Note

I // Operators & Conditions

1. Arithmetic Operators

_ Swift sử dụng các toán tử số học bình thường như cộng trừ nhân chia (+, - ,* , /) với 1 ngoại lệ là toán tử phần dư (%) mà dùng để tính toán phần dư còn lại của 2 số sau khi chia xong.

2. Operators Overloading

_ Các toán tử trong swift có thể được overload để thực hiện các chức năng khác nhau dựa vào kiểu của giá trị đang sử dụng toán tử đó.
_ Toán tử + trong swift đã được overload vài chức năng như: Nếu như mà sử dụng giữa 2 chuỗi thì sẽ gộp 2 chuỗi lại, với mảng thì sẽ gộp 2 mảng lại. Tuy nhiên Toán tử chỉ hoạt động khi mà giá trị hay bên đều cùng 1 kiểu.

3. Compound Assignment Operators

_ Toán tử này thường là 1 toán tử số học nối theo với dấu =. Toán tử này có chức năng thực hiện tính toán xong rồi gán giá trị đó cho giá trị bên trái.
_ Toán tử này thường dùng để làm cho code đẹp mắt hơn.

4. Comparison Operator

_ Toán tử so sánh thường dùng để so sánh giữa 2 giá trị với nhau. Bằng giá trị (==), khác giá trị (!=).
_ Chú ý: Toán tử này chỉ thường áp dụng cho các kiểu nào đã conform cho protocol Comparable. Nếu muốn sử dụng toán tử này với enum thì trước tiên phải conform cho enum đó.

5. Conditions

_ Lệnh điều kiện if-else trong swift có thể kéo dài thành nhiều chuỗi else-if theo ý muốn để tạo 1 control flow dễ nhìn:

```
if firstCard + secondCard == 2 {
    print("Aces")
} else if firstCard + secondCard == 21 {
    print("Black Jack")
} else {
    print("Regular Card")
}
```

_ Lệnh điều kiện if-else cũng có thể nhận giá trị boolean làm điều kiện.

6. Combining Conditions

_ Trong điều kiện If ta có thể sử dụng nhiều điều kiện hợp chồng lại với nhau sử dụng các toán tử như AND và OR (&& , | |) giúp cho thoả mãn các điều kiện phức tạp.

_ Chú ý: Ta có thể dùng ngoặc đơn () để cho các điều kiện kết hợp dễ đọc hơn.

7. Ternary Operator

_ Toán tử này hoạt động dựa theo 3 bước: so sánh điều kiện → nếu đúng thì chạy dòng code sau ký hiệu ?, → nếu sai thì chạy dòng code sau ký hiệu :.

_ Cú pháp : điều kiện ? (đúng) : (sai)

_ Toán tử này hiểu cơ bản như 1 cách rút gọn để viết if-else trong cùng 1 dòng.

8. Switch Statements

_ Mệnh đề Switch .. case thường dùng để xử lý tất cả các trường hợp có thể xảy ra cho giá trị đưa vào trong switch case đó.

_ Do Switch..case xử lý tất cả các trường hợp có thể xảy ra nên ta có hướng làm việc với nó. Liệt kê tất cả các trường hợp ra hoặc chỉ liệt kê các trường hợp cần thiết và xử lý các trường hợp còn lại bên trong case default.

_ Switch case trong swift không có thực hiện fallthrough nên nếu ta muốn đi xuống thì phải dùng lệnh fallthrough.

9. Range Operators

_ Toán tử range trong swift khác với thông thường là chỉ có 2 toán tử: đi từ a và đến z (..<) và đi từ a qua z (…) với cái đầu tiên sẽ không có z bên trong và cái còn lại sẽ kèm theo z bên trong.

_ Toán tử range có thể dùng với switch..case, với subscript để truy cập các thành phần bên trong mảng, hay tạo 1 subString với 1 chuỗi…

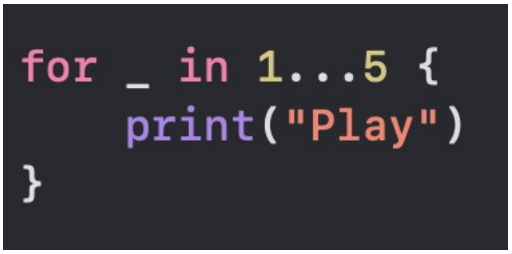
II // Loops

1. For loops

_ Dòng lệnh for … in thường thực hiện lặp lại theo số lần nhất định dựa theo 1 range hoặc 1 mảng và cho phép ta truy cập giá trị, hay thành phần của mảng trong lúc đang lặp dưới dạng 1 hằng.
Cú pháp:

```
for (hằng) in (range/mảng) {
        //Thực hiện công việc
}
```

_ Ta có thể loại bỏ tên của hằng bằng cách sử dụng _ cho tên hằng.



