can directly modify a character within a string by using the index notation, like `s[2] = "z"`.

   (d) To loop over the indices of string s, you can do `for i in range(len(s) + 1)`.

   (e) Strings in Python are 0-based indexed (i.e., the first character is at index 0).

   (f) The `strip()` method in Python removes both leading and trailing whitespace from a string.

   (g) Python allows you to compare strings using the relational operators `(<, <=, >, >=)` based on their lengths.

   (h) Strings can indexed using negative numbers.

   (i) On string s, `min(s) == max(s)` will always evaluate to False.

   (j) `islower()`, `isdigit()` and `isalpha()` are all functions, so they take a string as a parameter rather than being directly called on a string.

   (k) When `s` is a string, in loops like `for i in s`, `i` will refer to a character of s each iteration.

a	b	c	d	e	f	g	h	i	j	k
F	F	F	F	T	T	F	T	F	F	T

Notes:
(C) Remember that strings are immutable! So this would cause an error.
(D) This will cause an index out of bounds error. Remember that the last character in our string has index len(s) - 1 (ex: "pluto" has 5 characters, indices 0 through 4). A range(len(s) + 1) loop when s = "pluto" would be range(6) and include [0, 1, 2, 3, 4, 5] ← on the last iteration, an error would occur)
(I) counterexample: when s is only one character, its min and max character will be the same.

MULTIPLE CHOICE

2	3	4	5	6	7	8	9
A	A	B	D	D	B	F	C

2. What is the result of `"oink oink" * 2 + " oink"`?

   A. `"oink oinkoink oink oink"`
   B. `"oink oinkoink"`
   C. `"oink oink oink oink"`
   D. Error
   E. None of the above

Correct answer explanation:
(A) Yes, we can multiply strings in Python (so it is not (D), an error)! "oink oink" * 2 will result in "oink oinkoink oink". We then concatenate " oink" onto this, resulting in "oink oinkoink oink oink", resulting in (A).

3. What is the result of `"Teehee".find("e")`?

   A. 1 B. 2 C. 4 D. 5 E. -1

Correct answer explanation:

(A) .find() returns the index of the first occurrence of the specified character (or substring). If there is no occurrence, .find() returns -1. “Teehee” has “e” in it, so the answer cannot be (E) -1. We will return the index of the first “e”, which is (A) 1.

4. Which operator can be used to check if a substring is present in a string?

   A. `contains` B. `in` C. `exists` D. `substring` E. Both A and B

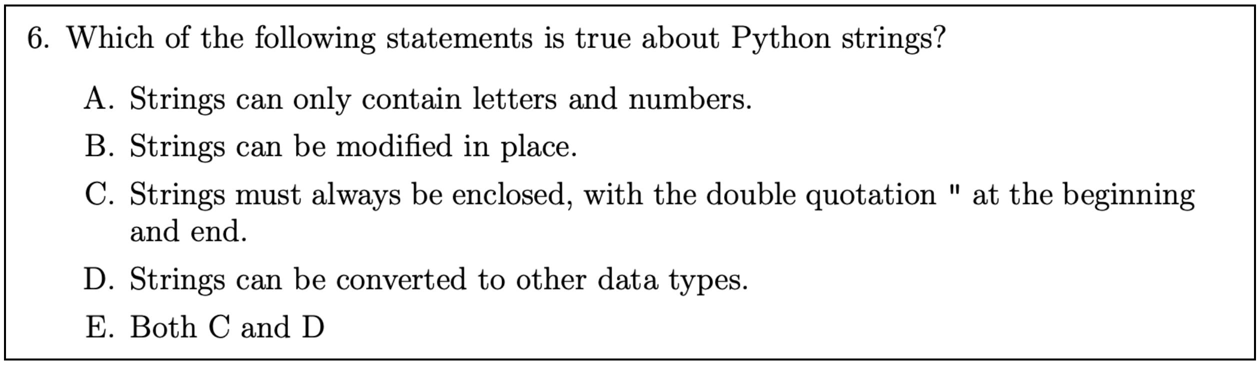
Correct answer explanation:
(B) ‘in’ is used for this purpose. We can do expressions like “‘a’ in ‘abracadabra’”, for instance, to return a boolean.

