

Swift Programming

Lecture 4

Control Statements

Tatsuo Minohara



文とブロック

- 文
 - 式
- ブロック
 - 複数の文をまとめる
 - Swiftでは、`{ }`で囲まれるとブロックと見做される



Swiftの文の実行

- 上から順番に実行される
 - `print( 1 )`
`print( 2 )`
`print( 3 )`
- 1行の中で2つ以上の命令を記述した場合は、;(セミコロン)で区切り、左から右に実行される
 - `print( 1 ); print( 2 ); print( 3 );`



文とブロック

- ブロックは、複数の文をまとめるもの
- Swiftでは、波括弧{ } —英語ではBraceと呼ばれる— で文をブロックとしてまとめることができる
- 左側に空白が空いている量が文のアウトラインのレベルを示すが、ブロックに所属する文は1つレベルが下がる（右側にずれる）
- インデントはXcodeでは自動的に行なわれる（BBEditでは行なわれない）が、自分で調整したい場合はTABキーを使う、BackSpace/Deleteで戻せる

制御構文(Control Statement)

- 通常の逐次実行の他に、繰返し、条件分岐がある



Normal



Repetative



Conditional



3つのif文

- 書式1：条件を満足しなければスキップする
 - `if 条件式 { 満足するときにすること }`
- 書式2：条件を満足する場合としない場合
 - `if 条件式 { 満足するとき } else { しないとき }`
- 書式3：上から条件を満たすものを当てはめていく
 - `if 条件式 { 満足するとき } else if 次の条件 { } ...`



条件式の書き方

- 条件式は、基本的に論理値に評価される式
- 比較演算子を使った条件式
 - 書式： 式 比較演算子 式
 - 演算子の左右の式はどちらに書いても構わない
- 比較演算子は6つある（空白不可）
 - ==, !=, >, <, <=, >=
 - >は大きい、<は小さい
 - <=と>=は、不等号を先に書く（2文字続けて記述）
 - 等しいは、==（=が2つ、2文字続けて記述）
 - 等しくないは、!= （2文字続けて記述）



タプル・文字列と比較演算子

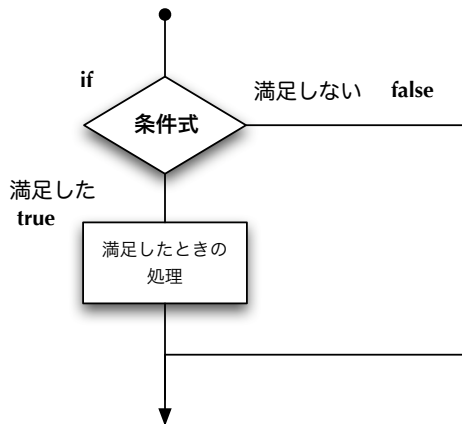
- タプル同士の比較では、比較演算子を使うときは、要素の個数を合わせる必要がある
 - ➔ 例： `(34, 45) < (34, 45, 89)` // エラーになってしまう
- タプルで、先頭の要素から大小の比較が行なわれる
 - ➔ 例： `(34, 45, 56) < (34, 45, 57)` // trueに評価
- 文字列で、比較演算子を使うと文字列の長さには関係なく、すべてアルファベット順（Unicode順）の評価になる
 - ➔ 例： `"あいう" < "かきく"` // trueに評価
 - `"ABC" < "AE"` // trueに評価

