Основы Swift

# Exceptions

# Error

```
enum ProcessingError: Error {
    case incorectInput
    case noConnection
    case internalError(errorCode: Int)
}
```

# Выбрасывание исключения

```
throw ProcessingError.internalError(errorCode: 2)
```

# Бросание исключения из функции

```
func canThrowErrors() throws -> String {
    throw ProcessingError.internalError(errorCode: 2)
}
```

# Исключение в инициализаторе

```
struct MyStruct {
    init(parameter: Int) throws {
        guard parameter != 0 else {
            throw ProcessingError.incorectInput
        }
    }
}
```

# Передача выше по стеку

```
func canThrowErrorsTo() throws {
    try canThrowErrors()
}
```

# catch

```
do {
    try canThrowErrors()
} catch ProcessingError.incorectInput {
    // ...
} catch ProcessingError.noConnection {
    // ...
} catch ProcessingError.internalError(let errorCode)
where errorCode == 1 {
    // ...
} catch ProcessingError.internalError(let errorCode) {
    // ...
} catch let anotherError {
    // ...
}
```

# try?

```
let result = try? canThrowErrors()
type(of: result) // Optional<String>
```

# try!

```
try! canThrowErrors()
```

# Использование guard let и if let

```
guard let result = try? canThrowErrors() else {
    return
}

// или

if let result = try? canThrowErrors() {
    // ...
} else {
    // ...
}
```

# Исключение - это enum

```
enum Result<T> {
    case fail(Error)
    case success(T)
}
```

# Использование своего типа

```
func canThrowErrorUsingEnum() -> Result<String> {
    return .fail(ProcessingError.internalError
                                    (errorCode: 2))
}
```

# Метод для создания ссылки в Objective-C

```
- (BOOL)createSymbolicLinkAtURL:(NSURL *)url
withDestinationURL:(NSURL *)destURL
error:(NSError * _Nullable
*)error;
```

# Метод для создания ссылки в Swift

```
open func createSymbolicLink(at url: URL,
withDestinationURL destURL: URL)
throws
```

# Обработка исключения

```
do {
    try FileManager.default.createSymbolicLink(at:
    URL(string: "/")!,
withDestinationURL: URL(string: "/")!)
} catch let error {
    print("\(error.localizedDescription)")
    // The file couldn’t be saved because the specified
                    URL type isn’t supported.
}
```

# Exception в замыкании

```
let linkDestinations = [URL(string: "/somePath/file")!,
                        URL(string: "/somePath2/file")!,
                        URL(string: "/anotherPath/file")!]
linkDestinations.forEach {
    url in
        do {
            try FileManager.default.createSymbolicLink(at:
            URL(string: "/")!, withDestinationURL: url)
        } catch let error {
            // ...
    }
}
```

# Объявление forEach

```
func forEach(_ body: (Element) throws -> Void)
rethrows
```

# Использование rethrows

```
do {
    try linkDestinations.forEach {
        url in

        try FileManager.default.createSymbolicLink(at:
                                URL(string: "/")!,
withDestinationURL: url)
    }
} catch let error {
    // ...
}
```

# В данном случае исключений нет

```
linkDestinations.forEach {
    url in

    print("\(url)")
}
```

# defer

```
defer {
    // Какие-то операции
}

try? canThrowErrors()
```

# Простейший итератор

```
struct MultiplicityIterator: IteratorProtocol {
    var base = 1

    public mutating func next() -> Int? {
        defer { base += base }
        return base
    }
}
```