6.100L | Fall 2022 | Undergraduate

Introduction To CS And Programming Using Python

Finger Exercises

Finger Exercise Lecture 1

Assume three variables are already defined for you: `a`, `b`, and `c`. Create a variable called `total` that adds `a` and `b` then multiplies the result by `c`. Include a last line in your code to print the value: `print(total)`

6.100L Finger Exercises Lecture 1 Solutions

Finger Exercise Lecture 2

Assume you are given a variable named `number` (has a numerical value). Write a piece of Python code that prints out one of the following strings:

- `positive` if the variable `number` is positive
- `negative` if the variable `number` is negative
- `zero` if the variable `number` is equal to zero

6.100L Finger Exercises Lecture 2 Solutions

Finger Exercise Lecture 3

Assume you are given a positive integer variable named `N`. Write a piece of Python code that prints `hello world` on separate lines, `N` times. You can use either a `while` loop or a `for` loop.

6.100L Finger Exercises Lecture 3 Solutions

Finger Exercise Lecture 4

Assume you are given a positive integer variable named `N`. Write a piece of Python code that finds the cube root of `N`. The code prints the cube root if `N` is a perfect cube or it prints `error` if `N` is not a perfect cube. Hint: use a loop that increments a counter—you decide when the counter should stop.

6.100L Finger Exercises Lecture 4 Solutions

Finger Exercise Lecture 5

Assume you are given a string variable named `my_str`. Write a piece of Python code that prints out a new string containing the even indexed characters of `my_str`. For example, if `my_str = "abcdefg"` then your code should print out `aceg`.

6.100L Finger Exercises Lecture 5 Solutions

Finger Exercise Lecture 6

Assume you are given an integer 0 \<= N \<= 1000. Write a piece of Python code that uses bisection search to guess N. The code prints two lines: `count:` with how many guesses it took to find N, and `answer:` with the value of N. Hints: If the halfway value is exactly in between two integers, choose the smaller one.

6.100L Finger Exercises Lecture 6 Solutions

Finger Exercise Lecture 7

Question 1: Implement the function that meets the specifications below:

```python
def eval_quadratic(a, b, c, x):
    """
    a, b, c: numerical values for the coefficients of a quadratic equation
    x: numerical value at which to evaluate the quadratic.
    Returns the value of the quadratic a×x² + b×x + c.
    """
    # Your code here

# Examples:
print(eval_quadratic(1, 1, 1, 1)) # prints 3
```

Question 2: Implement the function that meets the specifications below:

```
def two_quadratics(a1, b1, c1, x1, a2, b2, c2, x2):
    """
    a1, b1, c1: one set of coefficients of a quadratic equation
    a2, b2, c2: another set of coefficients of a quadratic equation
    x1, x2: values at which to evaluate the quadratics
    Evaluates one quadratic with coefficients a1, b1, c1, at x1.
    Evaluates another quadratic with coefficients a2, b2, c2, at x2.
    Prints the sum of the two evaluations. Does not return anything.
    """
    # Your code here

# Examples:
two_quadratics(1, 1, 1, 1, 1, 1, 1, 1) # prints 6
print(two_quadratics(1, 1, 1, 1, 1, 1, 1, 1)) # prints 6 then None
```

6.100L Finger Exercises Lecture 7 Solutions

Finger Exercise Lecture 8

```
def same_chars(s1, s2):
    """
    s1 and s2 are strings
    Returns boolean True is a character in s1 is also in s2, and vice
    versa. If a character only exists in one of s1 or s2, returns False.
    """
    # Your code here

# Examples:
print(same_chars("abc", "cab"))     # prints True
print(same_chars("abccc", "caaab")) # prints True
print(same_chars("abcd", "cabaa"))  # prints False
print(same_chars("abcabc", "cabz")) # prints False
```

6.100L Finger Exercises Lecture 8 Solutions

Finger Exercise Lecture 9

Implement the function that meets the specifications below:

```
def dot_product(tA, tB):
    """
    tA: a tuple of numbers
    tB: a tuple of numbers of the same length as tA
    Assumes tA and tB are the same length.
    Returns a tuple where the:
    * first element is the length of one of the tuples
    * second element is the sum of the pairwise products of tA and tB
    """
    # Your code here

# Examples:
tA = (1, 2, 3)
tB = (4, 5, 6)
print(dot_product(tA, tB)) # prints (3,32)
```

6.100L Finger Exercises Lecture 9 Solutions

Finger Exercise Lecture 10

Implement the function that meets the specifications below:

```
def all_true(n, Lf):
    """ n is an int
        Lf is a list of functions that take in an int and return a Boolean
    Returns True if each and every function in Lf returns True when called
    with n as a parameter. Otherwise returns False.
    """
    # Your code here

# Examples:
all_true() # prints 6
```