Wrong answer explanations:
(A) contains is not correct because contains is actually a method (we could do “abracadabra”.contains(“a”)).
(C) and (D) are not real operators.

5. If s = `"Basil and Wybie =)"`, what is the result of the expression `len(s) - len(s.replace(" ", ""))`?

   A. 0 B. 1 C. 2 D. 3 E. None of the above

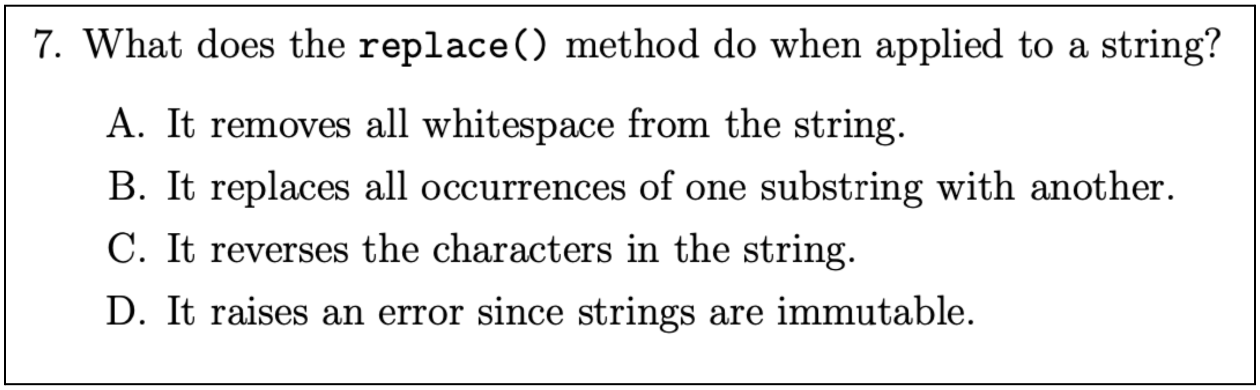
Correct answer explanation:
(D) Given string s, .replace() will substitute out any occurrence of one substring, with another specified substring. In this case, we’re substituting whitespace “ “ with “” (empty string). We know that since the string “Basil and Wybie =)” has three spaces, the length of the new string will be shortened by 3. (for what it’s worth, the new string is simply “BasilandWybie=)” length 15 compared to the original 18).


6. Which of the following statements is true about Python strings?

   A. Strings can only contain letters and numbers.
   B. Strings can be modified in place.
   C. Strings must always be enclosed, with the double quotation `"` at the beginning and end.
   D. Strings can be converted to other data types.
   E. Both C and D

Correct answer explanation:
(D) We also know that strings can be converted to ints by doing something like int("52"), and of the four answer choices A, B, C, D, D is the only truthful answer.

Wrong answer explanations:
(A) Not true! We've had many strings in homework assignments with colons and periods (like, "Enter name: ")
(B) Not true! Remember that strings are immutable, so once they are created they cannot be changed.
(C) is almost correct, but you can use single quotes to enclose a string. For example: "Toodles =)" and 'Toodles =)' are both valid. However, this question asserts that only the former is correct, which is false.
(E) because (C) is incorrect, (E) cannot be correct.


7. What does the `replace()` method do when applied to a string?

   A. It removes all whitespace from the string.
   B. It replaces all occurrences of one substring with another.
   C. It reverses the characters in the string.
   D. It raises an error since strings are immutable.

Correct answer explanation:
(B) Yes! .replace() is a method that requires two arguments: the substring you want to find in a string, and the substring with which you want to replace it. Technically, you can remove all whitespace in a string by doing .replace(" ", ""). That would replace any space with the empty string. But, because .replace() does more than that, (A) still isn't exactly correct.

8. If you want to traverse through a string S using negative indices in Python, which loop should you use?

   A. `for char in S`
   B. `for i in range(len(S))`
   C. `for i in range(-len(S), 0, +1)`
   D. `for i in range(-len(S), 0)`
   E. `for i in range(-1, -len(S)-1, -1)`
   F. C, D, and E
   G. C and E

Correct answer explanation:
(F) Because (C), (D), and (E) are all true, (F) is correct.

