**第15.2节问题：计算因子**

15.1以下哪项陈述是正确的？

 A.每个递归函数都必须具有基本情况或停止条件。

 B.每次递归调用都会减少原始问题，使其越来越接近基本情况，直到成为这种情况。

 C.如果递归不能以允许它最终收敛到基本情况的方式减少问题，则可能发生无限递归。

 D.每个递归函数都必须有一个返回值。

 E.递归函数的调用与非递归函数不同。

ABC

答案分析：ABC

15.2填写代码以完成以下计算阶乘的功能。

def factorial(n):

if n == 0: # Base case

return 1

else:

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ # Recursive call

A. n \* (n - 1)

B. n

C. n \* factorial(n - 1)

D. factorial(n - 1) \* n

CD

答案分析：CD

15.3以下递归函数的基本情况是什么？

def xfunction(n):

if n > 0:

print(n % 10)

xfunction(n // 10)

 A. n> 0

 B. n <= 0

 C.没有基本情况

 D. n <0

B

答案分析：B

15.4分析以下递归函数。

def factorial(n):

return n \* factorial(n - 1)

 A.调用factorial（0）返回0。

 B.调用factorial（1）返回1。

 C.调用阶乘（2）返回2。

 D.调用阶乘（3）返回6。

 E.该函数无限运行并导致StackOverflowError。

Ë

答案分析：E

15.5为factorial（5）调用的清单15.1中的阶乘函数有多少次？

 A. 3

 B. 4

 C. 5

 D. 6

D

答案分析：D

**第15.3节问题：计算Fibonacci数**

15.6以下哪项陈述属实？

A.Fibonacci系列以0和1开头，每个后续数字是系列中前两个数字的总和。

 B.斐波那契数列以1和1开头，每个后续数字是系列中前两个数字的总和。

 C.斐波纳契数列从1和2开始，每个后续数字是系列中前两个数字的总和。

 D.斐波那契数列以2和3开头，每个后续数字是系列中前两个数字的总和。

A

答案分析：A

15.7为fib（5）调用清单15.2中的fib函数多少次？

 A. 14

 B. 15

 C. 25

 D. 31

 E. 32

B

答案分析：B

15.8填写代码以完成以下用于计算Fibonacci数的函数。

def fib(index):

if index == 0: # Base case

return 0

elif index == 1: # Base case

return 1

else: # Reduction and recursive calls

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

A. fib(index - 1)

B. fib(index - 2)

C. fib(index - 1) + fib(index - 2)

D. fib(index - 2) + fib(index - 1)

CD

答案分析：CD

**第15.4节使用递归解决问题**

15.9在以下功能中，基本情况是什么？

def xfunction(n):

if n == 1:

return 1

else

return n + xfunction(n - 1)

 A. n是1。

 B. n大于1。

 C. n小于1。

 D.没有基本情况。

A

答案分析：A

15.10调用以下函数后xfunction（4）的返回值是多少？

def xfunction(n):

if n == 1:

return 1;

else:

return n + xfunction(n - 1)

A. 12

B. 11

C. 10

D. 9

C

答案分析：C

15.11填写代码以完成以下功能，以检查字符串是否为回文序列。

def isPalindrome(s):

if len(s) <= 1: # Base case

return True

elif \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

return False

else:

return isPalindrome(s.substring(1, len(s) - 1))

A. s[0] != s[-1]: # Base case

B. s[0] != s[len(s)]: # Base case

C. s[1] != s[len(s) - 1]: # Base case

D. s[1] != s[len(s)]: # Base case

A

答案分析：A

15.12分析以下代码：

def xfunction(x, length):

print(x[length - 1], end = " ")

xfunction(x, length - 1)

x = [1, 2, 3, 4, 5]

xfunction(x, 5)

 A.程序显示1 2 3 4 6。

 B.程序显示1 2 3 4 5，然后引发超出范围异常的索引。

 C.程序显示5 4 3 2 1。

 D.程序显示5 4 3 2 1，然后引发超出范围异常的索引。

D

答案分析：D

**第15.5节递归助手功能**

15.13填写代码以完成以下功能，以检查字符串是否为回文结构。

def isPalindrome(s):

return isPalindromeHelper(s, 0, len(s) - 1)

def isPalindromeHelper(s, low, high):

if high <= low: # Base case

return True

elif s[low] != s[high]: # Base case

return False

else:

