Introduction to Functional Programming: Part 1

Formative

What's the value of filter(odd, [2, 25, 89, 32, 35])?

[Correct Answer] A. [25, 89, 35]

B. [2, 25, 89, 32, 35]

C. [2,32]

The filter function returns a list consisting of all the numbers for which odd is true. See “Solving a Problem Both Ways.”

Your credit card bill categorizes purchases into categories like Food, Entertainment, and so on. You want to display items in the Food category whose amounts are $10 or more. Which of the following code fragments solves that problem?

[Correct Answer] A. for each purchase in purchasesList  
    if purchase.category == Food  
        if purchase.amount >= 10  
            print purchase

B. for each purchase in purchasesList  
    if purchase.category == Food or purchase.amount >= 10  
       print purchase

C. for each purchase in purchasesList  
    if purchase.amount >= 10 or purchase.category == Food  
        print purchase

D. for each purchase in purchasesList  
    if purchase.amount == Food  
        if purchase.category > 10  
            print purchase

The print purchase statement is executed only if both purchase.category == Food and purchase.amount >= 10 are true. See “Solving a Problem Both Ways.”

Your credit card bill categorizes purchases into categories like Food, Entertainment, and so on. You want to display items in the Food category whose amounts are $10 or more. Which of the following code fragments solves that problem?

[Correct Answer] A. hasCategoryFood(purchase) = purchase.category == Food  
tenOrMore(purchase) = purchase.amount >= 10  
print(filter(tenOrMore, filter(hasCategoryFood, purchasesList)))

B. hasCategoryFood(purchase) = purchase.category == Food  
tenOrMore(purchase) = purchase.amount >= 10  
print(tenOrMore, hasCategoryFood, purchasesList)

C. hasCategoryFood(purchase) = purchase.category == Food  
tenOrMore(purchase) = purchase.amount >= 10  
filter(print(tenOrMore, hasCategoryFood, purchasesList))

D. hasCategoryFood(purchase) = purchase.category == Food  
tenOrMore(purchase) = purchase.amount >= 10  
print(filter(tenOrMore,filter(purchasesList, hasCategoryFood)))

The print statement's argument filters first for Food and then for ten or more. See “Solving a Problem Both Ways.”

What's the value of map(timesTwo, [2, 4, 5])?

[Correct Answer] A. [4, 8, 10]

B. 22

C. [2, 4]

When you apply timesTwo to all three values in the list [2, 4, 5], you get the list [4, 8, 10]. See “Using Filter, Map, and Fold.”

You're given a list of customers. Each customer has a name and an outstanding balance. Which of the following code snippets prints the balance of the customer who has the smallest negative balance?

(Assume that the list has at least one customer in it and that customers' balances range between –1,000 and 1,000. Also, assume that the max function finds the largest of two numeric values.)

[Correct Answer] A. print(foldFromLeft(max, -1000,  
  filter(λ number -> number < 0,  
    map(λ customer -> customer.balance , customersList))))

B. print(foldFromLeft(max, 1000,  
  filter(λ number -> number < 0,  
    map(λ customer -> customer.balance , customersList))))

C. print(foldFromLeft(max, -1000,  
  filter(λ number -> number < 0, customersList))))

D. print(foldFromLeft(max, 1000,  
  map(λ customer -> customer.balance,   
    filter(λ number -> number < 0))))

First, map creates a list of customer balances, then filter creates a list containing only the negative balances, and then foldFromLeft finds the largest of those remaining negative balances. See “Using Filter, Map, and Fold.”

Which of the following is true?

[Correct Answer] A. In functional programming, map behaves like an imperative loop, and filter behaves like an imperative if.

B. In functional programming, map behaves like an imperative if, and filter behaves like an imperative loop.

C. In imperative programming, map behaves like a functional loop, and filter behaves like a functional if.

The map function "iterates" over all values in a list, and the filter function checks to see if a particular condition is true. See “Imperative and Functional Programming Languages.”

In the imperative programming style, we combine statements by performing them one after another, by performing them upon checking an if condition, or by repeating them in a loop. How do we combine evaluations of functions in the functional programming style?

[Correct Answer] A. We take the result of evaluating one function and make that result an argument for evaluating the next function.

B. We filter each function's input.

C. We apply functions to one another in the order in which they're defined in the program.

D. We use each function's result as an argument to every other function.

For example, to find the sum of all food purchases, we take the result of filtering the purchaseList for hasCategoryFood, and make that result be an argument to the map function. See “Imperative and Functional Programming Languages.”

