py101-py109_sp_easy_3

December 4, 2024

1 PY 101 - PY 109

1.1 Small Problems

1.1.1 Easy 3

1.2 Repeat Yourself

Write a function that takes two arguments, a string and a positive integer, then prints the string as many times as the integer indicates.

```
repeat('Hello', 3)
```

```
Hello
Hello
Hello
```

[11]:
```
def repeater(str_to_repeat, num_times):
    for i in range(num_times):
        print(str_to_repeat)

repeater("Hello", 3)
```

```
Hello
Hello
Hello
```

1.3 ddaaiillyy ddoouubbllee

Write a function that takes a string argument and returns a new string that contains the value of the original string with all consecutive duplicate characters collapsed into a single character.

[32]:
```
def crunch(str_to_crunch):
    string_builder = ""
    for char in str_to_crunch:
        if not string_builder:
            string_builder = string_builder + char
        if char != string_builder[-1]:
            string_builder = string_builder + char
    return string_builder
```

```
print(crunch('ddaaiillyy ddoouubbllee') == 'daily double')
print(crunch('4444abcabccba') == '4abcabcba')
print(crunch('gggggggggggggggg') == 'g')
print(crunch('abc') == 'abc')
print(crunch('a') == 'a')
print(crunch('') == '')
```

```
True
True
True
True
True
True
```

1.4 Bannerizer

Write a function that takes a short line of text and prints it within a box.

```
print_in_box('To boldly go where no one has gone before.')
```

```
+--------------------------------------------+
|                                            |
| To boldly go where no one has gone before. |
|                                            |
+--------------------------------------------+
```

```
print_in_box('')
```

```
+--+
|  |
|  |
|  |
+--+
```

[45]:
```
# I need the length of the text + 2 characters for the dashes above and below

string = "To boldly go where no one has gone before."

print('+' + ('-' * 44) + '+')
print('|' + (" " * 44) + '+')
print('| ' + string + ' |')
print('|' + (" " * 44) + '+')
print('+' + ('-' * 44) + '+')

# Now that that works, let's work on making it a little more abstract
def print_in_box(str_to_print):
    border = f'\n+{'-'*(len(str_to_print)+2)}+'
    padding = f'\n|{' '*(len(str_to_print)+2)}|'
    new_str = f'\n| {str_to_print} |'
```

```
    print(f'{border} {padding} {new_str} {padding} {border}')

print_in_box("To boldly go where no one has gone before.")
print_in_box("")
```

```
+-------------------------------------------+
|                                           +
| To boldly go where no one has gone before. |
|                                           +
+-------------------------------------------+

+-------------------------------------------+
|                                           |
| To boldly go where no one has gone before. |
|                                           |
+-------------------------------------------+

+--+
|  |
|  |
|  |
+--+
```

1.4.1 Further Exploration

Modify this function so that it truncates the message if it doesn't fit inside a maximum width provided as a second argument (the width is the width of the box itself). You may assume no maximum if the second argument is omitted.

[48]:
```
def print_in_box(str_to_print, max_length):
    if (len(str_to_print) -2) > max_length:
        str_to_print = str_to_print[:(max_length-2)]
    border = f'\n+{'-'*(len(str_to_print)+2)}+'
    padding = f'\n|{' '*(len(str_to_print)+2)}|'
    new_str = f'\n| {str_to_print} |'

    print(f'{border} {padding} {new_str} {padding} {border}')

print_in_box("To boldly go where no man - no one - has gone before.", 44)
print_in_box("To boldly go where no one has gone before.", 44)
```

```
+-------------------------------------------+
|                                           |
| To boldly go where no man - no one - has g |
|                                           |
+-------------------------------------------+

+-------------------------------------------+
```

```
|                                            |
| To boldly go where no one has gone before. |
|                                            |
+--------------------------------------------+
```

1.5 Further Exploration 2

For a challenging but fun exercise, try word wrapping messages that are too long to fit, so that they appear on multiple lines but are still contained within the box. This isn't an easy problem, but it's doable with basic Python.