6.100L Finger Exercises Lecture 10 Solutions

Finger Exercise Lecture 11

Implement the function that meets the specifications below:

```
def remove_and_sort(Lin, k):
    """ Lin is a list of ints
        k is an int >= 0
    Mutates Lin to remove the first k elements in Lin and 
    then sorts the remaining elements in ascending order.
    If you run out of items to remove, Lin is mutated to an empty list.
    Does not return anything.
    """
    # Your code here

# Examples:
L = [1,6,3]
k = 1
remove_and_sort(L, k)
print(L)   # prints the list [3, 6]
```

6.100L Finger Exercises Lecture 11 Solutions

Finger Exercise Lecture 12

Implement the function that meets the specifications below:

```
def count_sqrts(nums_list):
    """
    nums_list: a list
    Assumes that nums_list only contains positive numbers and that there are no duplicates.
    Returns how many elements in nums_list are exact squares of elements in the same list, including itself.
    """
    # Your code here

# Examples:
print(count_sqrts([3,4,2,1,9,25])) # prints 3
```

6.100L Finger Exercises Lecture 12 Solutions

Finger Exercise Lecture 13

Implement the function that meets the specifications below:

```
def sum_str_lengths(L):
    """
    L is a non-empty list containing either: 
    * string elements or 
    * a non-empty sublist of string elements
    Returns the sum of the length of all strings in L and 
    lengths of strings in the sublists of L. If L contains an 
    element that is not a string or a list, or L's sublists 
    contain an element that is not a string, raise a ValueError.
    """
    # Your code here

# Examples:
print(sum_str_lengths(["abcd", ["e", "fg"]]))  # prints 7
print(sum_str_lengths([12, ["e", "fg"]]))      # raises ValueError
print(sum_str_lengths(["abcd", [3, "fg"]]))    # raises ValueError
```

6.100L Finger Exercises Lecture 13 Solutions

Finger Exercise Lecture 14

Question 1: Implement the function that meets the specifications below:

```
def keys_with_value(aDict, target):
    """
    aDict: a dictionary
    target: an integer or string
    Assume that keys and values in aDict are integers or strings.
    Returns a sorted list of the keys in aDict with the value target.
    If aDict does not contain the value target, returns an empty list.
    """
    # Your code here

# Examples:
aDict = {1:2, 2:4, 5:2}
target = 2
print(keys_with_value(aDict, target)) # prints the list [1,5]
```

Question 2: Implement the function that meets the specifications below:

```python
def all_positive(d):
    """
    d is a dictionary that maps int:list
    Suppose an element in d is a key k mapping to value v (a non-empty list).
    Returns the sorted list of all k whose v elements sums up to a 
    positive value.
    """
    # Your code here

# Examples:
d = {5:[2,-4], 2:[1,2,3], 1:[2]}
print(all_positive(d))   # prints the list [1, 2]
```

6.100L Finger Exercises Lecture 14 Solutions

Finger Exercise Lecture 15

Implement the function that meets the specifications below:

```python
def recur_power(base, exp):
    """
    base: int or float.
    exp: int >= 0

    Returns base to the power of exp using recursion.
    Hint: Base case is when exp = 0. Otherwise, in the recursive
    case you return base * base^(exp-1).
    """
    # Your code here

# Examples:
print(recur_power(2,5))  # prints 32
```

6.100L Finger Exercises Lecture 15 Solutions

Finger Exercise Lecture 16

Implement the function that meets the specifications below:

```python
def flatten(L):
    """ 
    L: a list 
    Returns a copy of L, which is a flattened version of L 
    """
    # Your code here

# Examples:
L = [[1,4,[6],2],[[[3]],2],4,5]
print(flatten(L)) # prints the list [1,4,6,2,3,2,4,5]
```

6.100L Finger Exercises Lecture 16 Solutions

Finger Exercise Lecture 17

Write the class according to the specifications below:

```
class Circle():
    def __init__(self, radius):
        """ Initializes self with radius """
        # your code here

    def get_radius(self):
        """ Returns the radius of self """
        # your code here

    def set_radius(self, radius):
        """ radius is a number
        Changes the radius of self to radius """
        # your code here

    def get_area(self):
        """ Returns the area of self using pi = 3.14 """
        # your code here

    def equal(self, c):
        """ c is a Circle object
        Returns True if self and c have the same radius value """
        # your code here

    def bigger(self, c):
        """ c is a Circle object
        Returns self or c, the Circle object with the bigger radius """
        # your code here
```

6.100L Finger Exercises Lecture 17 Solutions

Finger Exercise Lecture 18

Write the class according to the specifications below:

```
class Circle():
    def __init__(self, radius):
        """ Initializes self with radius """
        # your code here

    def get_radius(self):
        """ Returns the radius of self """
        # your code here

    def __add__(self, c):
        """ c is a Circle object
        Returns a new Circle object whose radius is
        the sum of self and c's radius """
        # your code here

    def __str__(self):
        """ A Circle's string representation is the radius """
        # your code here
```

6.100L Finger Exercises Lecture 18 Solutions

Finger Exercise Lecture 19

In this problem, you will implement two classes according to the specification below: one `Container` class and one `Stack` class (a subclass of `Container`).

