**Bài 4 – Toán tử trong Swift**

**1. Thuật ngữ – Terminology**

Có ba loại toán tử:

* Toán từ một ngôi (*unary*) : Toán tử một ngôi là toán tử hoạt động dựa trên một phần tử duy nhất (VD: ++a) .Các tiền tố (prefix) xuất hiện ngay trước biến và các hậu tố (postfix) xuất hiện ngay sau biến (VD: a–)
* Toán tử hai ngôi(*binary*) : Hoạt động dựa trên hai phần tử ( VD : 1 + 1 ) và được gọi là trung tố (infix ) vì chúng xuất hiện ở giữa hai biến.
* Toán tử ba ngồi (*ternary*) : Hoạt động dựa trên ba biến. Giống như C , Swift chỉ có một toán tử ba ngôi duy nhất ( a ? b : c).

**2. Toán tử gán – Assignment Operator**

Toán tử gán ( a = b ) khởi tạo hay cập nhật giá trị của a với b.

|  |  |
| --- | --- |
| 1  2  3  4 | var b = 10  var a = 20  a = b  //Giá trị của a lúc này = 10 |

Nếu bên phải phép gán là một tuple với nhiều giá trị , thì bên trái chúng ta cũng có thể tách thành nhiều biến constant hoặc variable:

|  |  |
| --- | --- |
| 1 | let (x,y,z) = (10,20,30) |

Không giống như trong C hoặc Objective C , phép gán trong Swift không trả về một giá trị. Xem đoạn code sau để hiểu rõ hơn:

|  |  |
| --- | --- |
| 1  2  3 | if x = y {      // Nếu code như này sẽ bị lỗi vì phép gán x = y không trả về giá trị nào .  } |

**3. Toán tử toán học – Arithmetic Operators**

Swift hỗ trợ bốn toán tử toán học tiêu chuẩn cho tất cả các kiểu số:

* Phép cộng – Addition ( + )
* Phép trừ – Subtraction ( – )
* Phép nhân – Multiplication ( \* )
* Phép chia – Division ( / )

Ví dụ về 4 toán tử toán học :

|  |  |
| --- | --- |
| 1  2  3  4 | 1 + 1  // = 2  2 - 2  // = 0  2 \* 2  // = 4  2 / 2  // = 1 |

Swift không cho phép các giá trị của biến bị tràn như mặc định (  ).

|  |  |
| --- | --- |
| 1  2  3 | var maxInt = Int16.max  //Sẽ nhận được thông báo lỗi khi ta cộng giá trị minInt lên 1 vì Swift không cho phép điều này  maxInt += 1 |

Nếu bạn muốn tràn giá trị mặc định trong một vài trường hợp thì sử dụng toán tử tràn ( & ).

|  |  |
| --- | --- |
| 1  2 | var maxInt = Int16.max  maxInt &+ 1 |

Như đoạn code trên thì ta sẽ cộng thêm giá trị maxInt thêm 1 và sử dụng toán tử tràn ( & ) .  
Ngoài ra thì toán tử cộng ( + ) cũng được sử dụng để cộng chuỗi trong Swift.

|  |  |
| --- | --- |
| 1  2 | var world =  "hello, " + "world"  // Cộng chuỗi "hello," và chuỗi "world" lại . |

**4. Toán tử số dư – Remainder Operator**

Toán tử này khá đơn giản, đây là toán tử lấy số dư của phép chia.

|  |  |
| --- | --- |
| 1  2  3 | var remainder = 7 % 3  //Output  remainder = 1 |

**5.Toán tử gán phức hợp – Compound Assignment Operators**

Toán tử gán phức hợp trong Swift được dùng khi sử dụng cặp toán tử ( += ) và  ( -= ) .

|  |  |
| --- | --- |
| 1  2  3  4 | var a = 10  a += 10 // Giống như khi ta viết a = a + 10  //Output  a = 20 |

Chú ý là toán tử gán phức hợp không trả về giá trị nên ta không thể gán giá trị vào biến :

|  |  |
| --- | --- |
| 1  2  3 | var a = 10  var b = a += 10  //Trình biên dịch sẽ bắt lỗi khi a += 10 không trả về giá trị để gán vào biến b |

**6. Toán tử so sánh – Comparison Operators**

Swift hỗ trợ tất cả chuẩn so sánh :

* Bằng (Equal) a == b
* Không bằng ( Not equal ) a != b
* Lớn hơn ( Greater than ) a > b
* Bé hơn (Less than) a < b
* Lớn hơn hoặc bằng ( Greater than or equal to ) a >= b
* Bé hơn hoặc bằng ( Less than or equal to ) a <= b
* Toán tử === hoặc !== để kiểm tra references của hai Object.