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

A. isPalindromeHelper(s)

B. isPalindromeHelper(s, low, high)

C. isPalindromeHelper(s, low + 1, high)

D. isPalindromeHelper(s, low, high - 1)

E. isPalindromeHelper(s, low + 1, high - 1)

E

答案分析：E

15.14填写代码以完成以下排序列表的功能。

def sort(lst):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ # Sort the entire list

def sortHelper(lst, low, high):

if low < high:

# Find the smallest number and its index in lst[low .. high]

indexOfMin = low

min = lst[low]

for i in range(low + 1, high + 1):

if lst[i] < min:

min = lst[i]

indexOfMin = i

# Swap the smallest in list(low .. high) with list(low)

lst[indexOfMin] = lst[low]

lst[low] = min

# Sort the remaining list(low+1 .. high)

sortHelper(lst, low + 1, high)

A. sortHelper(lst)

B. sortHelper(lst, len(lst) - 1)

C. sortHelper(lst, 0, len(lst) - 1)

D. sortHelper(lst, 0, len(lst) - 2)

C

答案分析：C

15.15填写代码以完成以下二进制搜索功能。

def recursiveBinarySearch(lst, key):

low = 0

high = len(lst) - 1

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def recursiveBinarySearchHelper(lst, key, low, high):

if low > high: # The list has been exhausted without a match

return ?low - 1

mid = (low + high) // 2

if key < lst[mid]:

return recursiveBinarySearchHelper(lst, key, low, mid - 1)

elif key == lst[mid]:

return mid

else:

return recursiveBinarySearchHelper(lst, key, mid + 1, high)

A. recursiveBinarySearchHelper(lst, key)

B. recursiveBinarySearchHelper(lst, key, low + 1, high - 1)

C. recursiveBinarySearchHelper(lst, key, low - 1, high + 1)

D. recursiveBinarySearchHelper(lst, key, low, high)

D

答案分析：D

15.16以下代码将显示什么？

def main():

times = count("abcabc", 'a')

print(ch + " appears " + str(times) + (" times " if times > 1 else " time ") + "in " + s)

def count(s, a):

return countHelper(s, a, len(s) - 1)

def countHelper(s, a, high):

result = 0;

if high > 0:

result = countHelper(s, a, high - 1) + (1 if s[high] == a else 0)

return result;

main()

 A. a在abcdabc中出现1次

 B. a在abcdabc中出现2次

 C. a在abcdabc中出现1次

 D. a在abcdabc中出现2次

C

答案分析：C

**第15.7节河内塔**

15.17为3个磁盘调用的递归moveDisks函数有多少次？

 A. 3

 B. 7

 C. 10

 D. 14

B

答案分析：B

15.18为4个磁盘调用的递归moveDisks函数有多少次？

 A. 5

 B. 10

 C. 15

 D. 20

C

答案分析：C

15.19分析以下两个程序：

A:

def xfunction(length):

if length > 1:

print(length - 1, end = " ")

xfunction(length - 1)

xfunction(5)

B:

def xfunction(length):

while length > 1:

print(length - 1, end = " ")

xfunction(length - 1)

xfunction(5)

 A.这两个程序产生相同的输出5 4 3 2 1。

 B.这两个程序产生相同的输出1 2 3 4 5。

 C.这两个程序产生相同的输出4 3 2 1。

 D.这两个程序产生相同的输出1 2 3 4。

 E.程序A产生输出4 3 2 1，程序B无限运行。

Ë

答案分析：E

**第15.10节递归与迭代**

15.20以下哪项陈述属实？

 A.递归函数比非递归函数运行得更快。

 B.递归函数通常比非递归函数占用更多的内存空间。

 C.递归函数总是可以用非递归函数代替。

 D.但是，在某些情况下，使用递归可以让您为程序提供一个自然，直接，简单的解决方案，否则难以解决。

BCD

答案分析：BCD

**第15.11节尾递归**

15.21分析以下功能;

def f1(n):

if n == 0:

return 0

else:

return n + f1(n - 1)

def f2(n, result):

if n == 0:

return result

else:

return f2(n - 1, n + result)

print(f1(3))

print(f2(3, 0))

 A. f1是尾递归，但f2不是

 B. f2是尾递归，但f1不是

 C.f1和f2都是尾递归的

 D. f1和f2都不是尾递归的

B

答案分析：B

15.22显示以下代码的输出：

def f2(n, result):

if n == 0:

return 0

else:

return f2(n - 1, n + result)

print(f2(2, 0))

A. 0

B. 1

C. 2

D. 3

A

答案分析：A