Our `Container` class will initialize an empty list. The two methods we will have are to calculate the size of the list and to add an element. The second method will be inherited by the subclass. We now want to create a subclass so that we can add more functionality—the ability to remove elements from the list. A `Stack` will add elements to the list in the same way, but will behave differently when removing an element.

A stack is a last-in, first-out data structure. Think of a stack of pancakes. As you make pancakes, you create a stack of them with older pancakes going on the bottom and newer pancakes on the top. As you start eating the pancakes, you pick one off the top so you are removing the newest pancake added to the stack. When implementing your `Stack` class, you will have to think about which end of your list contains the element that has been in the list the shortest amount of time. This is the element you will want to remove and return.

```
class Container(object):
    """
    A container object is a list and can store elements of any type
    """
    def __init__(self):
        """
        Initializes an empty list
        """
        self.myList = []

    def size(self):
        """
        Returns the length of the container list
        """
        # Your code here

    def add(self, elem):
        """
        Adds the elem to one end of the container list, keeping the end
        you add to consistent. Does not return anything
        """
        # Your code here

class Stack(Container):
    """
    A subclass of Container. Has an additional method to remove elements.
    """
    def remove(self):
        """
        The newest element in the container list is removed
        Returns the element removed or None if the queue contains no elements
        """
        # Your code here
```

6.100L Finger Exercises Lecture 19 Solutions

Finger Exercise Lecture 20

In this problem, you will implement two classes according to the specification below: one `Container` class and one `Queue` class (a subclass of `Container`).

Our `Container` class will initialize an empty list. The two methods we will have are to calculate the size of the list and to add an element. The second method will be inherited by the subclass. We now want to create a subclass so that we can add more functionality—the ability to remove elements from the list. A `Queue` will add elements to the list in the same way, but will behave differently when removing an element.

A queue is a first-in, first-out data structure. Think of a store checkout queue. The customer who has been in the line the longest gets the next available cashier. When implementing your `Queue` class, you will have to think about which end of your list contains the element that has been in the list the longest. This is the element you will want to remove and return.

```
class Container(object):
    """
    A container object is a list and can store elements of any type
    """
    def __init__(self):
        """
        Initializes an empty list
        """
        self.myList = []

    def size(self):
        """
        Returns the length of the container list
        """
        # Your code here

    def add(self, elem):
        """
        Adds the elem to one end of the container list, keeping the end
        you add to consistent. Does not return anything
        """
        # Your code here

class Queue(Container):
    """
    A subclass of Container. Has an additional method to remove elements.
    """
    def remove(self):
        """
        The oldest element in the container list is removed
        Returns the element removed or None if the stack contains no elements
        """
        # Your code here
```

6.100L Finger Exercises Lecture 20 Solutions

[No Lecture 21 Finger Exercise]

Finger Exercise Lecture 22

Question 1: Simplify `n*n + log(n) + 2**a` to determine θ in terms of `n`.
Question 2: Simplify `2**n + n*log(n) + n**2` to determine θ in terms of `n`.
Question 3: Simplify `f*log(f) + 100000 + 300*a + x*y*z` to determine θ in terms of `n`.

6.100L Finger Exercises Lecture 22 Solutions

Finger Exercise Lecture 23

Question 1: Choose the worst-case asymptotic order of growth (upper and lower bound) for the following function. Assume `n = a`.

```
def running_product(a):
    """ a is an int """
    product = 1
    for i in range(5,a+5):
        product *= i
        if product == a:
            return True
    return False
```

Question 2: Choose the worst-case asymptotic order of growth (upper and lower bound) for the following function. Assume `n = len(L)`.

```
def tricky_f(L, L2):
    """ L and L2 are lists of equal length """
    inL = False
    for e1 in L:
        if e1 in L2:
            inL = True
    inL2 = False
    for e2 in L2:
        if e2 in L:
            inL2 = True
    return inL and inL2
```

Question 3: Choose the worst-case asymptotic order of growth (upper and lower bound) for the following function.

```
def sum_f(n):
    """ n > 0 """
    answer = 0
    while n > 0:
        answer += n%10
        n = int(n/10)
    return answer
```

6.100L Finger Exercises Lecture 23 Solutions

[No Finger Exercises 24–26]

MIT Open Learning

Over 2,500 courses & materials
Freely sharing knowledge with learners and educators around the world. Learn more

Accessibility

Creative Commons License

Terms and Conditions

Proud member of: Open Education GLOBAL

© 2001–2024 Massachusetts Institute of Technology