Forward School

Program Code: J620-002-4:2020

Program Name: FRONT-END SOFTWARE DEVELOPMENT

Title : Exercise 1

Name: Phua Yan Han

IC Number: 050824070059

Date : 24/6/23

Introduction : python basics

Conclusion : learned basic syntax

EXERCISE 1

RUN ME

Please run the code snippet below. It is required for running tests for your solution.

In [2]:
```
def test(got, expected):
    if got == expected:
        prefix = ' OK '
    else:
        prefix = '  FAIL '
    print(('%s got: %s expected: %s' % (prefix, repr(got), repr(expected))))
```

Question 1

In [3]:
```
# A. donuts
# Given an int count of a number of donuts, return a string
# of the form 'Number of donuts: <count>', where <count> is the number
# passed in. However, if the count is 10 or more, then use the word 'many'
# instead of the actual count.
# So donuts(5) returns 'Number of donuts: 5'
# and donuts(23) returns 'Number of donuts: many'
def donuts(count):
    #++ your code here ++

    if count >= 10:
        num = "many"
    else:
        num = str(count)

    return "Number of donuts: " + num


print('donuts')
# Each line calls donuts, compares its result to the expected for that call.
test(donuts(4), 'Number of donuts: 4')
test(donuts(9), 'Number of donuts: 9')
test(donuts(10), 'Number of donuts: many')
test(donuts(99), 'Number of donuts: many')
```

```
donuts
 OK  got: 'Number of donuts: 4' expected: 'Number of donuts: 4'
 OK  got: 'Number of donuts: 9' expected: 'Number of donuts: 9'
 OK  got: 'Number of donuts: many' expected: 'Number of donuts: many'
 OK  got: 'Number of donuts: many' expected: 'Number of donuts: many'
```

Question 2

In [9]:
```
# B. both_ends
# Given a string s, return a string made of the first 2
# and the last 2 chars of the original string,
# so 'spring' yields 'spng'. However, if the string length
# is less than 2, return instead the empty string.
def both_ends(s):
    #++ your code here ++
    if len(s)<=2:
        return ""
    return f"{s[0:2]}{s[len(s)-2]}{s[-1]}"




print()
print('both_ends')
test(both_ends('spring'), 'spng')
test(both_ends('Hello'), 'Helo')
test(both_ends('a'), '')
test(both_ends('xyz'), 'xyyz')
```

```
both_ends
 OK  got: 'spng' expected: 'spng'
 OK  got: 'Helo' expected: 'Helo'
 OK  got: '' expected: ''
 OK  got: 'xyyz' expected: 'xyyz'
```

Question 3

In [21]:
```
# C. fix_start
# Given a string s, return a string
# where all  occurences of its first char have
# been changed to '*', except do not change
# the first char itself.
# e.g. 'babble' yields 'ba**le'
# Assume that the string is length 1 or more.
# Hint: s.replace(stra, strb) returns a version of string s
# where all instances of stra have been replaced by strb.
def fix_start(s):
    #++ your code here ++
    arr =[]
    char=""
    for i,x in enumerate(s):
        if i == 0:
            char = x
            arr.append(x)
        elif x == char:
            arr.append("*")
        else:
            arr.append(x)

    return "".join(arr)



print()
print('fix_start')
test(fix_start('babble'), 'ba**le')
test(fix_start('aardvark'), 'a*rdv*rk')
test(fix_start('google'), 'goo*le')
test(fix_start('donut'), 'donut')
```

```
fix_start
 OK  got: 'ba**le' expected: 'ba**le'
 OK  got: 'a*rdv*rk' expected: 'a*rdv*rk'
 OK  got: 'goo*le' expected: 'goo*le'
 OK  got: 'donut' expected: 'donut'
```

Question 4

In [38]:
```
# D. MixUp
# Given strings a and b, return a single string with a and b separated
# by a space '<a> <b>', except swap the first 2 chars of each string.
# e.g.
#   'mix', pod' -> 'pox mid'
#   'dog', 'dinner' -> 'dig donner'
# Assume a and b are length 2 or more.
def mix_up(a, b):
   #++ your code here ++
    sliceA = b[0:2]
    sliceB = a[0:2]
    resA = []
    resB = []
    for i,x in enumerate(a):
        if i==1:
            resA.append(sliceA)
        elif i>1:
            resA.append(x)
    for z,y in enumerate(b):
        if z==1:
            resB.append(sliceB)
        elif z>1:
            resB.append(y)

    return f"{''.join(resA)} {''.join(resB)}"

print()
print('mix_up')
test(mix_up('mix', 'pod'), 'pox mid')
test(mix_up('dog', 'dinner'), 'dig donner')
test(mix_up('gnash', 'sport'), 'spash gnort')
test(mix_up('pezzy', 'firm'), 'fizzy perm')
```

```
mix_up
 OK  got: 'pox mid' expected: 'pox mid'
 OK  got: 'dig donner' expected: 'dig donner'
 OK  got: 'spash gnort' expected: 'spash gnort'
 OK  got: 'fizzy perm' expected: 'fizzy perm'
```