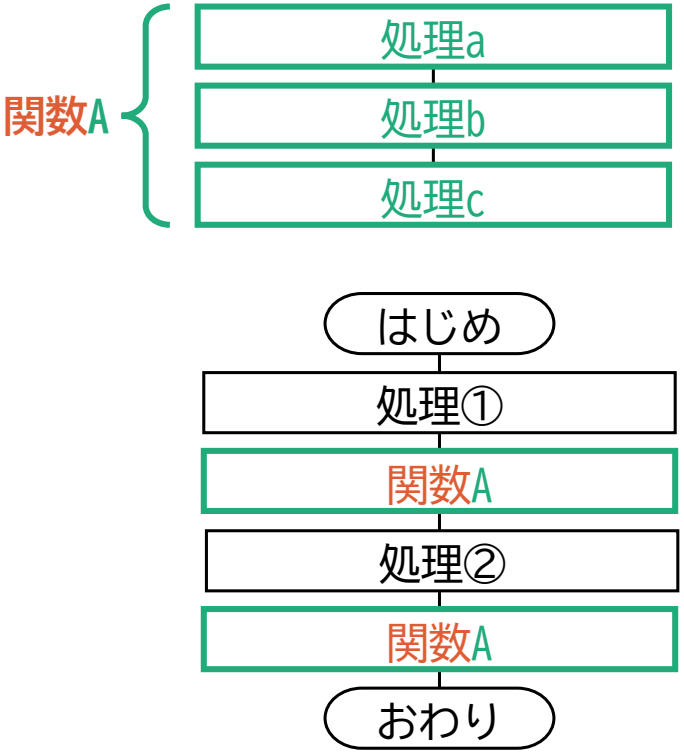
速習！情報 ～共通テスト対策講座～

プログラミング（関数）

関数、引数、戻り値
組み込み関数、ユーザ定義関数、乱数

複雑なプログラムは、機能単位で切り出した関数に分けていく

プログラム内で同じ処理が出てくる場合、処理のかたまりを関数として定義する



- プログラムを読むときに機能のかたまりの単位で区切られていて、理解しやすい
- 再利用しやすい
- 修正を加える際に修正が1箇所で済むので、修正しやすい
- 不具合がないかのテストを機能単位で行えるので、テストしやすく品質が向上する

関数

- プログラミング言語であらかじめ定義されている関数を組み込み関数という
- ユーザが自分で作成する関数をユーザ定義関数という
- 関数に引き渡すデータを引数（ひきすう）、関数から呼び出し元に戻されるデータを戻り値または返り値という

