Boost up Your Certification Score

APPLE

pp-Development-with-Swift-Certified-U

App Development with Swift Certified User Exam

For More Information – Visit link below:

https://www.examsboost.com/

Product Version

✓ Up to Date products, reliable and verified.
✓ Questions and Answers in PDF Format.

Latest Version: 6.0

Question: 1

Review the code.
var capitalCities = [" USA " : " Washington D.C. " , " Spain " : " Madrid " , " Peru " : " Lima "]
Which two statements add the capital city of " Italy " to the dictionary? (Choose 2.)

A. capitalCities[" Rome "] = " Italy "
B. capitalCities.append([" Italy " : " Rome "])
C. capitalCities[" Italy "] = " Rome "
D. capitalCities = capitalCities + [" Italy " : " Rome "]
E. capitalCities.updateValue(" Rome " , forKey: " Italy ")

Answer: C, E

Explanation:
Comprehensive and Detailed Explanation From App Development with Swift domains:
This question falls under Swift Programming Language , specifically the domain for managing data using collection types , with emphasis on dictionaries . In Swift, a dictionary stores data as key-value pairs , so in this example the country name is the key and the capital city is the value. To add a new entry, Swift supports two standard approaches. The first is subscript assignment , which is shown in option C : capitalCities[" Italy "] = " Rome " . Apple's documentation explains that you can add a key-value pair to a dictionary by assigning a value for a new key through the dictionary subscript.
The second correct approach is option E : capitalCities.updateValue(" Rome " , forKey: " Italy "). Apple documents that updateValue(_:forKey:) updates the value for an existing key, or adds a new key-value pair if the key does not already exist . That makes it equally valid for inserting " Italy " : " Rome " into the dictionary.
The incorrect options fail for different reasons. A reverses the key and value, making " Rome " the key and " Italy " the value. B is wrong because append is used with arrays, not dictionaries. D is not the standard valid insertion syntax for Swift dictionaries in this context; Swift's documented mutation approaches here are subscript assignment and updateValue. Therefore, the two correct answers are C and E .

Question: 2

Which two assignments of a value to direction are allowed? (Choose 2.)

```swift
enum CompassPoint {
    case north, south, east, west
}

var direction; CompassPoint
```

A. direction = CompassPoint(north)
B. direction = CompassPoint.north
C. direction = north
D. direction = .north
E. direction = direction.north

Answer: B, D

Explanation:
This question belongs to Swift Programming Language , specifically the domain covering basic Swift types and how Swift handles enumerations . The code defines an enum named CompassPoint with the cases north, south, east, and west, and then declares direction as type CompassPoint. In Swift, an enum case can be assigned using the fully qualified form CompassPoint.north, so B is valid. Swift also allows the shorthand form .north when the compiler already knows the expected type is CompassPoint, so D is also valid. Apple's Swift language documentation explains that once a variable is known to be of a specific enumeration type, you can set its value using the shorter dot syntax.
The other options are not allowed. A is invalid because enum cases are not assigned using constructor-style syntax like CompassPoint(north). C is invalid because north by itself is not enough unless it is written with dot shorthand in a context with inferred enum type. E is invalid because north is a case of the enum type, not a member accessed from the variable instance as direction.north. Swift enum cases are referenced from the enum type or by shorthand dot syntax, not as instance properties.

Question: 3

Review the code.
var capitalCities = [" USA " : " Washington D.C. " , " Spain " : " Madrid " , " Peru " : " Lima "]
Which two statements add the capital city of " Italy " to the dictionary? (Choose 2.)

A. capitalCities[" Rome "] = " Italy "
B. capitalCities.append([" Italy " : " Rome "])
C. capitalCities[" Italy "] = " Rome "
D. capitalCities = capitalCities + [" Italy " : " Rome "]
E. capitalCities.updateValue(" Rome " , forKey: " Italy ")

Answer: C, E

Explanation:

Comprehensive and Detailed Explanation From App Development with Swift domains:
This question falls under Swift Programming Language , specifically the domain for managing data using collection types , with emphasis on dictionaries . In Swift, a dictionary stores data as key-value pairs , so in this example the country name is the key and the capital city is the value. To add a new entry, Swift supports two standard approaches. The first is subscript assignment , which is shown in option C : capitalCities[" Italy "] = " Rome " . Apple's documentation explains that you can add a key-value pair to a dictionary by assigning a value for a new key through the dictionary subscript.
The second correct approach is option E : capitalCities.updateValue(" Rome " , forKey: " Italy "). Apple documents that updateValue(_:forKey:) updates the value for an existing key, or adds a new key-value pair if the key does not already exist . That makes it equally valid for inserting " Italy " : " Rome " into the dictionary.
The incorrect options fail for different reasons. A reverses the key and value, making " Rome " the key and " Italy " the value. B is wrong because append is used with arrays, not dictionaries. D is not the standard valid insertion syntax for Swift dictionaries in this context; Swift's documented mutation approaches here are subscript assignment and updateValue. Therefore, the two correct answers are C and E .