Which of the following *imperative* code fragments correctly displays the sum of the even numbers in a list?

[Correct Answer] A. evenNumbers = filter(even, list)  
total = sum(evenNumbers)  
print(total)

B. print(total)  
total = sum(evenNumbers)  
evenNumbers = filter(even, list)

C. total = sum(evenNumbers)  
evenNumbers = filter(even, list)  
print(total)

The filter function finds the even numbers. Then the total function sums up the even numbers. See “Imperative and Functional Programming Languages.”

Introduction to Functional Programming: Part 2

Formative

Which of the following is a pure function?

[Correct Answer] A. f(x) {  
  return x + 1  
}

B. f(x) {  
  x = x + 7  
  print x  
}

C. f(x) {  
  return x + y  
}

D. f(x) {  
  y = y + x  
  return y  
}

This function is pure. It has no side effects because it uses only its parameter x and it doesn't modify any values that are declared outside of the function. See “Pure Functions.”

Which of the following is a pure function?

[Correct Answer] A. length(the\_string\_s) = number of characters in the\_string\_s

B. f(x) = x + current\_day\_of\_the\_month  
where current\_day\_of\_the\_month is a number from 1 to 31

C. post(message, URL) {  
  add the message to the message board at the URL  
}

This function is pure. It uses only its parameter, s. It returns a value, but it doesn't modify any value that's defined outside the function. See “Pure Functions.”

Which of the following is a referentially transparent expression?

[Correct Answer] A. 7 + 6

B. inputFromKeyboard(x)

(If you execute y = inputFromKeyboard(x), and the user types 7, then the value of y becomes 7.)

C. f(x) = x + current\_day\_of\_the\_month

where current\_day\_of\_the\_month is a number from 1 to 31

This function is referentially transparent. You can substitute 13 for 7 + 6 anywhere in the code. See “Pure Functions.”

Which of the following is an advantage of using pure functions?

[Correct Answer] A. Code that uses pure functions is easier to understand (and thus easier to maintain) than code that uses impure functions.

B. Code that uses pure functions is more efficient (and thus runs faster) than code that uses impure functions.

C. Code that uses pure functions consumes less memory than code that uses impure functions.

There's nothing about a pure function that makes it run faster than an impure function. See “Pure Functions.”

Consider the following code:

total = 0  
for n = 1 to 10 inclusive  
  total = total + n  
print(total)

Which of the following is true about this code? Select all that apply.

[Correct Answer] A. This code prints the value 55.

[Correct Answer] B. This code has a side effect.

[Correct Answer] C. The total variable is mutable.

This code adds 1 + 2 + 3 + . . . + 10 = 55. See “Pure Functions.”

You want to store as much as you can in a bag whose capacity is 9 kilograms. You have objects weighing 7 kilograms, 5 kilograms, and 4 kilograms. If you followed a greedy method to solve the problem, you would start by putting the 7-kilogram object in first. Why is this a bad idea?

[Correct Answer] A. You can store more in the bag if you don't include the 7-kilogram object.

B. A greedy method never yields the correct solution.

C. The function from sets of objects to total weight in the bag isn't a pure function.

D. You should always begin by putting the object with the least weight into the bag.

If you include the 7-kilogram object, you have no room for anything else. But, if you omit the 7-kilogram object, you have room for the 5-kilogram and 4-kilogram objects, giving you a total of 9 kilograms. See “Testing, Debugging, and Memoization.”

Consider this code:

integer amount  
amount = 0  
f(7)  
printToScreen(output)  
  
function f(n) {  
  integer input  
  input = readFromKeyboard()  
  amount = input + n  
  printToScreen(amount)  
}

Which feature of function f is considered to be a side effect? Select all that apply.

[Correct Answer] A. The f function gets input from the keyboard, which is not internal to the function.

[Correct Answer] B. The f function sends output to the screen, which is not internal to the function.

[Correct Answer] C. The f function modifies the value of the amount variable, which is available outside of the function.

D. The function f uses the value of the variable n, but n gets its value 7 from code that's outside of the function.

Since the exact value of a user's keyboard input is not coded into function f, keyboard input is a side effect. See “Testing, Debugging, and Memoization.”

