Exam 2

Due Nov 5 at 3:20pm **Points** 50 **Questions** 14
Available Nov 5 at 2:20pm - Nov 5 at 3:20pm about 1 hour **Time Limit** 60 Minutes

Instructions

Content and timing

This exam will evaluate your understanding of all concepts discussed in the course thus far. It has 14 questions that you need to answer in one hour. You have one attempt at this exam.

General instructions

This is an open notes exam. You are free to look at your notes, any of the resources we used in class, or any website. However, you are not allowed to talk to your classmates or any other person to get help with the exam. You should not help your classmates with the exam or share any information about the exam to anyone. Any form of cheating will not be tolerated and will result in a grade of 0 for the exam and disciplinary action.

Exam-taking tips

I want you to complete the exam in time. Here are some tips that can help you answer it successfully.

- Answer questions that you think are easier first. Anything that takes you more than a minute to answer is "hard". Skip hard questions then come back for them later. This exam allows you to review and change your previous answers.
- Reading other questions can help you recall answers. In many cases, reading similar questions may help you answer other questions. Some questions may even offer hints that help you answer other questions. This gives you more reasons to skip "hard questions."
- Write the problem number in a piece of paper for review. This will help you review questions you skipped and ensure you answer all questions.
- Answer all questions. The chances of getting points for a question is 0 if you don’t provide an answer. It’s better to guess than to leave a problem blank.

Good luck!

This quiz was locked Nov 5 at 3:20pm.

Attempt History

Attempt	Time	Score

	Attempt	Time	Score
LATEST	Attempt 1	58 minutes	37 out of 50

Score for this quiz: **37** out of 50
Submitted Nov 5 at 3:19pm
This attempt took 58 minutes.

Question 1

0 / 0 pts

By selecting "I agree", I indicate my understanding of the University Policies on Academic Honesty. I will exercise complete academic honesty by not giving or receiving any unauthorized assistance on this examination or engage in any form of cheating. I will not share any information about the exam to anyone. I also acknowledge that any confirmed act of dishonesty will result in the disqualification of this exam and appropriate sanctions, including formal conduct charges and possible course failure.

- ○ I disagree
- ◉ I agree

Correct!

Question 2

2 / 2 pts

What data type would you use to allow a function to receive an Int or nil value as its parameter?

```
func display(num: Int? ) {
  if num == nil {
    print("Not a number")
  } else {
    print(num)
  }
}
```

display(num: 5)
display(num: nil)

Answer 1:

Correct!

Int?

Question 3

3 / 3 pts

Why do we use the if-let statement when creating an instance of a struct or class that uses a failable initializer?

We should use an if-var statement instead because initializers do not return a constant value.

We use an if-let statement because failable initializers return a Bool value that indicates whether an instance is created successfully or not.

You do not need to use the if-let statement because struct or classes with failable initializers will always return a valid instance.

Correct!

A struct or class with a failable initializer may return nil. The if-let statement will allow us to store the instance in a constant when it is successfully created.

Question 4

2 / 2 pts

What do we call the method of conditionally unwrapping composed optional properties? For example:

```
if let zip = dog.owner?.address?.zipCode {
  print(zip)
}
```

Correct!

optional chaining

orrect Answers

Optional Chaining

Optional chaining

optional chaining

Question 5

0 / 3 pts

Which among the following explanations accurately describe the code below?

```
class Fruit {
    var name: String = ""
    func consume() { print("consume") }
}

class Apple : Fruit {
    override init() {
        super.init()
        self.name = "Apple"
    }
    func core() { print("core") }
    func peel() { print("peel") }
    override func consume() { print("crunch crunch") }
}

class Orange : Fruit {
    override init() {
        super.init()
        self.name = "Orange"
    }
    func squeeze() { print("squeeze") }
    override func consume() { print("squish") }
}
```

```
var food: Fruit
food = Apple()

if let snack = food as? Apple {
  snack.consume()
} else {
  print("I don't like this snack.")
}
```

ou Answered

- The Apple object is stored in a variable called food whose data type is Fruit. It checks whether food's data type is an Apple. It stores the the Fruit instance called food into snack because it is an Apple. When snack.consume() is called, it uses Fruit's implementation of the function.

orrect Answer

- The Apple object is stored in a variable called food whose data type is Fruit. It tries to typecast food into an Apple and stores it into snack. It succeeds because food is in fact an Apple. When snack.consume() is called, it uses Apple's implementation of the function.
- The Apple object is stored in a variable called food whose data type is Fruit. It tries to typecast food into an Apple, but fails because a Fruit is not an Apple. It prints "I don't like this snack." on the screen.
- The code is syntactically incorrect, because you cannot assign an Apple instance into a Fruit variable.