(C) 'for i in range(-len(S), 0, +1)' will traverse the string from left to right using negative indices. For example: "howdy", letter:index combos are
h:0, o:1, w:2, d:3, y:4.
But using negative indices, these become
h:-5, o:-4, w:-3, d:-2, y:-1.

So, -len(S) is -5. Thus, range(-len(S), 0, +1) is the same as range(-5, 0, 1). The 1 is simply our step parameter which signifies that the difference between each number in our range is 1. So, our range starts at -5 and goes up to (but does not include) 0, meaning our range is [-5, -4, -3, -2, -1] – the negative indices of "howdy". These indices are characters h, o, w, d, y.

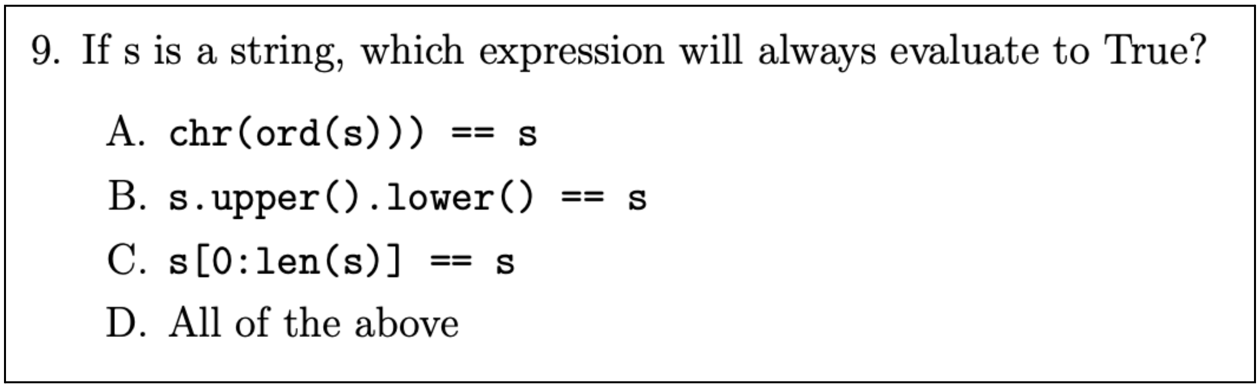
(D) range() has a default value of 1 for the step parameter. So, (D) 'for i in range(-len(S), 0)' is the same as 'for i in range(-len(S), 0, 1)'.

(E) 'for i in range(-l, -len(S) - 1, -1)' will traverse the string from right to left using negative indices. Using "howdy" as an example again…

-len(S) is -5, so our range is range(-1, -6, -1). The second -1 here is our step parameter, which indicates that the numbers in our range are descending (decrement 1 between each). So, we our range starts at -1, and goes down to (but does not include) -6, meaning our range is [-1, -2, -3, -4, -5] – negative indices of "howdy". These indices are characters y, d, w, o h.

Wrong answer explanations:
(A) this will not iterate over the string using indices. Because our loop is 'for char in S', we iterate directly over S, so each iteration of the loop char will refer to a character in S.
(B) range(len(S)) will only contain non-negative numbers. For example, when s = "dollyParton",

len(S) is 11. range() has a default starting parameter of 0, so range(len(S)) is equivalent to range(0, 11) which is numbers [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10].



9. If s is a string, which expression will always evaluate to True?

   A. `chr(ord(s)) == s`
   B. `s.upper().lower() == s`
   C. `s[0:len(s)] == s`
   D. All of the above

Correct answer explanation:
(C) This will always be true. We simply obtain all characters in s, starting from index 0 to len(s) - 1 (recall that slicing is exclusive of the last character, similar to range()). This is the whole length of s, so s[0:len(s)] simply recreates string s.

Wrong answer explanations:
(A) ord() only takes a single character (or else it causes an error). The question only specified that s was a string, not that s was a string consisting of a single character. So if s has multiple characters, (A) will be erroneous.
(B) s.upper().lower() converts the whole string to lowercase, but it's possible that s was entirely uppercase before. For example, s = "SILLYGOOSE". s.upper().lower() will be "sillygoose".
(D) Cannot be correct because A and B are both incorrect

TRACING

10. (3 points)

```
s = "vpduwFrrnlh"
new = ""
for char in s:
   new += chr(ord(char) - 3)
print(new)
```

smartCookie

This is a fun one!