Ví dụ để hiểu rõ hơn.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | 1 == 1   // true  2 != 1   // true  2 > 1    // true  1 < 2    // true  1 >= 1   // true  2 <= 1   // false bởi vì 2 > 1 |

Như bài trước chúng ta cũng đã học về Tuple , vậy với dữ liệu như tuple thì Swift so sánh như nào .

|  |  |
| --- | --- |
| 1  2  3 | (1, "zebra") < (2, "apple")   // true because 1 is less than 2; "zebra" and "apple" are not compared  (3, "apple") < (3, "bird")    // true because 3 is equal to 3, and "apple" is less than "bird"  (4, "dog") == (4, "dog")      // true because 4 is equal to 4, and "dog" is equal to "dog |

Touple cũng sử dụng toán tử so sánh để so sánh . Tuple so sánh lần lượt từ trái qua phải , tại mỗi thời điểm tuple chỉ so sánh một giá trị. Với giá trị String Swift sẽ so sánh lần lượt các characters trong String (Ký tự nào xuất hiện trước trên bảng chữ cái sẽ bé hơn ký tự xuất hiện sau ‘a’ < ‘b’).

**6. Toán tử điều kiện ba ngôi – Ternary Conditional Operator**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | var check = true  var a = 0  if check {     a = 10  }else {     a = 20  } |

Đoạn xử lý trên khá đơn giản mà lại tốn khá nhiều dòng code cho nó , toán tử điều kiện ba ngôi sẽ giúp chúng ta giải quyết xử lý trên khá nhanh gọn.

|  |  |
| --- | --- |
| 1  2  3  4  5 | var check = true  var a = check ? 10 : 20  //check : điều kiện so sánh ( chỉ nhận giá trị true hoặc false )  // 10   : Nếu điều kiện check là đúng thì sẽ nhận giá trị này  // 20   : Nếu điều kiện check là sai thì nhận giá trị này |

**7. Toán tử hợp nhất Nil – Nil-Coalescing Operator**

Toán tử Nil-Coalescing Operator dùng để unwraps một optional ( a ?? b ) .Nếu optional a có chứa một giá trị thì sẽ unwrap optiona biến a hoặc sẽ trả về giá trị biến b nếu biến a là nil . Kiểu dữ liệu của biến b phải trùng với kiểu dữ liệu bên biến a.

Xem ví dụ unwrap optional biến a sử dụng toán tử điều kiện ba ngôi.

|  |  |
| --- | --- |
| 1 | a != nil ? a! : b |

Và ví dụ unwrap optional biến a sử dụng tóa tử Nil-Coalescing Operator.

|  |  |
| --- | --- |
| 1 | var c = a ?? b |

Nếu biến a không nil thì biến a sẽ được unwrap và gán giá trị vào biến c ngược lại biến c  sẽ bằng giá trị biến b.

|  |  |
| --- | --- |
| 1  2  3  4  5 | let defaultColorName = "red"  var userDefinedColorName: String?   // defaults to nil    var colorNameToUse = userDefinedColorName ?? defaultColorName  // userDefinedColorName là nil, vì vậy colorNameToUse sẽ được set giá trị là "red" |

**8. Toán tử phạm vi – Range Operators**

Swift gồm có hai toán tử phạm vi :

***8.1 Toán tử phạm vi khép kín – Closed Range Operator***

Cùng xem ví dụ này và phân tích nó xem nhé.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | for i in 0...5 {      print("i = \(i)")  }  //Output  i = 0  i = 1  i = 2  i = 3  i = 4  i = 5 |

Ta thấy vòng lặp chạy lần lượt từ 0 đến 5 vậy ta có thể kết luận toán tử khép kín chạy từ giá trị a đến giá trị b (a…b). Xem tiếp ví dụ nữa .

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | for i in 6...5 {      print("i = \(i)")  }  //Output  Error : Execution was interrupted.  Vì giá trị a = 6 > b = 5 |

=> Giá trị a không thể lớn hơn giá trị b .