インプットするデータ

アウトプットされたデータ

組み込み関数の例

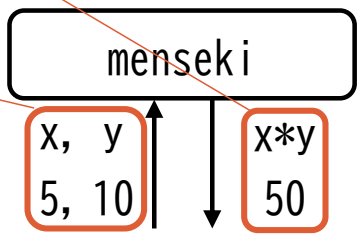
```
表示する(”にゃー”)

乱数()
```

- 数字が不規則(ランダム)に並んだ数を乱数という
  (例)0.3185493186
- 乱数という関数は、実行ごとに異なる値を出力する

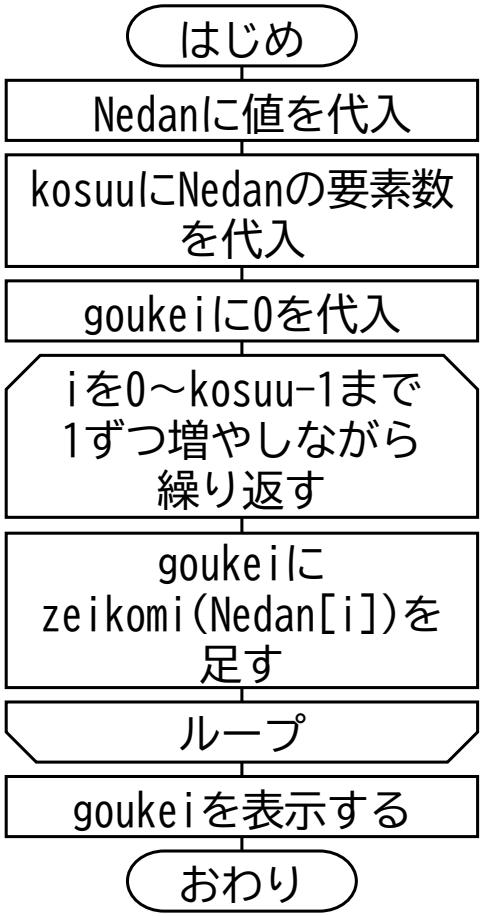
ユーザ定義関数の例

```
関数 menseki(x,y) を定義する：
  └ 返す( x * y )

hirosa = menseki(5,10)
```

hirosaという変数に50という値が代入される



関数は呼び出されたときだけ処理される



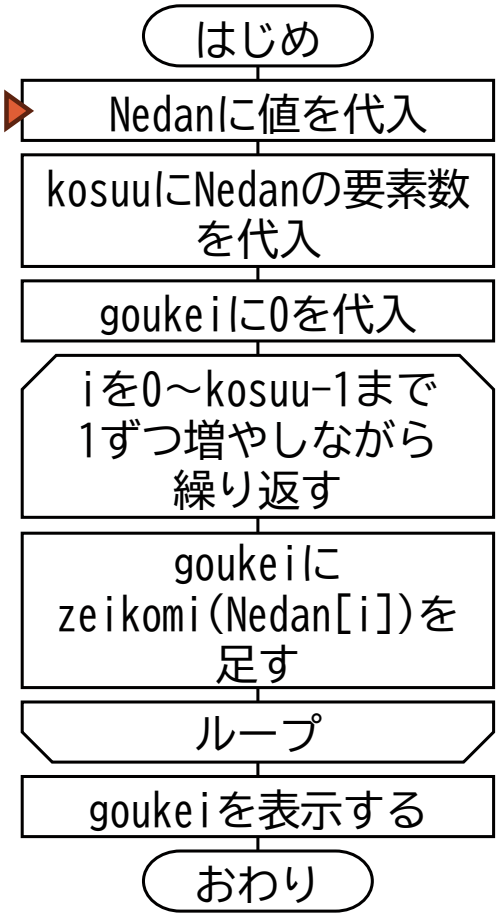
共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

コメント
「#」以降は無視され実行されない

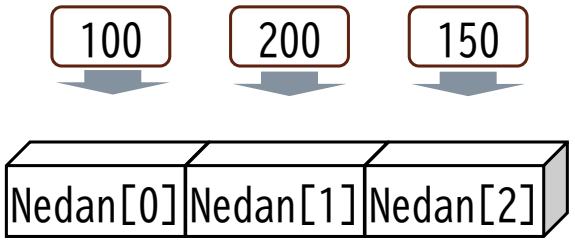
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない



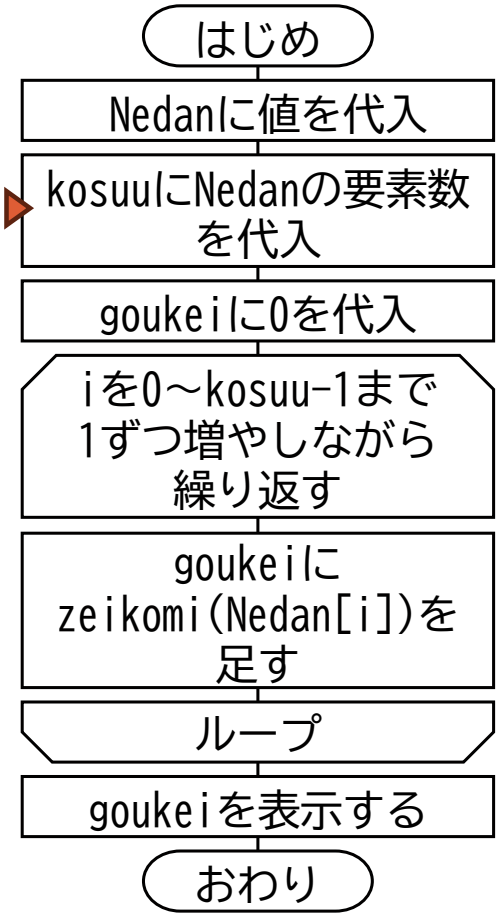
共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
 ┗ 返す( x * 1.1 )　# 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)　# 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
 ┗ goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