OK – we make a new string, initially empty. And we iterate through the characters in s and add onto our new string. We convert each character in s to its ASCII value, subtracting three, and then adding the resulting character to our new string. This just means that each character in the original string is shifted three positions backward in the alphabet. So, three characters before "v" is "s", three characters before "p" is "m", etc. The following table shows all the conversions.

Original Character	Character 3 before
v	s
p	m
d	a
u	r
w	t
F	C
r	o
r	o
n	k
l	i
h	e

After appending all the converted characters to our string new, we get "smartCookie". Yes, YOU are a smart cookie for solving this question! =)

11. (3 points)

```
s = "SillyGoose-781"
result = ""

for char in s:
    if char in str(not s):
        result += "B"
    elif char.isalpha():
        result += char.upper()
    elif char.isdigit():
        result += str(int(char) + 5)
    else:
        result += char

print(result)
```

SIBBYGOOBB-12136

This is a fun question, too!

OK – we iterate over string s, but depending on what the current character is, we'll add on something different to our result string (initially empty).

The 'if' condition here is 'if char in str(not s)'. Because s is a string with characters in it, not s will be False. So, str(not s) will be "False". So this if condition really just evaluates to "if char in 'False' ". If the current character we're at is "F", "a", "l", "s", or "e", then instead of adding that character, we will instead add "B" to the result.

elif char.isalpha() – this is just saying, else if the character is any other alphabetical letter, then we'll add the uppercase version of it to the string.

elif char.isdigit() – this says, else if the character is a number, we'll convert it to an int, so that we can add 5 to it, and then we'll convert the outcome back to a string and add that to our result.

else: we just add the character with no modification made

The following table shows the adjustments that are made for each character in s.

Character	Branch Triggered	Adding to Result	Result string
S	elif char.isalpha()	S	S
i	elif char.isalpha()	I	SI
l	if char in str(not s)	B	SIB
l	if char in str(not s)	B	SIBB
y	elif char.isalpha()	Y	SIBBY
G	elif char.isalpha()	G	SIBBYG
o	elif char.isalpha()	O	SIBBYGO
o	elif char.isalpha()	O	SIBBYGOO
s	if char in str(not s)	B	SIBBYGOOB
e	if char in str(not s)	B	SIBBYGOOBB
-	else	-	SIBBYGOOBB-
7	elif char.isdigit()	12	SIBBYGOOBB-12
8	elif char.isdigit()	13	SIBBYGOOBB-1213
1	elif char.isdigit()	6	SIBBYGOOBB-12136

Thus, our final result is SIBBYGOOBB-12136.

12. (3 points)

```
s = "oompaLoompa"
result = (s[-len(s):-len(s)+5] + " " + s[-6:])

print(result)
```

oompa Loompa

Practice with negative indices!!! Fun!!!

For reference, here are the letter:index pairs for s (which has length 11).
s (oompaLoompa): o:0, o:1, m:2, p:3, a:4, L:5, o:6, o:7, m:8, p:9, a:10

However, we're using negative indexes here. Oops. So, negative indices for oompaLoompa would be o:-11, o:-10, m:-9, p:-8, a:-7, L:-6, o:-5, o:-4, m:-3, p:-2, a:-1

OK. So when we slice to obtain s[-len(s):-len(s) + 5], this is equivalent to s[-11:-6]. So we get all character starting at index -11, and up to (but not including) index -6. This is "oompa".
We then add on a space. And then we slice to get s[-6:]. This gives us the characters starting at index -6, and going to the end of the string – so, "Loompa".

Thus, our final result is "oompa Loompa".

```
spongebob = "goofyG00ber"
patrick = "krus7ykr4b"
gary = "+25"

sandy = patrick.islower() and patrick[999:9999] == ""
squidward = gary.isalnum() and gary.isdigit()
mrKrabs = spongebob[6:8].isdigit() and spongebob.count("o") == 2

print(sandy, squidward, mrKrabs)
```

True False True

The movie was SO iconic!

