Qualified by andela Assessment Result

Tanyaradzwa Chisepo

Code2040: Advanced Algorithms [2023] Results

✉ tanyachisepo04@gmail.com

Assessment Summary

100%

1 h, 26 m, 13 s
Active Time

- Tanyaradzwa Chisepo **opened** this assessment on Sunday, July 3, 2022 12:06 AM
- Tanyaradzwa Chisepo **started** this assessment on Sunday, July 3, 2022 12:06 AM
- Tanyaradzwa Chisepo **submitted** this assessment after **5 days, 21 hours and 55 minutes** on Friday, July 8, 2022 10:02 PM
- This student spent **1 hour and 26 minutes** active in the browser working on the assessment

Solutions Summary

Challenge	Score	Active Time
#1: Split Integer	100%	46 m, 33 s
#2: Valid Braces	100%	18 m, 23 s
#3: Base64 Encoding	100%	21 m, 17 s

Qualified by andela Assessment Code Solution

#1: Split Integer

Scoring

100%

4 / 4 Tests (5 Attempts)

Timing

46 m, 33 s Active Time

765 ms Run Time

Candidate Notes:

integer division
modulus function, get remainder

split(10, 2) = [5, 5]

div = 10//#parts

result = div x #parts

if number%parts ==0:

```
    for i in range(#parts):
                array.append(div)
```

else:

Instructions

Task

In this challenge, you will write a function to divide an integer into a number of even parts, which will be returned in a result array. Summing the integers in this result array will produce the original number.

For example, given `number = 10` and `parts = 2`, `splitInteger(10, 2)` should return an array of integers with length equal to `parts`: `[5, 5]`.

Not all numbers will offer a clean division. In this case, we should split the number as closely as possible to even, with the smaller numbers in the front of the array. For example,

```
splitInteger(11, 3) → [3, 4, 4]
```

There is no reason to test for edge cases; the input to your function will always be valid for this challenge. Please see the below specification for the exact constraints on the input:

```
{
  "method": "split_integer",
  "desc": "Divides an integer into an even number of parts.",
  "args": {
    "num": {
      "type": "Integer",
      "desc": "The integer number that should be split into equal parts"
    },
    "parts": {
      "type": "Integer",
      "desc": "The number of parts that the integer should be split into"
    }
  },
  "returns": {
```

```
    "type": "List<Integer>",
    "desc": "An array of part values which sum to `num`. The parts will be ordered from
smallest to largest."
  },
  "constraints": [
    "- `0 < num ≤ 100`",
    "- `0 < parts ≤ num`"
  ],
  "examples": [
    {
      "name": "Completely even parts",
      "args": [10, 5], "returns": [2,2,2,2,2]
    },
    {
      "name": "Divided as closely as possible into even parts with the smallest parts grouped
at the front of the result",
      "args": [20, 6],
      "returns": [3,3,3,3,4,4]
    }
  ]
}
```

Solution Code

Python

```python


def split_integer(num, parts):
    x = num
    y = parts



#108 21 21 22 22 22

    final_array = []

    modulo = x%y
    quotient = x//y

    #for case with no clean division
    if modulo != 0:

        #evenly spread the remainder to each part

        for i in range(y):
            if i>= (y-modulo):
              final_array.append(quotient+1)

            else:

```

```
            final_array.append(quotient)

  #for clean division
    else:
        for i in range(y) :
          final_array.append(quotient)


    return final_array
```

Candidate's Tests

```
import unittest
from solution import split_integer

class Test(unittest.TestCase):
    def test_should_handle_evenly_distributed_cases(self):
        self.assertEqual(split_integer(10, 1), [10])
        self.assertEqual(split_integer(2, 2), [1,1])
        self.assertEqual(split_integer(20, 5), [4, 4, 4, 4, 4])
```

#2: Valid Braces

Scoring

100%

6 / 6 Tests (1 Attempt)

Timing

18 m, 23 s Active Time

724 ms Run Time

Candidate Notes:

create regex, define valid brace patterrn
correct order
return T or F
'(){}[]' and '([{}])' is valid,, with matched braces

create two sets, opening and closing braces

create a dictionary, for matched braces

stack???

go through input, if current brace is opening, add it to the stack,
check to see if it is a closing brace and not in stack, immediately return false

Instructions

Write a function called `validBraces` that takes a string of braces, and determines if the order of the braces is valid. `validBraces` should return true if the string is valid, and false if it's invalid.

All input strings will be nonempty, and will only consist of open parentheses `'('` , closed parentheses `')'`, open brackets `'['`, closed brackets `']'`, open curly braces `'{'` and closed curly braces `'}'`.

What is considered Valid?

A string of braces is considered valid if all braces are matched with the correct brace. For example:

`'(){}[]'` and `'([{}])'` would be considered valid, while `'(}'`, `'[(])'`, and `'[({})](]'` would be considered invalid.