[64]:
```python
def print_in_box(banner_str, line_length):
    str_length = len(banner_str)
    max_str_length = line_length - 2
    banner_builder = {}
    border = f'+{'-'*(line_length)}+'
    padding = f'|{' '*(line_length)}|'
    # new_str = f'\n| {str_to_print} |'

    if str_length > max_str_length:
        num_lines = (str_length // max_str_length) + 5
    else:
        num_lines = 5

    for i in range(num_lines):
        if i == 0 or i == num_lines - 1:
            print(border)
        elif i == 1 or i == num_lines - 2:
            print(padding)
        else:
            if len(banner_str) <= max_str_length:
                blanks = max_str_length -len(banner_str)
                print(f'| {banner_str + (" " * blanks)} |')
            else:
                str_to_print = banner_str[0:max_str_length]
                banner_str = banner_str[max_str_length:]
                print(f'| {str_to_print} |')


print_in_box("To boldly go where no one has gone before.", 38)
```

```
+--------------------------------------+
|                                      |
| To boldly go where no one has gone b |
| efore.                               |
|                                      |
+--------------------------------------+
```

1.6 Strings Strings

Write a function that takes one argument, a positive integer, and returns a string of alternating '1's and '0's, always starting with a '1'. The length of the string should match the given integer.

[68]:
```
def stringy(num):
    string_builder = "1"
    for i in range(2,num+1):
        addme = '1' if i % 2 else '0'
        string_builder = string_builder + addme

    return string_builder



print(stringy(6) == "101010")           # True
print(stringy(9) == "101010101")        # True
print(stringy(4) == "1010")             # True
print(stringy(7) == "1010101")          # True
```

```
True
True
True
True
```

1.7 Right Triangles

Write a function that takes a positive integer, n, as an argument and prints a right triangle whose sides each have n stars. The hypotenuse of the triangle (the diagonal side in the images below) should have one end at the lower-left of the triangle, and the other end at the upper-right.

[74]:
```
def triangle(num):
    for i in range(1, num + 1):
        num_spaces = num - i
        print(f'{" " * (num - i)}{i * "*"}')

triangle(5)
triangle(9)
```

```
    *
   **
  ***
 ****
*****
        *
       **
      ***
     ****
    *****
```

```
   ******
  *******
 ********
*********
```

1.8 Madlibs

Madlibs is a simple game where you create a story template with "blanks" for words. You, or another player, then construct a list of words and place them into the story, creating an often silly or funny story as a result.

Create a simple madlib program that prompts for a noun, a verb, an adverb, and an adjective, and injects them into a story that you create.

[5]:
```
def get_game_data(string):
    while True:
        if string[0] == "a":
            string = input(f"Enter an {string}: ")
        else:
            string = input(f"Enter a {string}: ")
        if string != "" and not string.isdigit():
            return string

def game():
    noun = get_game_data("noun")
    verb = get_game_data("verb")
    adjective = get_game_data("adjective")
    adverb = get_game_data("adverb")

    print(f"Do you {verb} your {adjective} {noun} {adverb}?")
    print(f"The {adjective} {noun} {verb} {adverb} over the lazy {noun}.")

game()
```

```
Enter a noun:  mom
Enter a verb:  flies
Enter an adjective:  sweetly
Enter an adverb:  short

Do you flies your sweetly mom short?
The sweetly mom flies short over the lazy mom.
```

I really chose the least efficient way possible to do this.

1.9 Double Doubles

A double number is an even-length number whose left-side digits are exactly the same as its right-side digits. For example, 44, 3333, 103103, and 7676 are all double numbers, whereas 444, 334433, and 107 are not.

Write a function that returns the number provided as an argument multiplied by two, unless the argument is a double number. If the argument is a double number, return the double number as-is.