if文

if 条件式 {

条件式が満足されたときに実行されること

}



if - else 文

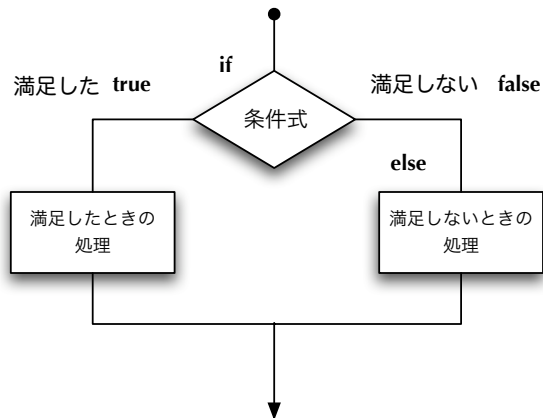
if 条件式 {

条件式が満足されたときに実行されること

} **else** {

満足されないときに実行されること

}



if - else if - else 文

if 条件式1 {

条件式1が満足されたときに
実行されること

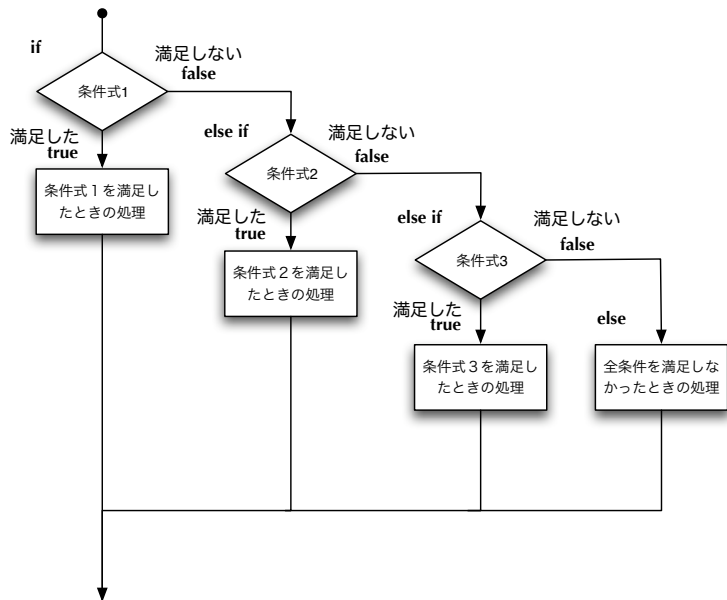
} else if 条件式2 {

条件式2が満足されたときに
実行されること

} else {

全てが満足されないときに
実行されること

}





よくある文法エラー

- 反対の条件を書いてしまう
- 条件や**else**の後に; (セミコロン) を入れる
 - **if**文や**else**文と関係ない独立ブロックに
- **else if** の**else**を忘れる→別の2つの **if** 文に
- 複合条件を, (カンマ) で記述する
- 2文字の演算子の間に空白を入れる
 - 例 : `= =` `> =` `< =`
- `==`を`=`で書く

オプションナル束縛のついたif文

- オプションナル型の定数への代入とif文を組み合わせたことができる
 - 書式：**if let** 定数 = オプションナル型の式 { }
else { }
- 定数宣言された変数に代入されて、ブロックが実行される
- オプションナル型の式の評価結果がnilの場合は、**else**のブロックが実行される（**else**ブロックは省略可能）
 - 例：**var s : String = "1234"**
if let num = Int(s) { print(num) } // 変換できた場合
else { print("\s) は整数に変換できません") }



式の型の判定

- 式の型を判定するis演算子がある
 - 書式： 式 **is** 型名
 - 例： x **is** Int
y **is** Double
- 式が、その型になっていると**true**、そうでないと**false**が返される
- Intは、Int64, UInt, Int32などと互換ではないので注意
 - 例： 100 **is** UInt // falseとなる
(100 **as** UInt) **is** UInt // trueとなる
- 何でも良いという型名としてAnyがある
 - 例： 100 **is** Any // trueとなる



if文のネスト

- 外側のif文
 - 大きく分けたいときにつかう
- 内側のif文
 - その条件のなかで更に細かく分けたいときにつかう



例題：大の月・小の月

- 小の月...月の日数が31日ないもの
- 大の月...月の日数が31日あるもの
- 西向く侍、小の月 2, 4, 6, 9, 11(=土)
- 偶数の月と奇数の月で分かれる
 - 偶数：8よりも小さい月が小の月
 - 奇数：8よりも大きい月が小の月
- if文のネストで表現してみる
- ユーザから月を入力してもらい、その月が大の月か小の月かを判定する



論理式

- 条件式を複数使うことができる
 - 論理積 (AND) `&&` 両方の条件を満たす
 - 論理和 (OR) `||` どちらかの条件を満たす
 - 否定 (NOT) `!( )` 反対の条件にする
- 論理式
 - 条件式
 - `!(論理式)`
 - 論理式 `&&` 論理式
 - 論理式 `||` 論理式