Question 6

0 / 3 pts

What data type can we use for the display function's data parameter so that it allows us to pass instances of all the objects declared below.

```
class Fruit {
   var name: String = ""
   func consume() { print("consume") }
```

```
}

class Apple : Fruit {
    override init() {
        super.init()
        self.name = "Apple"
    }
    func core() { print("core") }
    func peel() { print("peel") }
    override func consume() { print("crunch crunch") }
}

class Orange : Fruit {
    override init() {
        super.init()
        self.name = "Orange"
    }
    func squeeze() { print("squeeze") }
    override func consume() { print("squish") }
}

class Fish {
    func swim() { print(swim) }
}

func display(_ data: Fruit? ) {
    print(data)
}

var apple = Apple()
var orange = Orange()
var fish = Fish()
display(apple)
display(orange)
display(fish)
```

Answer 1:

ou Answered

Fruit?

orrect Answer

Any

Question 7

0 / 3 pts

When will the print statement be executed in the code below?

```
func setAge(age: Int) {
  guard age >= 18 else {
    print("You are \(age) years old")
    return
  }
  self.age = age
}
```

- After the guard block finishes executing.

ou Answered

- When the value of age is greater than or equal to 18.

orrect Answer

- When the value of age is less than 18.
- It will never be called.

Question 8

7 / 7 pts

Which among the following are required to create a struct that implements the Printable and CustomStringConvertible protocols? Make sure to only select the minimum required options, otherwise points will be deducted from your answer.

```
protocol Printable {
  var pages: Int { get set }
  func printableFormat() -> String
}
```

ou Answered

- [x] Create a computed property called pages that returns the number of pages represented by an Int.

Correct!

- [x] Inherit both protocols by adding them after the structure definition. For example,

```
struct Paper Printable, CustomStringConvertible {
  // code
}
```

Correct!

- [x] Create a property called description.
- [ ] Create a function called start.

Correct!

- [x] Create a printableFormat function.

orrect Answer

- [ ] Create a property called pages.
- [ ] Inherit Printable protocol by adding them after the structure definition. For example,

```
struct Paper: Printable {
  // code
}
```

Question 9

2 / 4 pts

Which among the following show a valid use of the Color enumeration as a parameter to the setPixel function?

```
enum Color {
  case red, green, blue
}
```

```
func setPixel(x: Int, y: Int, color: Color) {
  // code here
}
```

orrect Answer

- [ ] setPixel(x: 0, y: 0, color: Color.green)
- [ ] setPixel(x: 0, y: 0, color: Color.yellow)
- [ ] setPixel(x: 0, y: 0, color: .orange)

Correct!

- [x] setPixel(x: 0, y: 0, color: .blue)

Question 10

4 / 4 pts

Complete the function prototype so that the callback parameter accepts a closure. The closure accepts a String and Int parameter but does not return anything.

func send(to: String, data: String, callback: (String, Int) -> Void) {

// function body

}

Answer 1:

Correct!

- (String, Int) -> Void

orrect Answer

- (String,Int) -> Void

orrect Answer

- (String, Int)->Void

orrect Answer

- (String,Int)->Void

Question 11

2 / 4 pts

Which among the following closures will add all even numbers when used with the reduce function on numbers?

```
var numbers = [1, 2, 4, 3, 9, 5, 8]
```

Correct!

- [x]
```
let sum = numbers.reduce(0) {
    (oldValue, newValue) -> Int in
    if newValue % 2 == 0 {
        return oldValue + newValue
    } else {
        return oldValue
    }
}
```

- [ ]
```
let sum2 = numbers.reduce(0) {
    if $0 % 2 == 0 {
        return $1 + $0
    } else {
        return $1
    }
}
```

- [ ]
```
let sum = numbers.reduce(0) {
    if newValue % 2 == 0 {
        return currentTotal + newValue
    } else {
        return currentTotal
    }
}
```

orrect Answer

```
let sum = numbers.reduce(0) {
    if $1 % 2 == 0 {
        return $0 + $1
    } else {
        return $0
    }
}
```

Question 12

5 / 5 pts

What are the advantages of using enumerations?

Your Answer:

In my head, when i think of enums, i think of a bag. this bag is special in that it contains items that are related simply bc they exist in the same bag together. This is useful when you want to work with type-safe values within your program. You know that you will be picking from a safe bag of values, rather than something outside of the enum.

Question 13

5 / 5 pts

How do protocols help developers build projects that are extensible? Extensibility is the ability to extend a system and the level of effort required to implement the extension.

Your Answer:

protocols are nice bc the protocol does not contain the implementation of the functions or properties, however they do enforce the usage of such items listed within the protocol. This means a team can all use the same protocol and have differing implementations, however they must implement all of the properties and functions from the protocol. In this way,

a program can move forward in design and devs can assume certain functions are usable if the protocol is used.

Question 14

5 / 5 pts

Why does Swift promote the use of the guard statement instead of using nested if statements?

Your Answer:

the guard statement greatly reduces the need for if statenments. and this is intended to reduce the confusion that comes with many nested if statements. For example, a nested if to determine a valid password from the pogil activity. it has a lot of things to check and the nested if logic is hard to follow. However if we instead check for certain things it is not supposed to have which will return invalid automatically, then we dont need all of the nested if statements, we can return immediately.

Quiz Score: **37** out of 50