What output(s) may result from running the following code?

x = 0  
three times do {  
  simultaneously do {  
    x = x + 1  
  }  
  and  
  {  
    x = x + 1  
  }  
}  
print x

[Correct Answer] A. 3, 4, 5, or 6

B. Only 3

C. Only 6

D. 1, 2, 3, 4, 5, or 6

During any iteration of the loop, we have two threads executing x = x + 1. If both threads fetch the value of x (on the right side of the assignment) before either thread has updated x (on the left side of the assignment), then the result from both threads is to add only 1 to x. But if one thread updates x before the other thread fetches x, the result from both threads is to add 2 to x. So, the printed value of x depends on how many times one thread updates before the other thread fetches. This can happen 0 times, 1 time, 2 times, or all 3 times. The resulting values of x are 3, 4, 5, and 6. See “Avoiding Race Conditions and Achieving Thread Safety.”

Alice visits an automated teller machine. She checks her account and learns that she has a $400 balance. Then she issues a request to withdraw $100, and the teller machine denies the request. How can this have been caused by a race condition?

[Correct Answer] A. Between the time Alice checked her balance and the time she issued the withdrawal request, Alice's partner made a withdrawal from the same account.

B. Between the time Alice checked her balance and the time she issued the withdrawal request, the bank's computers went down.

C. Alice isn't permitted to withdraw more than $100 during a visit to the teller machine.

D. The teller machine is almost out of cash.

A race condition occurs when two or more agents attempt to modify a particular value at the same time. In this case, the agents are Alice and her partner. The value is the amount in the account. See “Avoiding Race Conditions and Achieving Thread Safety.”

Which of the following phenomena make code susceptible to race conditions? Select all that apply.

[Correct Answer] A. Impure functions

[Correct Answer] B. Shared variables

[Correct Answer] C. Simultaneous threads

D. Memoization

A race condition occurs when two or more agents attempt to modify a particular value at the same time. An impure function may modify the value of a variable that's referenced by code outside the function. The impure function and the code outside the function are two agents, both of which may modify the variable's value. So, a race condition may occur. See “Avoiding Race Conditions and Achieving Thread Safety.”

Which of the following parameter passing schemes is the least safe?

[Correct Answer] A. Pass by reference to an impure function

B. Pass by value to an impure function

C. Pass by reference to a pure function

D. Pass by value to a pure function

Impure functions aren't as safe as pure functions. In addition, pass by reference lets a function point to values that are defined outside the function. Since the function can modify these values that live outside of the function, pass by reference is unsafe. See “Efficient Parameter Passing.”

What's the output of the following code?

x = 7  
f(x)  
print(x)  
Stop  
f(x) {  
    x = x + 1  
}

[Correct Answer] A. We're unable to determine the output without more information.

B. 7

C. 8

D. Runtime error

We're not told whether this code uses pass by value or pass by reference. If the code uses pass by value, the output is 7. But if the code uses pass by reference, the output is 8. See “Efficient Parameter Passing.”

A particular programming language implements parameter passing in three steps:

1. Copy the value of the calling code's variable to a target variable inside the function.

2. Execute the code inside the function.

3. Copy the value of the target variable inside the function back to the calling code's variable.

Which of the following is true about this parameter passing scheme?

[Correct Answer] A. It's not safe and not efficient.

B. It's safe, but it's not efficient.

C. It's efficient, but it's not safe.

D. It's safe and efficient.

This scheme isn't safe because copying a value from inside the function out to the calling code is a side effect. This scheme isn't efficient because it may involve copying a large amount of data from the calling code to the function and then back to the calling code. See “Efficient Parameter Passing.”

In this problem, ++x behaves as both an instruction and an expression. As an instruction, ++x adds 1 to the value of x. As an expression, the value of ++x is the newly obtained value of x. For example, the code

x=7   
print(++x)   
print(x)

displays the numbers 8 8.

With that in mind, what's the output of the following code when the language uses *eager* evaluation?

x = 18  
if ++x > 19 & ++x is even  
    print("\*", x, "\*")  
print(x)

[Correct Answer] A. 20

B. \*19\* 19

C. 19

D. \*20\* 20

With eager evaluation, the code checks both conditions ++x > 19 and ++x is even. The condition ++x > 19 makes the value of x be 19, and this condition is false (because 19 > 19 is false). With eager evaluation, the condition ++x is even makes the value of x be 20. The combined condition, ++x > 19 & ++x is even, is false. So, the code doesn't print \*20\*. Instead, the code goes directly to the last statement, where it prints only the number 20. See “Lazy Evaluation.”