論理式と論理値

- 論理演算子の評価

! true	true && true	true true
! false	true && false	true false
	false && true	false true
	false && false	false false



論理式の記述の仕方

- かならず論理積・論理和などで結ぶ必要がある
 - $100 < x < 200$ はだめ (Python3では使える)
 - $100 < x \ \&\& \ x < 200$
- 論理式の記号は、空白をあけない (2文字記号)
 - $100 < x \ \& \ \& \ x < 200$ はだめ
 - $100 == x \ || \ x == 200$ もだめ

ド モルガンの法則

- 否定のついた論理式を変換することが可能

$$\neg(\text{条件A} \ \&\& \ \text{条件B}) \Leftrightarrow \neg\text{条件A} \ || \ \neg\text{条件B}$$

$$\neg(\text{条件A} \ || \ \text{条件B}) \Leftrightarrow \neg\text{条件A} \ \&\& \ \neg\text{条件B}$$

- 適用例

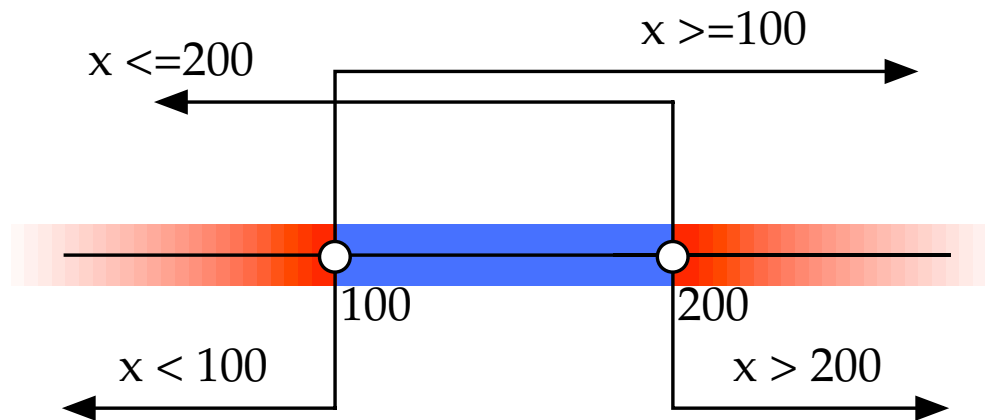
$$\neg(x == 1 \ || \ x == 2)$$

$$\Rightarrow \neg(x == 1) \ \&\& \ \neg(x == 2)$$

$$\Rightarrow x \neq 1 \ \&\& \ x \neq 2$$

ド・モルガン則と数直線

- $100 \leq x \ \&\& \ x \leq 200$
- $\neg(100 \leq x \ \&\& \ x \leq 200)$
 - ▶ $\neg(100 \leq x) \ || \ \neg(x \leq 200)$
 - ▶ $100 > x \ || \ x > 200$





それ以外の変換

- 否定と等号

- $!(a == b) \Leftrightarrow a != b$

- $!(a != b) \Leftrightarrow a == b$

- 不等号と否定

- $!(a \geq b) \Leftrightarrow a < b$

- $!(a > b) \Leftrightarrow a \leq b$

- 等号付き不等号の分解

- $a \geq b \Leftrightarrow a > b \parallel a == b$



if式

- ifで、評価する値をどちらかに決定できる

(条件式や論理式) ? 真の場合の値 : 偽の場合の値

- 式なので、代入文の中やメソッド呼出しのパラメータの中でも使える
 - ▶ `x = ( y >= 100 ) ? 10 : 20`
 - ▶ `print( ( x > 100 ) ? "Red" : "Blue" )`



if文とif式

- $x = (y \geq 100) ? 10 : 20$
- 等価なif文は次のようになる

```
if y >= 100 {  
    x = 10  
} else {  
    x = 20  
}
```



if式（続き）

- if式をネストさせることもできる

```
c = ( y >= 80 ) ? "A" : ( y >= 60 ) ? "B" :  
    ( y >= 40 ) ? "C" : "D" ;
```