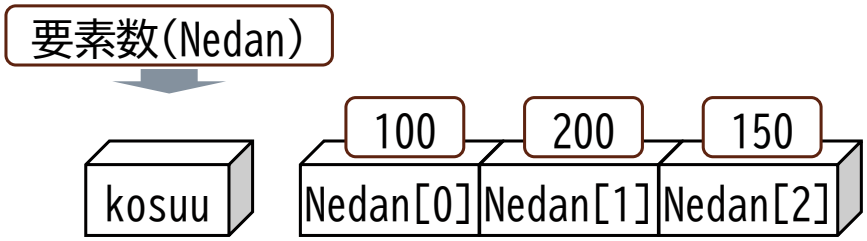
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない



共通テストで使用されるDNCL

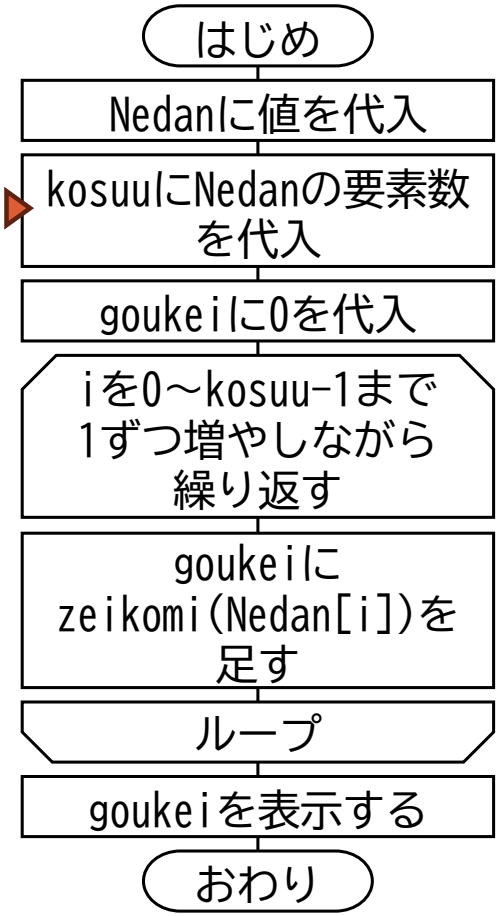
```
関数 zeikomi(x) を定義する：
 ┗ 返す( x * 1.1 )　# 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)　# 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
 ┗ goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



赤字：用語も意味も理解が必要、緑字：意味や概念の理解が必要

関数は呼び出されたときだけ処理される

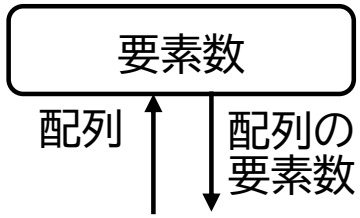
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する