In this problem, ++x behaves as both an instruction and an expression. As an instruction, ++x adds 1 to the value of x. As an expression, the value of ++x is the newly obtained value of x. For example, the code

x=7   
print(++x)   
print(x)

displays the numbers 8 8.

With that in mind, what's the output of the following code when the language uses *lazy* evaluation?

x = 18  
if ++x > 19 & ++x is even  
    print("\*", x, "\*")  
print(x)

[Correct Answer] A. 19

B. 18

C. No output

D. 20

With lazy evaluation, the code checks only one condition: ++x > 19. The condition ++x > 19 makes the value of x be 19, and this condition is false (because 19 > 19 is false). So, the code doesn't print a value with asterisks. Instead, the code goes directly to the last statement, where it prints the number 19. See “Lazy Evaluation.”

A function named firstOf returns the first of two values in a pair of arguments. For example, firstOf(25, 13) is 25. Consider the following code:

x = 0  
print(firstOf(x, 1/x))

What's the output of this code when the system uses *eager* evaluation?

[Correct Answer] A. An error message

B. 0

C. 1/0

D. No output, because there’s an endless loop

With eager evaluation, the system tries to evaluate both x and 1/x. Evaluation of x goes smoothly, but evaluation of 1/x causes an error because we can't divide by 0. See “Lazy Evaluation.”

A function named firstOf returns the first of two values in a pair of arguments. For example, firstOf(25, 13) is 25. Consider the following code:

x = 0  
print(firstOf(x, 1/x))

What's the output of this code when the system uses *lazy* evaluation?

[Correct Answer] A. 0

B. An error message

C. No output, because there's an endless loop

D. 1/0

With lazy evaluation, the system doesn't try to evaluate 1/x. So, the program evaluates x and prints that value, which is 0. See “Lazy Evaluation.”

Introduction to Functional Programming: Summative Quiz

Summative

What's the value of filter( isMammal, [cat, fly, rose] )?

[Correct Answer] A. [cat]

B. cat

C. []

D. [fly, rose]

The filter function returns a list containing all the items for which isMammal is true.

Learning Objective: Understand what distinguishes functional programming from other programming paradigms. Review “Part 1: Getting Started with Functional Programming.”

What's the value of the following expression?

sum( filter( isPositive, map(subtractTenFrom,[9,15,21,10,-4]) ) )

[Correct Answer] A. 16

B. 55

C. 45

D. 51

The expression's value is:

  sum( filter( isPositive, [-1,5,11,0,-14] ) )  
= sum( [5,11] )  
= 16

Learning Objective: Understand what distinguishes functional programming from other programming paradigms. Review “Part 1: Getting Started with Functional Programming.”

What's the value of the following expression?

foldFromLeft( smallerOf, 100, [9, 18, 21, 1] )

[Correct Answer] A. 1

B. 100

C. [9, 18, 21, 1]

D. None of the other choices are correct.

The value of foldFromLeft( smallerOf, 100, [9, 18, 21, 1] ) is:

  smallerOf(smallerOf(smallerOf(smallerOf(100,9),18),21),1)  
= smallerOf(smallerOf(smallerOf(      9         ,18),21),1)  
= smallerOf(smallerOf(             9                ,21),1)  
= smallerOf(                    9                       ,1)  
= 1

Learning Objective: Understand what distinguishes functional programming from other programming paradigms. Review “Part 1: Getting Started with Functional Programming.”

A *side effect* occurs when...

[Correct Answer] A. Evaluating an expression interacts with something other than the expression itself.

B. Your code involves an infinite loop.

C. You call a function with no parameters and no return value.

D. Evaluating an expression requires a call to a function.

"Something other than the expression" is what's on the side in the term *side effect*.

Learning Objective: Distinguish pure functions from impure functions; declaring and evaluating pure functions. Review “Part 2: Functions.”

Which of the following is an example of a *side effect*?

[Correct Answer] A. Referring to the same variable both inside and outside of a function

B. Calling a recursive function that has no base case

C. Attempting to call a function with an argument of the wrong type

D. Execution of any function written in an imperative programming style

For example, in the following code, the fact that x is referenced inside and outside of the function f constitutes a side effect:

x = 0  
f(7)  
  
f(y) {  
    x = x + y  
}

Learning Objective: Distinguish pure functions from impure functions; declaring and evaluating pure functions. Review “Part 2: Functions.”