- かなり多用するプログラマが多い
 - if文が省略できる、短く書ける
 - 使い過ぎると何をしているのかわからないので、ほどほどに



ExcelのIF関数

- =IF(条件式, 真の時の値, 偽の時の値)
- =AND(条件式, 条件式)
- =OR(条件式, 条件式)
- =NOT(条件式)
- =AVERAGEIF(範囲, 検索条件式か検索値)
- =COUNTIF(範囲, 検索条件式か検索値)
- =SUMIF(範囲, 検索条件式か検索値)
- 条件式は、 >=式 <=式 =式 <>式 >式 <式 あるいは式



switch文

- if ～ else if文で、定数と比較するときを使う
- 基本的な書式

```
switch ( 式 ){  
    case 定数1: このときの処理  
    case 定数2: このときの処理  
  
    :  
    default: すべてに該当しないときの処理  
}
```



switch文の例

```
let someCharacter: Character = "z"  
switch someCharacter {  
  case "a":  
    print("The first letter of the alphabet")  
  case "z":  
    print("The last letter of the alphabet")  
  default:  
    print("Some other character")  
}
```

switch文、複数の候補

- 例 :

```
let someCharacter: Character = "e"
```

```
switch someCharacter {
```

```
  case "a", "e", "i", "o", "u":
```

```
    print("\ (someCharacter) is a vowel")
```

```
  case "b", "c", "d", "f", "g", "h", "j", "k", "l", "m",
```

```
    "n", "p", "q", "r", "s", "t", "v", "w", "x", "y", "z":
```

```
    print("\ (someCharacter) is a consonant")
```

```
  default:
```

```
    print("\ (someCharacter) is not a vowel or a consonant")
```

```
}
```



タプルを伴うswitch文(1)

- `_` ... 受け取って捨てることを意味する

```
let somePoint = (1, 1)
switch somePoint {
case (0, 0):
  print("\ (somePoint) is at the origin")
case (_, 0):
  print("\ (somePoint) is on the x-axis")
case (0, _):
  print("\ (somePoint) is on the y-axis")
default:
  print("\ (somePoint) is not on any axis")
}
```




タプルを伴うswitch文(2)

- letで変数に代入することが可能

```
let anotherPoint = (2, 0)
switch anotherPoint {
  case (let x, 0):
    print("on the x-axis with an x value of \(x)")
  case (0, let y):
    print("on the y-axis with a y value of \(y)")
  case let (x, y):
    print("somewhere else at (\(x), \(y))")
}
```

タプルを伴うswitch文(3)

- whereで条件指定することが可能

```
let yetAnotherPoint = (1, -1)
switch yetAnotherPoint {
  case let (x, y) where x == y:
    print("(x, y) is on the line x == y")
  case let (x, y) where x == -y:
    print("(x, y) is on the line x == -y")
  case let (x, y):
    print("(x, y) is just some arbitrary point")
}
```

範囲指定を伴うswitch文

- 範囲指定を使うことが可能
 - `a...b` `a`から`b`まで（両端含む） `a..<b` `a`から`b`未満（上限含まない）

```
let approximateCount = 62
let naturalCount: String
switch approximateCount {
case 0: naturalCount = "no"
case 1..<5: naturalCount = "a few"
case 5..<12: naturalCount = "several"
case 12..<100: naturalCount = "dozens of"
case 100..<1000: naturalCount = "hundreds of"
default: naturalCount = "many" }
```



switch文で次のcaseも行ないたい場合

- fallthrough文を使う
- 使用例：
 - 100個の座標についての直線上、二次曲線上にあるかどうか判定

```
for n in 1...10 {  
  for m in 1...10 {  
    switch (n, m) {  
      case let (x, y) where x == y:  
        print( "\ (x), \ (y) in on the line x == y" )  
        fallthrough  
      case let (x, y) where x * x == y: print( "\ (x), \ (y) in on the line x**2 == y" )  
      case (_, _): print( terminator: "" ) } } }
```



guard文（飛ばします）

- オプショナル型を使うときに使う
 - 書式：**guard** オプショナル型束縛 **else** { ブロック }
- 関数定義の中でしか使えない。
- 例：
 - **guard** let line = readLine() **else** {
 print("入力がありません。")
 return
}