2. While loops

_ Dòng lệnh while … khác với lệnh for ở thay vì lặp số lần biết trước thì lệnh while chỉ chạy cho đến khi nào mà điều kiện không thoả mãn.
Cú pháp :

```
while (điều kiện) {
        // Thực hiện công việc
}
```

3. Repeating loops

_ Lệnh repeat khác với lệnh while ở việc khi nào mà ta chạy while thì nó sẽ kiểm tra điều kiện trước rồi mới chạy code. Còn với repeat công việc kiểm tra điều kiện được thực hiện sau cùng nên việc chạy code sẽ diễn ra ít nhất 1 lần.
Cú pháp:

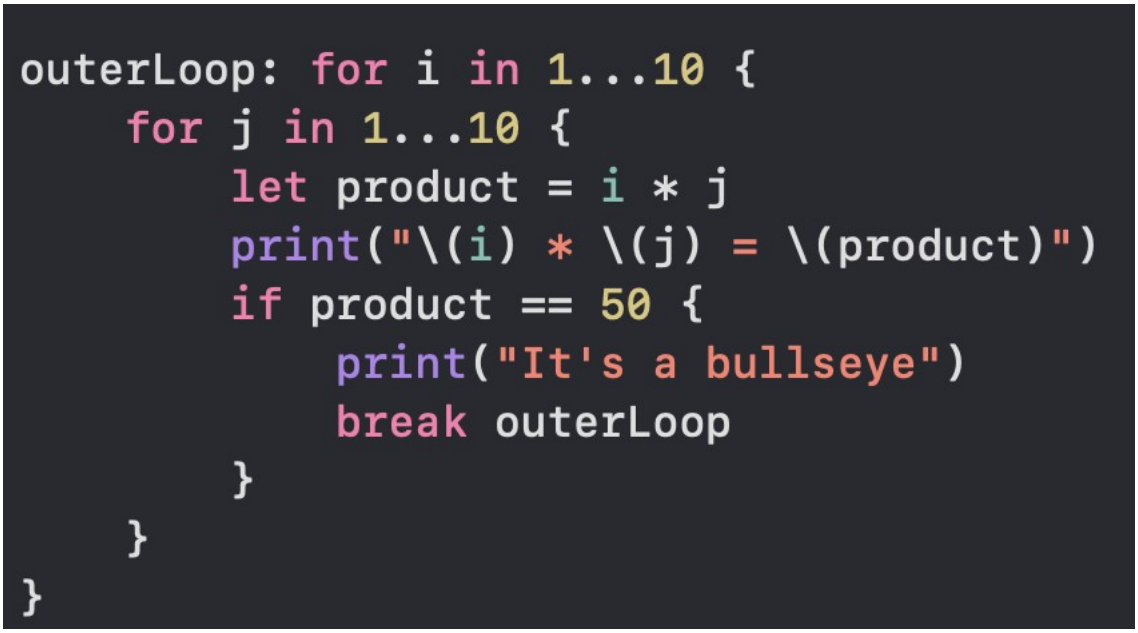
```
repeat {
        // Code
} while (điều kiện)
```

4. Exiting loops

_ Để thoát ra khỏi 1 vòng lặp ta dùng lệnh break.
_ Chú ý: Lệnh break chỉ hoạt động để đi ra khỏi vòng lặp hiện tại. Lệnh break chỉ hoạt động đi ra khỏi 1 vòng lặp

5. Exiting multiple loops

_ Để đi ra khỏi nhiều vòng lặp ta phải đi ra khỏi vòng lặp ngoài cùng nhất. Ta thực hiện bằng cách đặt 1 label cho vòng lặp ngoài cùng nào mà ta muốn đi ra. Thực hiện lệnh break ra khỏi vòng lặp đó.
Ví dụ:



```swift
outerLoop: for i in 1...10 {
    for j in 1...10 {
        let product = i * j
        print("\(i) * \(j) = \(product)")
        if product == 50 {
            print("It's a bullseye")
            break outerLoop
        }
    }
}
```

6. Skipping items

_ Để bỏ qua bước lặp hiện tại ta dùng lệnh continue. Thay vì ta đi ra khỏi vòng lặp. như lệnh break thì ta vẫn ở trong vòng lặp nhưng ở bước kế tiếp. Ta cũng có thể thực hiện continue ở vòng lặp nào ta muốn sử dụng bước như lệnh break

7. Infinite loops

_ Vòng lặp vô tận thường được sử dụng với lệnh while. Các vòng lặp này thường chạy mà không kết thúc hoặc chạy khi nào mà ta sẵn sàng.

