5-9 guard let-elseを使ったオプショナルバインディング

オプショナルバリューを安全にアンラップする方法としてオリジナルバインディングがある。オプショナルバインディングとは、オプショナルバリューがnilでなければアンラップして変数に代入する構文で、目的に応じたいくつかのやり方がある。
guard let-else文は関数を実行する前にオプショナルバリューをチェックするために用いるオプショナルバインディングの構文(制御構文)。場合によっては変数はletではなくvarで宣言しても構わない。

guard let-elseでオプショナルバインディング

```
guard let 変数 = オプショナルバリュー else {
   オプショナルバリューがnilの場合のステートメント
   return 戻り値(処理を中断する)
}
アンラップして変数に代入した値を使うステートメント
```

オプショナルバリューがnilではない時は,optional(値)のようにラップされている値をアンラップして変数に代入してguard文を抜け、そのままguard文より以降の処理を実行する。値がnilだった場合はelse[...]のブロックを実行し、returnで値を戻して関数を中断する。guard let-else文が書いてある関数が値を戻さない場合はreturnだけを実行して処理を実行する。

```
//個数のチェック
func kosuCheck(min:Int,max:Int) -> Bool {
    guard  let num = Int(kosu) else {
        //Int(kosu)がnilになった場合はfalseを返して処理を中断する。
        return false
    }
    //returnに入っていればtrue
    return (min...max).contains(num)
}
```

if let-elseを使ったオプショナルバインディング

if let-else文でもオプショナルバリューがnil出ない時に値をアンラップして変数に代入する。そして変数の値を使うために続くブロックを実行する。オプショナルバリューがnilだった場合はelse

のブロックを実行する。guard let-else文は値がnilだった場合に、処理をキャンセルしていることをはっきり示すのが目的だが、if let-else文は場合に応じて処理を分岐して続けたい場合に使われる。

if let-elseでオプショナルバインディング

```
if let 変数 = オプショナルバリュー{
    アンラップして変数に代入した値を使うステートメント
}else{
   オプショナルバリューがnilの場合のステートメント
}
```

```
//料金の計算をする
func price() -> Int {
    if let num = Double(kosu) {
        let result = Int(tanka * num * tax)
        return result
    } else {
        return -1
    }
}
```