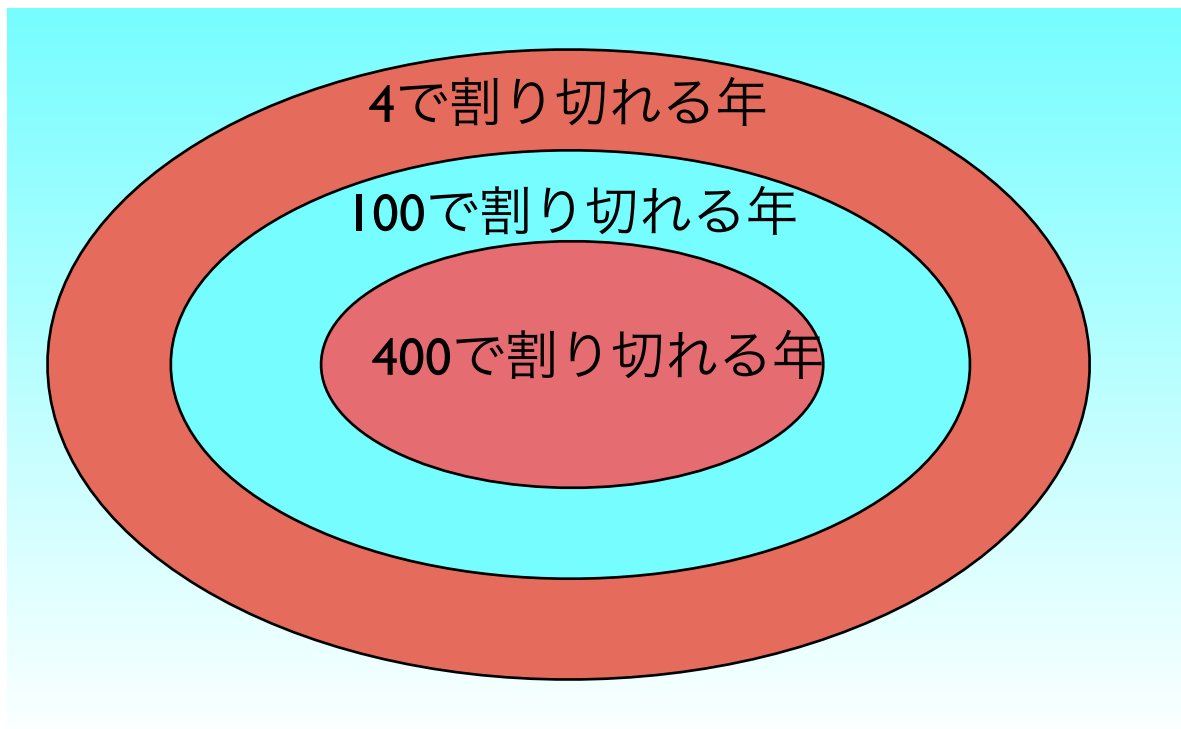


今日の授業時課題

- ユーザに西暦を入力してもらう
- その年が閏年(Leap Year)かどうか判定するプログラム
 - 4で割り切れない年は、平年 例：2019
 - 4で割り切れる年は、閏年 例：2008
 - 100で割り切れる年は、平年 例：1900
 - 400で割り切れる年は、閏年 例：2000

閏年の求め方

- 集合の図で描くと下記のようなになる





閏年の求め方

- 3つの考え方がある
 - ▶ Rare（起こりにくい）もの（内側）から記述する
 - ❖ if else if 文でできる
 - ▶ 起こりやすいもの（外側）から記述する
 - ❖ if文のネスト（多重化）を用いる
 - ▶ 集合の該当する部分だけを示す
 - ❖ 論理式を用いる



閏年とは？

- 地球の公転周期は、365日ぴったりではない。
- 4年に1日を挿入
 - $1/4 \dots 0.25$ 日
- 100年1日は省く
 - $1/100 \dots 0.01$ 日
- 400年1日を挿入
 - $1/400 \dots 0.0025$ 日
- $365\text{日} + 0.25 - 0.01 + 0.0025 = 365.2425\text{日}$
- 本来は、 365.2422日
- 古代マヤ文明は、この値に近づけるため、閏日を設定
- 陰暦は、月と年があわなくなるので、閏月を挿入