OK, so we have a handful of strings, and we do two checks on each string and store the results. Let's walk through them:

sandy = patrick.islower() and patrick[999:9999] == ""

.islower() will check whether all of the alphabetic characters in the string are lowercase. In other words, numbers will not cause this to be False. Because patrick is "krus7ykr4b", all of the alphabetic characters are lowercase. Also, remember that when we slice out of bounds (using indices longer than the string), we don't get an error. We simply get the empty string. Thus, sandy will be True and True, which evaluates to True.

squidward = gary.isalnum() and gary.isdigit()

.isalnum() checks whether the string only consists of alphabetic and or numeric characters. But because gary is “+25”, gary.isalnum() will be False (due to the + not being alphabetic or numeric). Now we know squidward is

False and gary.isdigit()

Which evaluates to False immediately (short-circuit evaluation, huzzah!!). But for what it’s worth, gary.isdigit() is also False, also because of the + not being a numeric character. So, squidward is False and False, which evaluates to False.

mrKrabs = spongebob[6:8].isdigit() and spongebob.count(“o”) == 2

spongebob is “goofyg00ber”. So, spongebob[6:8] is “00”. (indexing [6:8] will get us indices 6 and 7, which are the 7th and 8th characters in the string. Remember 0-based indexing!) Thus, spongebob[6:8].isdigit() will be True. And “goofyg00ber” has exactly 2 “o” in it, so spongebob.count(“o”) will be True. Thus, mrKrabs is True and True, which evaluates to True.

14. (3 points)

```
def catInTheHat(thing1, thing2):
    thing1 = thing1 + thing2
    thing2 = thing1
    print(thing1 is thing2)

string1 = "greenEggs"
string2 = "Ham"
catInTheHat(string1, string2)
print(string1, string2)
```

True greenEggs Ham

This question is intended to show folks that changes made to strings passed to functions do not persist. We pass thing1 and thing2 (“greenEggs” and “Ham”) to catInTheHat. catInTheHat adds overwrites thing1 by adding it and thing2 together. It then sets thing2 to this updated thing1. At this point within the function, thing1 and thing2 are both “greenEggsHam”, hence the print displaying True.

However, these changes to thing1 and thing2 do not “stick” after the function, because strings are immutable. When we modified those strings, we were only updating local variables. When we print string1 and string2 after the function, they will still be “greenEggs” and “Ham”.

15. (3 points)

```
myGuineaPig = "Jerry"  # =)
print(myGuineaPig[0:1000] * 2)
```

JerryJerry

This question is intended to show you kiddos that slicing with strings will not cause index out of bounds errors. Although myGuineaPig only has up to index 4, we specify an ending index of 1000 (well, 999, really, because slicing is exclusive of the end, but… semantics). But this doesn’t crash our program – we simply get the entirety of the string, and no more. So we get back “Jerry”, multiply by 2, and our result is “JerryJerry”.

16. (3 points)

```
alpha, beta = "castiel", "dean"
omega = beta[:2] + alpha[2:]
sigma = beta[:len(beta)] + alpha[:3]
print(min(omega, sigma), max(omega, sigma))
```

deancas destiel

My FAVORITE question out of any of these worksheets!! If you know, you know ♥

We assign the string "castiel" to the variable alpha and "dean" to the variable beta. To form omega and sigma, we use string slicing. The letter:index pairs for "castiel" (alpha) and "dean" (beta) are:

alpha (castiel): c:0, a:1, s:2, t:3, i:4, e:5, l:6
beta (dean): d:0, e:1, a:2, n:3

omega is created by combining the characters in beta up to the second index ("de") with the characters in alpha starting at the second index ("stiel"). This results in the string "destiel".

sigma is formed by taking the characters in beta up to the length of beta. Since len(beta) is 4, beta[:4] gives us the entire string "dean". We append this to the characters in alpha up to the third index ("cas"). This results in the string "deancas".

We then find the minimum and maximum between omega and sigma. To do this, we compare their lexicographic ordering (think which one would come first in a dictionary). We compare the first characters in omega (“destiel”) and sigma (“deancas”) – both “d”. We then compare their second characters – both “e”. We then compare their third characters. omega has “s” and sigma has “a”. Thus, sigma has a lower lexicographic ordering than omega, so min(omega, sigma) will be sigma and max(omega, sigma) will be omega. This means we will print deancas destiel.