Question: 4

Review the code snippet.

```
1  struct NewAppleProduct (
2      van name: String
3      var releaseDate: String?
4  )
9  var lastReleaseDate = "2020-04-17"
6  let nextApplePhone = NewAppleProduct(name: "iPhone 12", releaseDate: nil)
7  lastReleaseDate = nextApplePhone,releaseDate != nil ? _____________________________
```

Which statement completes the code snippet so that:
• The lastReleaseDate remains the same when nextApplePhone.releaseDate is nil.
• The lastReleaseDate updates to the nextApplePhone.releaseDate when nextApplePhone.releaseDate is NOT nil.

A. nextApplePhone.releaseDate : lastReleaseDate
B. lastReleaseDate : nextApplePhone.releaseDate
C. lastReleaseDate : nextApplcPhone.rcleaseDate!
D. nextApplePhone.releaseDate! : lastReleaseDate

Answer: D

Explanation:
This question is from Swift Programming Language , especially the domains for Optional types , safe and unsafe unwrapping , and control flow . The code uses the ternary conditional operator :
nextApplePhone.releaseDate != nil ? _____
In Swift, the ternary operator follows this structure:
condition ? valueIfTrue : valueIfFalse
So if nextApplePhone.releaseDate != nil is true, the expression must return the new release date. If it is false, it must keep lastReleaseDate unchanged. That means the missing part must be:
nextApplePhone.releaseDate! : lastReleaseDate

which is option D .
This works because nextApplePhone.releaseDate is declared as String?, so it is an optional. Once the condition confirms it is not nil, the code force-unwraps it with ! to access the underlying String value. If the optional is nil, the expression returns lastReleaseDate instead. Apple's Swift documentation describes the ternary conditional operator as a shortcut for choosing one of two expressions based on a condition, and it explains that force unwrapping with ! accesses an optional's wrapped value when you know it is not nil.
The other options are incorrect because they reverse the true/false logic, omit the needed unwrap, or contain invalid identifiers. Therefore, the correct completion is D .

Question: 5

Review the code.

```
1   struct FirstView: View {
2       @State var number: String = ""
5       var body: some View {
4          NavigationStack {
9              TextField("Enter a number", text: Snumber)
6              Navigationlink("Send The Number") {
7                  SecondView(passedNumber: Int(number) ?? 0)
8              }
9          }
10      }
11  }
12
13  struct SecondView: View {
14      @State var passedNumber: Int
16      var body: some View {
16          Color(passedNumber == 3 ? .blue : .red)
17      }
    }
```

When entered into the TextField, which number will display a blue canvas on the SecondView?

A. 3
B. 2
C. 4
D. 1

Answer: A

Explanation:
This question belongs to View Building with SwiftUI , especially the domain on creating multiple views to implement app logic and sharing values between views. In FirstView, the value typed into the TextField is stored in number, which is a String. When the NavigationLink is tapped, the code passes Int(number) ?? 0 into SecondView. In Swift, converting a string to an integer with Int(...) uses an optional initializer, which means it returns an optional value and will produce nil if the string cannot be converted. The ?? 0 nil-coalescing operator then supplies 0 if conversion fails.
Inside SecondView, the body is:

Color(passedNumber == 3 ? .blue : .red)
This uses the ternary conditional operator. If passedNumber equals 3, the displayed color is blue; otherwise, it is red. Therefore, entering 3 into the TextField causes Int(number) to become 3, which makes the condition passedNumber == 3 true and displays a blue canvas. Any other listed number results in red.
So the correct answer is A. 3 . This matches the SwiftUI pattern of taking user input, converting it to the needed type, passing it into another view, and rendering UI conditionally based on that value.

Question: 6

Review the code snippet and then predict the output.

```
var count = 0
let max = 101
for num in 0..<max {
    guard num % 5 == 0 && num % 2 != 0 else {
        continue
    }
    count += 1
}
print("Total count:\(count)")
```

Options:

A. Total count: 11
B. Total count: 10
C. Total count: 20
D. Total count: 9

Answer: B

Explanation:
This question belongs to Swift Programming Language , especially the domains covering control flow , loops , logical operators , and guard . The loop runs through 0.. < max, and since max = 101, the values of num are 0 through 100. Inside the loop, the guard statement keeps only values that satisfy both conditions:
num % 5 == 0 → the number must be divisible by 5
num % 2 != 0 → the number must be odd
So the code counts numbers from 0 to 100 that are odd multiples of 5 . Those values are:
5, 15, 25, 35, 45, 55, 65, 75, 85, 95
That gives a total of 10 numbers. Therefore count becomes 10, and the printed output is:
Total count: 10
The key Swift concept here is that guard ... else { continue } skips any loop iteration that does not meet the required condition. Only matching values reach count += 1. This is a standard use of guard for early exit and of the remainder operator % for divisibility checks. Therefore, the correct answer is B .

Thank You for Trying Our Product

For More Information – Visit link below:

https://www.examsboost.com/

15 USD Discount Coupon Code:

G74JA8UF

FEATURES

- ✓ 90 Days Free Updates
- ✓ Money Back Pass Guarantee
- ✓ Instant Download or Email Attachment
- ✓ 24/7 Live Chat Support
- ✓ PDF file could be used at any Platform
- ✓ 50,000 Happy Customer