Swift 치트시트

1. 기본: 상수, 변수, 타입

- 상수/변수: `let` (상수), `var` (변수). 타입 추론을 지원.
- 타입 명시: `var welcomeMessage: String`
- 데이터 타입: `Int`, `Double`, `Float`, `Bool`, `String`, `Character`.
- 튜플: `let http404Error = (404, "Not Found")`
- 타입 별칭 (Type Alias): `typealias AudioSample = UInt16`

2. 옵셔널 (Optionals)

값이 없을 수 있는 상황을 안전하게 처리하기 위한 핵심 기능.

- 선언: `var optionalString: String? = "Hello"`
- 강제 언래핑 (Forced Unwrapping): `optionalString!` (주의: nil일 경우 런타임 에러 발생)
- 옵셔널 바인딩 (Optional Binding):

```
if let constantName = someOptional {
    // someOptional에 값이 있으면 constantName으로 사용
}
```

- 가드 구문 (Guard Statement):

```
guard let constantName = someOptional else {
    // someOptional이 nil이면 이 블록이 실행됨 (주로 return, throw)
    return
}
// constantName을 여기서부터 사용 가능
```

- Nil-Coalescing 연산자: `a ?? b` (a가 nil이 아니면 a의 언래핑된 값, nil이면 b)
- 옵셔널 체이닝 (Optional Chaining): `resident?.printNumberOfRooms()`
- 암시적 언래핑 옵셔널 (Implicitly Unwrapped): `var assumedString: String!` (선언 후 즉시 값이 할당될 것이 확실할 때 사용)

3. 컬렉션 타입

- 배열 (Array): `var shoppingList: [String] = ["Eggs", "Milk"]`
- 집합 (Set): `var favoriteGenres: Set<String> = ["Rock", "Classical"]`
- 딕셔너리 (Dictionary): `var airports: [String: String] = ["YYZ": "Toronto Pearson"]`

4. 제어 흐름

- for-in: `for item in shoppingList { ... }`
- while: `while condition { ... }`
- repeat-while: `do-while`과 유사.
- if-else: `if ... else if ... else { ... }`
- switch: C와 달리 `break`가 필요 없으며, 모든 경우를 다루어야 함.

```
switch someValue {
case 1:
    // ...
case 2...5: // 범위 매칭
    // ...
case let (x, 0): // 튜플 및 값 바인딩
    // ...
case let (x, y) where x == y: // where 절
    // ...
default:
    // ...
}
```

5. 함수와 클로저

- 함수: `func greet(person: String, from hometown: String) -> String { ... }`
  - `_` : 인수 레이블 생략.
- 가변 매개변수: `func arithmeticMean(_ numbers: Double...) -> Double`
- in-out 매개변수: `func swapTwoInts(_ a: inout Int, _ b: inout Int)`
- 함수 타입: `(Int, Int) -> Int`
- 클로저 (Closures): 이름 없는 코드 블록.
  - `reversedNames = names.sorted(by: { (s1: String, s2: String) -> Bool in return s1 > s2 })`
  - 문맥에서 타입 추론: `names.sorted(by: { s1, s2 in s1 > s2 })`
  - 단축 인수 이름: `names.sorted(by: { $0 > $1 })`
  - 연산자 메서드: `names.sorted(by: >)`
- 후행 클로저 (Trailing Closure): 함수의 마지막 인수가 클로저일 때 `()` 밖에 작성 가능.
- `@escaping` 클로저: 함수가 반환된 후에도 클로저가 호출될 필요가 있을 때.

6. 구조체와 클래스 (Structures and Classes)

- 공통점: 프로퍼티 정의, 메서드 정의, 서브스크립트, 초기화 구문.
- 차이점:
  - 클래스: 상속, 타입 캐스팅, 소멸자(`deinit`), 참조 카운팅.
  - 구조체: 상속 불가.
  - 가장 큰 차이: 클래스는 **참조 타입 (Reference Type)**, 구조체와 열거형은 **값 타입 (Value Type)**. 값 타입은 전달될 때 복사됨.

7. 프로퍼티와 메서드

- 저장 프로퍼티 (Stored Properties): 값을 저장.
- 연산 프로퍼티 (Computed Properties): 값을 계산. `get`과 `set`을 가짐.
- 프로퍼티 옵저버 (Property Observers): `willSet`, `didSet`.
- 타입 프로퍼티 (Type Properties): `static` 키워드. 인스턴스가 아닌 타입 자체에 속함.
- 타입 메서드 (Type Methods): `static` 또는 `class` 키워드.

8. 프로토콜 (Protocols)

- 특정 작업이나 기능에 대한 청사진. `protocol SomeProtocol { ... }`
- 프로토콜 채택: `struct SomeStructure: FirstProtocol, AnotherProtocol { ... }`
- 프로토콜 합성 (Protocol Composition): `SomeProtocol & AnotherProtocol`
- 프로토콜 확장 (Protocol Extension): 프로토콜에 메서드나 연산 프로퍼티의 기본 구현을 제공.

9. 에러 처리

- Error 프로토콜: 에러 타입을 나타냄. `enum VendingMachineError: Error { ... }`
- Throwing Functions: `func canThrowErrors() throws -> String`
- 에러 처리 방법:
  1. `do-catch` 구문: `do { try expression } catch pattern { ... }`
  2. `try?`: 결과를 옵셔널로 변환. 에러 발생 시 `nil`.
  3. `try!`: 에러가 발생하지 않을 것이라고 확신할 때. 에러 발생 시 런타임 에러.
- `defer`: 현재 스코프를 나가기 직전에 실행되는 코드 블록.

10. 비동기 (Concurrency)

- `async/await`: 비동기 코드를 동기 코드처럼 순차적으로 작성하게 함.

```
func fetchWeatherHistory() async -> [Double] {
    // ...
}
let history = await fetchWeatherHistory()
```

- 비동기 시퀀스: `for await ... in`
- Task: 비동기 작업을 생성하고 관리.
- Actor: 여러 작업에서 접근할 때 데이터 경쟁을 방지하는 참조 타입. `actor TemperatureLogger { ... }`

11. 자동 참조 카운팅 (ARC)

- Swift는 ARC를 사용하여 앱의 메모리 사용을 추적하고 관리.
- 강한 참조 사이클 (Strong Reference Cycles): 두 클래스 인스턴스가 서로를 강하게 참조하여 메모리에서 해제되지 않는 문제.
- 해결책:
  - `weak`: 참조하는 인스턴스의 생명주기가 더 짧을 때 사용. 항상 옵셔널.
  - `unowned`: 참조하는 인스턴스의 생명주기가 같거나 더 길 때 사용.
  - 클로저에서의 강한 참조 사이클: 캡처 리스트 (`[weak self]`, `[unowned self]`) 사용.