**8.2 Toán tử bán phạm vi – Half-Open Range Operator**

Tiếp tục xem ví dụ để hiểu về toán tử bán phám vị.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | for i in 0..<5 {      print("i = \(i)")  }  //Output  i = 0  i = 1  i = 2  i = 3  i = 4 |

Ta thấy vòng lặp chạy lần lượt từ 0 đến 4 vậy ta có thể kết luận toán tử bán phạm vị chạy từ giá trị a đến giá trị b – 1 (a..<b) => *phạm vi chạy của toán tử bán phạm vi bao gồm giá trị a nhưng không bao gồm giá trị b.*Cũng giống như toán tử Closed range operator , giá trị a cũng không được lớn hơn giá trị b trong toán tử Half-Open range operator .

**9- Toán tử logic – Logical Operators**

Toán tử logic -Logical operators – chỉnh sửa hoặc kết hợp các giá trị logic Boolean *true* và *false*. Swift hỗ trợ ba chuẩn toán tử logic được tìm thấy dựa trên ngôn ngữ C:

* Logical NOT (!a)
* Logical AND (a && b)
* Logical OR (a || b)

***9.1 Toán tử logic NOT – Logical NOT Operator***

Toán tử logic NOT có nhiệm vụ đảo một giá trị true thành false và ngược lại false thành true.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var checkA = false  if !checkA {      print("Success")  }  //Output  Success |

Như ví dụ trên thì giá trị biến checkA đã được biến đổi thành true sau khi sử dụng toán tử logic NOT.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | var checkA = true  if !checkA {      print("Success")  }else {      print("Hello")  }  //Output  Hello |

Còn ví dụ này thì ngược lại biến checkA từ true đã đổi sang false sau khi sử dụng toán tử logic NOT.

***9.2 Toán tử logic AND – Logical AND Operator***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | var checkName = true  var checkAge = false  if (checkName && checkAge) {      print("Check ok")  } else {      print("Check error")  }  //Output  Check error |

Toán tử logic AND *(a && b)* tạo ra các biểu thức logic mà cả hai giá trị phải *true* .  
Nếu một trong hai giá trị là *false*, kết quả cộng chung cũng sẽ là *false*. Trong thực tế, nếu giá trị đầu tiên là false, giá trị thứ hai thậm chí sẽ không được đánh giá, bởi vì nó không thể làm thay đổi sự kết quả cộng chung tương đương với *true*. Điều này được gọi là đánh giá ngắn mạch – short-circuit evaluation.

**9.3 Toán tử logic OR – Logical OR Operator**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | var checkName = true  var checkAge = false  if (checkName || checkAge) {      print("Check ok")  } else {      print("Check error")  }  //Output  Check ok |

Toán tử logic OR (a || b) là một toán tử được viết bởi hai dấu gạch liền kề. Bạn sử dụng nó để tạo ra các biểu thức logic trong đó chỉ có một trong hai giá trị là true thì những kết quả cộng chung là true.

Giống như toán tử logic AND, toán tử logic OR sử dụng đánh giá ngắn mạch để xem xét biểu thức của nó. Nếu phía bên trái của một biểu thức logic OR là *true*, bên phải không được đánh giá, bởi vì nó không thể thay đổi kết quả của biểu thức tổng thể.

**9.4 Kết hợp toán tử logic – Combining Logical Operators**

Bạn có thể kết hợp nhiều toán tử logic để tạo ra một biểu thức.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | var checkName = true  var checkAge = true  var checkStatus = false  if (checkName && checkAge) || checkStatus  {      print("Check ok")  }else{      print("Check error")  }    //Output  Check ok |

Như ví dụ trên ta kết hợp hai toán tử AND và OR .

**9.5 Dấu ngoặc đơn – Explicit Parentheses**

Cùng xem lại ví dụ trên.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | var checkName = true  var checkAge = true  var checkStatus = false  if (checkName && checkAge) || checkStatus  {      print("Check ok")  }else{      print("Check error")  }    //Output  Check ok |

Trong ví dụ này chúng ta sử dụng dấu ngoặc đơn ( )  để nhóm hai giá trị checkName và checkAge để được coi như một điều kiện riêng biệt tức là nếu giá trị checkName và checkAge cùng true thì sẽ không cần phải check giá trị biến checkStatus nữa.