III // Functions

1. Writing functions

_Viết phương thức trong swift thường dùng để thực hiện một công việc mà ta muốn làm nhiều lần mà không lặp lại code.

2. Accepting parameters

_Ta cũng có thể thêm các tham số để đưa dữ liệu vào bên trong 1 phương thức. Thường thì ta để ít tham số do nếu để nhiều tham số thì sẽ dẫn đến khó đọc code.

_Có 3 hướng giải quyết nếu 1 phương thức có nhiều hơn 5~6 tham số:

+ Xem các tham số đó có cần thiết hay không?
+ Có thể làm nhỏ phương thức đó thành nhiêù phương thức với 1 hay nhiều tham số khác nhau hay không?
+ Có thể gộp các tham số đó lại thành 1 giá trị hay không?

3. Returning values

_ Sau khi mà ta đưa dữ liệu vào trong 1 phương thức thì ta cũng có thể đưa dữ liệu ra. Trong swift ta có thể loại bỏ lệnh return nếu mà phương thức chỉ có 1 dòng lệnh.

_Các trường hợp không nhận:

+câu lệnh như break, continue, ...
+vòng lặp
+điều kiện như if-else , switch case, …
+khai báo 1 giá trị mới.

_Điều kiện ngoại lệ: Ternary operator

_ Ta có thể trả về nhiều giá trị mà cùng 1 kiểu như sử dụng 1 container kiểu mảng. Còn với các giá trị mà khác kiểu thì tốt nhất dùng tuples do:

+ Kiểu tuples có giá trị và thứ tự định sẵn nên luôn đảm bảo an toàn khi gọi tới gía trị tuples.
+ Do ta sử dụng kiểu tuples nên ta không bao giờ bị lỗi khi mà gọi key của giá trị trong tuples (Đó là lí do mà không chọn Dictionary do ta phải biết trước key của từng phần tử để truy cập).
+ Kiểu tuple khi truy cập là không phải Optional làm cho code đơn giản hơn.

4. Parameter labels

_ Ta cũng có thể đặt tên bên ngoài khi mà gọi phương thức. Tên này giúp cho ta có thể sử dụng tên của tham số bên trong phương thức và chỉ hiện tên đó mỗi khi gọi phương thức. Cách này thường dùng để cho lệnh gọi phương thức có ý nghĩa hơn.

5. Omitting parameters labels

_ Ta có thể làm ẩn tên tham số bằng cách đặt tên bên ngoài là _ . Cách này thường dùng chỉ cho phương thức mà có tên là động từ giúp cho phương thức dễ nhìn hơn.

6. Default parameters

_ Ta cũng có thể cho tham số có giá trị mặc định để cho khi gọi thì có thể không cần gọi đến tham số đó.
- Chức năng này dùng để cho các phương thức có thêm tính linh động khi mà cần xử lý các công việc mà đôi lúc mới cần đổi giá trị của tham số.

7. Variadic functions

_ Chức năng này dùng để thêm cho các phương thức chức năng tạo 1 tham số là mảng mà chứa tất cả các tham số lúc đang gọi. Chức năng này thường dùng để cho các phương thức xử lý nhiều giá trị nhập vào cùng 1 lúc.

8. Writing and running throwing functions

_ Đôi lúc khi chạy phương thức ta muốn xử lý các trường hợp mà bị lỗi. Với phương thức ném lỗi ta có thể làm cho các phương thức ném lỗi nếu xảy ra lỗi.

Cú pháp :

```
func Name () throwing → {
        // Code
}
```

_ Chú ý:

+ Lệnh throw dùng để ném lỗi và ra khỏi phương thức.
+ Để gọi 1 phương thức mà ném lỗi ta phải dùng try và thực hiện xử lý lỗi với do..catch.
+ Phương thức ném lỗi có thể gọi phương thức ném lỗi khác miễn là ta có gọi phương thức đầu tiên trong do catch.

Gọi phương thức:

```
do {
        try func()
} catch {
        // Error Handling
}
```

9. Inout parameter

_ Mệnh đề inout là lệnh dùng để báo hiệu 1 tham số đầu vào có thể bị biến đổi trong lúc thực hiện phương thức.

_ Điều kiện:

+ Tham số đầu vaò phải là biến.
+ Phải đặt inout trước kiểu của tham số
+ Phải đặt dấu & ở trước tên tham số đầu vào.
+ Không cho phép đặt 2 tham số hay nhiều hơn là cùng 1 biến.