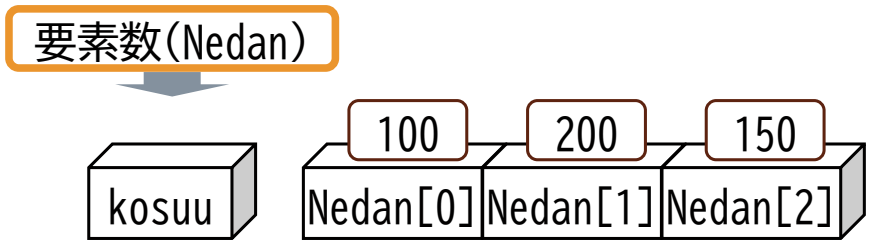


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
  └ 返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
  └ goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

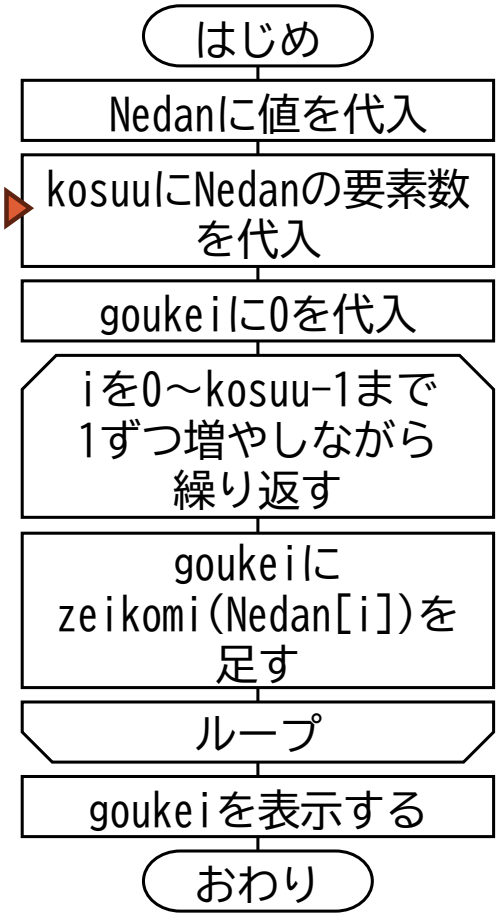
通常、問題文中に関数の意味が記載されている





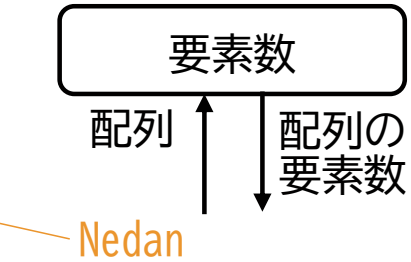
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する

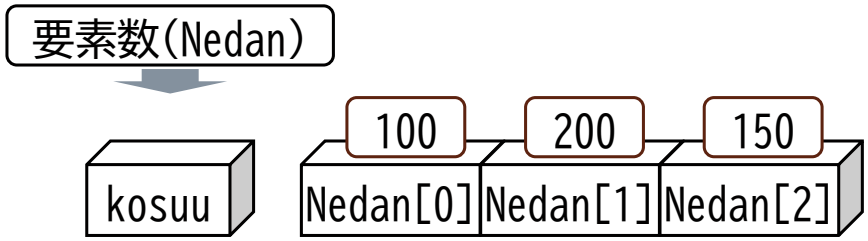


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



