オプショナルバインディング

nillかもしれないオプショナルバインディング

変数には値が入っていると思いがちだが、実際には何も入ってないことがある。

```
import UIKit
//空っぽの配列
var nums:[Int] = []
//numsの最後の要素をlastNumに代入する。
let lastNum = nums.last
//lastNumはnilなのでエラーになる
let ans = lastNum * 2
```

lastは配列から最後の要素を参照するプロパティ。ここではnums.lastでnumsから最後の要素を取り出して変数lastNumに代入している。ところが、numsが空っぽなので参照の対象がない。するとnums.last
はエラーを返すのではなく、「値なし」を示すnilを返す。つまりlastNumにはnilが代入され、続く式のlastNum*2が計算できずにエラーになる。
このように、変数にnilが入っているとエラーの原因となるので、Swiftでは通常の変数にはnilを代入できなくしている。次のように変数numにnilを代入しようとするとエラーになる。

```
var num:Int
//代入できる
num = 5
//nilは代入できずにエラーになる。
num = nil
```

変数にnilを代入することが絶対できないと言うのかとそうではない。変数を宣言する際にnum:Int?のように型の後ろに?を付けると普通のInt型ではなくOptional型になり、numはInt型の整数だけでなくnilも代入できる変数になる。

```
var num:Int?
//代入できる
num = 5
//代入できる
num = nil
print(num)
```

optional()で値をラップする

次にnumに整数5の値を代入して出力する。するとそのままの整数の5ではなく、Optional(5)と出力される。この状態はOptional(Int)のように型がラップされている状態と言える

```
var num:Int?

//代入できる
num = nil
//代入できる
num = 5
print(num)
```

結果

```
Optional(5)
```

このように、numがOptional型になった事から、numの値はnilかもしれない注意すべき値になった。このようなOptional型の値、すなわち「nilかもしれない値」をオプショナルバリューと呼ぶ。

オプショナルバリューはそのままでは使えない

ここで、最初の例の配列numが[1,2,3]だとどうなるかを確認してみる。今度はnumsに要素が入っているので、lastNumにはnilではなく最後の要素の3が代入されて変数ansにはlastNum*2の計算結果の6が代入されるはず。しかし、今回もlast*2でエラーになる。

```
var nums:[Int] = [1,2,3]
//lastNumには3が代入されるはずだがされない。
let lastNum = nums.last
let ans = lastNum * 2
```

その原因は、lastNumの値を出力して確かめるとわかるようにlastNumには3でなく、Optional(3)が入っている。nums.lastが返す値はnilかもしれない値、すなわちオブジェクトバリューであることから、Optional(3)このようにオプショナルバリューはそのまま使うことはできない。。

オプショナルバリューを強制アンラップして使う。

Optional(3)を使うにはOptional()を取り去って3に戻さなければならない。この操作をアンラップと言う。アンラップする方法にはいくつかあり、その一つがオプショナルバリューに!を付ける方法。
次のコードはオプショナルバリューが入っている変数numをnum!のようにアンラップして5の整数に戻している。

```
import UIKit
var num:Int?
num = 5
print(num)
print(num!)//強制アンラップする
```

結果

```
Optional(5)
5
```

lastnumのコードではnums.last!のように強制アンラップした値をlastNumに代入すればlastNumにはオプショナルバリューにならないためlastNum*2がエラーにならず計算する事ができる。

```
import UIKit
var nums:[Int] = [1,2,3]

//強制アンラップする。
let lastNum = nums.last!
let ans = lastNum * 2
print(lastNum)
```

結果

```
6
```

しかし、一番最初に示したようにnumsが空だった場合はnums.last!はnilであって強制アンラップしても結果はnilになり、lastNum!*2がエラーになる。強制アンラップは、値がnilだった場合に対応できる確実なやり方がいくつかある。

nilだった場合は0を使う

オプショナルバリューがnilだった場合は代わりとなる値を指定しておく必要がある。それが??演算子。
上記の例で配列が空だった場合はlastNumに0を代入するようにしたのが以下のコードである。

```
iamport UIKit
let nums:[Int] = []
let lastNum = nums.last ?? 0
let ans = lastNum * 2
print(ans)
```

結果

```
0
```