A *pure function* is...

[Correct Answer] A. A function with no side effects

B. A function with no output to the screen or to a file

C. A recursive function

D. A function with parameters and a return value

Any side effect makes a function impure.

Learning Objective: Distinguish pure functions from impure functions; declaring and evaluating pure functions. Review “Part 2: Functions.”

Which of the following is *not* an advantage of using pure functions?

[Correct Answer] A. They can call themselves recursively.

B. They can be reused without regard to the context in which they're run.

C. They can be tested without any setup.

D. They can be memoized.

A function that calls itself recursively may have side effects. Both pure and impure functions may call themselves recursively.

Learning Objective: Distinguish pure functions from impure functions; declaring and evaluating pure functions. Review “Part 2: Functions.”

What's the output of the following code?

x = 7  
f(x)  
print(x)  
Stop  
f(x) {  
    x = x + 1  
}

[Correct Answer] A. 7 with pass by value; 8 with pass by reference

B. 7

C. 8

D. 7 with pass by reference; 8 with pass by value

With pass by value, when a function changes the value of one of its parameters, that change affects the value in the code that calls the function. With pass by reference, the change doesn't affect the value in the calling code.

Learning Objective: Distinguish pure functions from impure functions; declaring and evaluating pure functions. Review “Part 2: Functions.”

In a language with lazy evaluation, what's the output of the following code?

f(x) = x + f(x)  
g(x,y) = x + x  
print g(3,f(5))

[Correct Answer] A. 6

B. 5 + 5 + 5 + ...

C. No output. The program runs forever.

D. No output. The program stops running and displays an error message.

With lazy evaluation, the system doesn't try to find the value of f(5). So the value that's printed is g(3,f(5)) = 3 + 3 = 6.

Learning Objective: Distinguish pure functions from impure functions; declaring and evaluating pure functions. Review “Part 2: Functions.”

In a language with eager evaluation, what's the output of the following code?

x = 2  
print secondElementOf([0\*x,3\*x,6\*x,9\*x,...])

[Correct Answer] A. No output with an infinite loop or terminating error

B. 6

C. [6]

D. 12

With eager evaluation, the system tries to find infinitely many values in the list before looking for a second element.

Learning Objective: Distinguish pure functions from impure functions; declaring and evaluating pure functions. Review “Part 2: Functions.”

In JavaScript, the call setTimeout(code,delay) executes the instructions in the code after waiting at least delay milliseconds. Which of the following statements is true about the setTimeout function?

[Correct Answer] A. setTimeout is a higher-order function.

B. The instructions in the code must be a pure function.

C. The instructions in the code must be referentially transparent.

D. setTimeout is a composite function.

In the call setTimeout(code,delay), the code is a set of instructions to be executed. In other words, the code is a function.

Learning Objective: Declare and evaluate functions whose parameters or result values (or both) are functions. Review “Part 3: Higher-Order Functions.”

An empty list has no tail. But we can define

tailMaybe([]) = Nothing  
tailMaybe(h::t) = Just t

What's the value of tailMaybe([3,2]) >>= tailMaybe?

[Correct Answer] A. Just []

B. []

C. Undefined or error message

D. Nothing

tailMaybe([3,2])  
= tailMaybe(3::[2])   
= Just [2]

and

Just [2] >>= tailMaybe   
= Just 2::[] >>= tailMaybe   
= Just []

Learning Objective: Understand monad operations and use them to create elegant code. Review “Part 5: Monads.”

An empty list has no tail. But we can define

tailMaybe([]) = Nothing  
tailMaybe(h::t) = Just t

What's the value of the following expression?

tailMaybe([3,2]) >>= tailMaybe >>= tailMaybe >>= tailMaybe

[Correct Answer] A. Nothing

B. Just []

C. Undefined or error message

D. []

tailMaybe([3,2])  
= tailMaybe(3::[2])   
= Just [2]

and

Just [2] >>= tailMaybe  
= Just 2::[] >>= tailMaybe  
= Just []

So,

tailMaybe([3,2]) >>= tailMaybe >>= tailMaybe  
= Just [] >>= tailMaybe  
= Nothing

And once you have Nothing in a chain of Maybe binds, all binds that come later in the sequence have the Nothing value.

Learning Objective: Understand monad operations and use them to create elegant code. Review “Part 5: Monads.”