Specification

```
{
"method": "valid_braces",
"desc": "Checks if the brace order is valid",
"args": {
```

```
  "braces": {"type": "String", "desc": "A string representation of an order of braces"}
},
"returns": {"type": "Boolean", "desc": "Returns `true` if order of braces are valid"}
}
```

Examples:

Input	Output
`validBraces( "(){}[]" )`	`true`
`validBraces( "(}" )`	`false`
`validBraces( "[(])" )`	`false`
`validBraces( "([{}])" )`	`true`

Solution Code

Python

```python
def valid_braces(braces: str) -> bool:

    opening = set('([{')
    closing = set(')]}')
    match = {')' : '(' , ']' : '[' , '}' : '{'}
    stack  = []

    for i in braces:
      if i in opening :
           stack.append(i)
      if i in closing :
          if not stack :
               return False
          elif stack.pop() != match[i] :
               return False
          else :
               continue
    if not stack :
       return True
    else :
           return False


```

Candidate's Tests

```python
import unittest
from solution import valid_braces

class TestSampleTests(unittest.TestCase):
    def test_distinguish_valid_braces(self):
        """ distinguish valid braces """
        self.assertEqual(valid_braces("([{}])"), True)
        self.assertEqual(valid_braces("[(])"), False)

```

#3: Base64 Encoding

Scoring

100%

3 / 3 Tests (2 Attempts)

Timing

21 m, 17 s Active Time

534 ms Run Time

Instructions

Create a function or a method that converts the value of the String **to Base64** using the ASCII (UTF-8 for C#) character set.

Do not use built in functions.

As an example the input string `this is a string!!` should produce the output string `dGhpcyBpcyBhIHN0cmluZyEh`.

You can learn more about Base64 encoding and decoding here (http://en.wikipedia.org/wiki/Base64) .

Specification

```
{
"method": "to_base64",
"desc": "",
"args": {
  "str": {"type": "String", "desc": "String to be converted using Base64 format"}
},
"returns": {"type": "String", "desc": "The converted string"},
"examples": [
  {"args": ["this is a string!!"], "returns": "dGhpcyBpcyBhIHN0cmluZyEh"},
  {"args": ["ABCDEFGHIJKLMNOPQRSTUVWXYZ "], "returns": "QUJDREVGR0hJSktMTU5PUFFSU1RVVldYWVog"}
]
}
```

Solution Code

Python

```python

#func to convert each char in string to 8-bit binary
def char_to_binary(string):

    bin_str = ''
    binary = []

    for char in string:
        #get ascii value of each char
```

```
        ascii_char = ord(char)
        #remove '0b' prefix
        binary_char = bin(ascii_char).replace('0b', '')

        #make 8-bit binary
        asc = binary_char[::-1]
        while len(asc) < 8:
            asc += '0'
            binary_char = asc[::-1]

        binary.append(binary_char)
    #combine all 8-bit binaries into sinlge string
    for bits in binary:
        bin_str += bits
    return bin_str



def to_base64(string):
  temp = ''
  count = 0
  encoded = ''
  result = []
  six_bit = []


  #dictionary to store base64 table
  b64_dict = { 0:   'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H' , 8: 'I', 9:
'J', 10:    'K', 11:    'L', 12:    'M', 13:    'N', 14:    'O', 15:    'P', 16:    'Q', 17:
'R' , 18:   'S', 19:    'T', 20:    'U', 21:    'V', 22:    'W', 23:    'X', 24:    'Y', 25:
'Z', 26:    'a', 27: 'b', 28:   'c', 29:    'd', 30:    'e', 31:    'f', 32:    'g', 33:
'h',34: 'i', 35:    'j', 36:    'k', 37:    'l', 38:    'm', 39:    'n', 40:    'o', 41:
'p', 42:    'q', 43: 'r', 44:   's', 45:    't', 46:    'u', 47:    'v', 48:    'w', 49:
'x', 50:    'y', 51:    'z', 52:    '0', 53:    '1', 54:    '2', 55:    '3', 56:    '4', 57:
'5', 58:    '6', 59:    '7', 60:    '8', 61:    '9', 62: '+' , 63:  '/' }


  bin_str = char_to_binary(string)

  #bunch binary digits into groups of 6
  for bit in bin_str:
      temp += bit
      count += 1
      if count== 6:
          six_bit.append(temp)
          temp = ''
          count = 0

  #6-bit binary back to base 10 integer
  for group in six_bit:
      decimal =   int(group, 2)
      result.append(decimal)
```

```
55
56    for decimal in result:
57        #use dictionary to convert integer to base64 char
58        letter  = b64_dict[decimal]
59        encoded += letter
60
61    return encoded
62
```

Candidate's Tests

```
1  import unittest
2  # Note: the class must be called Test
3  class Test(unittest.TestCase):
4    def test_should_handle_sentences(self):
5      self.assertEqual(to_base64("this is a string!!"), "dGhpcyBpcyBhIHN0cmluZyEh")
6      self.assertEqual(to_base64("this is a test!"), "dGhpcyBpcyBhIHRlc3Qh")
```