[22]:
```
def twice(num):
    temp_str = str(num)
    len_str = len(temp_str)
    part_a = temp_str[0:(len_str//2)]
    part_b = temp_str[(len_str//2):]

    if part_a == part_b:
        return num
    else:
        return num * 2



print(twice(37) == 74)                  # True
print(twice(44) == 44)                  # True
print(twice(334433) == 668866)          # True
print(twice(444) == 888)                # True
print(twice(107) == 214)                # True
print(twice(103103) == 103103)          # True
print(twice(3333) == 3333)              # True
print(twice(7676) == 7676)              # True
```

```
True
True
True
True
True
True
True
True
```

1.10 Grade Book

Write a function that determines the mean (average) of the three scores passed to it, and returns the letter associated with that grade.

Numerical score letter grade list:

90 <= score <= 100: 'A' 80 <= score < 90: 'B' 70 <= score < 80: 'C' 60 <= score < 70: 'D' 0 <= score < 60: 'F' Tested values are all between 0 and 100. There is no need to check for negative values or values greater than 100.

[29]:
```
def get_grade(*args):
    counter = 0
    running_total = 0
    for arg in args:
```

```
        if type(arg) != int:
            raise TypeError('Integers only.')
        counter += 1
        running_total += arg
    mean = running_total // counter

    if 90 <= mean <= 100:
        return 'A'
    elif 80 <= mean < 90:
        return 'B'
    elif 70 <= mean < 80:
        return 'C'
    elif 60 <= mean < 70:
        return 'D'
    else:
        return 'F'

print(get_grade(95, 10, 20, 80, 75, 35))
print(get_grade(50, 50, 95) == "D")      # True
```

```
F
True
```

1.11 Clean up the words

Given a string that consists of some words and an assortment of non-alphabetic characters, write a function that returns that string with all of the non-alphabetic characters replaced by spaces. If one or more non-alphabetic characters occur in a row, you should only have one space in the result (i.e., the result string should never have consecutive spaces).

[36]:
```
def clean_up(string):
    string_builder = ""
    last_char = ""
    for char in string:
        if char.isalpha():
            string_builder += char
            last_char = char
            continue
        if last_char == " ":
            continue
        last_char = " "
        string_builder += " "
    return string_builder

print(clean_up("---what's my +*& line?") == " what s my line ")
print(clean_up("K     ") == "K     ")   # True
```

```
True
True
```

1.12 What Century is That?

Write a function that takes a year as input and returns the century. The return value should be a string that begins with the century number, and ends with 'st', 'nd', 'rd', or 'th' as appropriate for that number.

New centuries begin in years that end with 01. So, the years 1901 - 2000 comprise the 20th century.

[67]:
```
def get_digits(num):
    num_to_str = str(num)
    len_of_num = len(str(num))
    if num <= 1000:
        if num <= 100:
            return "1"
        if num_to_str[-1] == "0":
            return num_to_str[0]
        else:
            return str(int(num_to_str[0]) + 1)
    digits_to_take = len_of_num - 2
    if num_to_str[-1] == "0":
        return num_to_str[:digits_to_take]
    else:
        return str(int(num_to_str[:digits_to_take]) + 1)

def century(year):
    SUFFIX = {
        "1": 'st',
        "2": 'nd',
        "3": 'rd',
        "4": 'th',
        "5": 'th',
        "6": 'th',
        "7": 'th',
        "8": 'th',
        "9": 'th',
        "0": 'th'
    }

    prefix_digits = get_digits(year)
    last_digit = prefix_digits[-1]
    if 11 <= int(prefix_digits[-2:]) <= 19:
        return prefix_digits + "th"
    else:
        return prefix_digits + SUFFIX[last_digit]

print(century(2000) == "20th")          # True
print(century(2001) == "21st")          # True
print(century(1965) == "20th")          # True
```

```
print(century(256) == "3rd")            # True
print(century(5) == "1st")              # True
print(century(10103) == "102nd")        # True
print(century(1052) == "11th")          # True
print(century(1127) == "12th")          # True
print(century(11201) == "113th")        # True
```

```
True
True
True
True
True
True
True
True
True
```