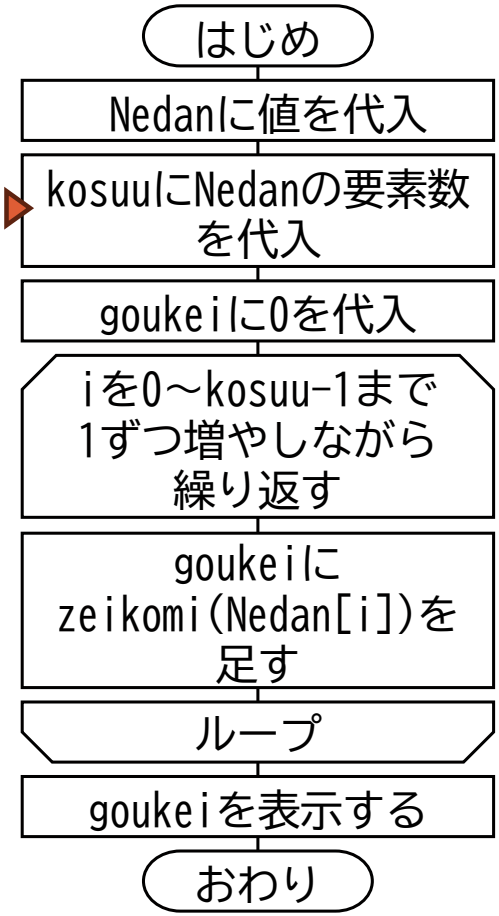
Nedan
引数(インプットする値)がNedan



赤字：用語も意味も理解が必要、**緑字**：意味や概念の理解が必要

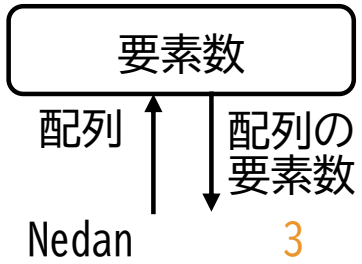
関数は呼び出されたときだけ処理される

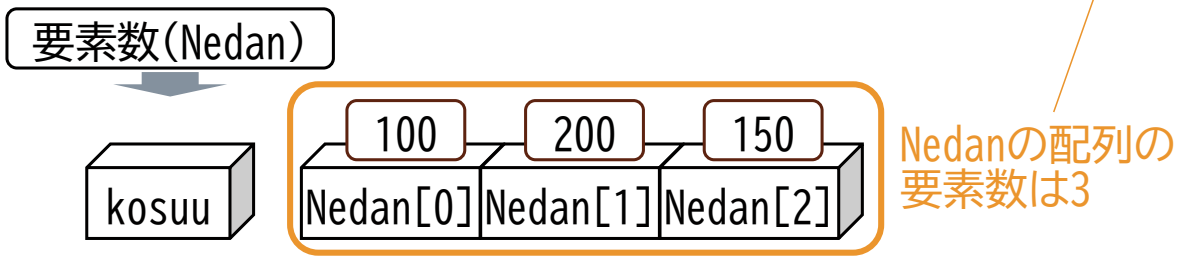
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する



共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

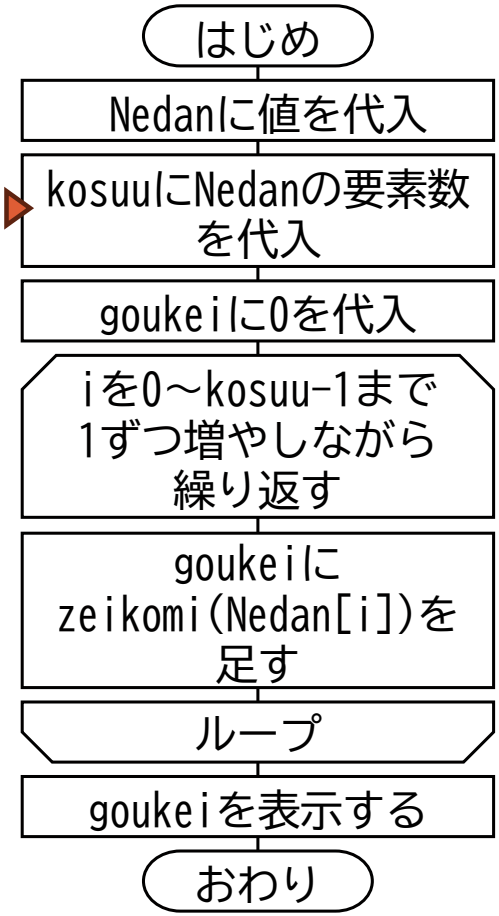




赤字：用語も意味も理解が必要、**緑字**：意味や概念の理解が必要

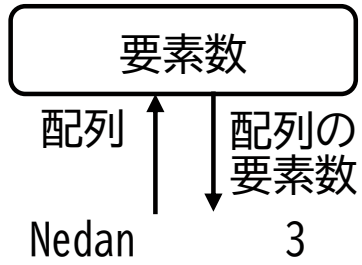
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する

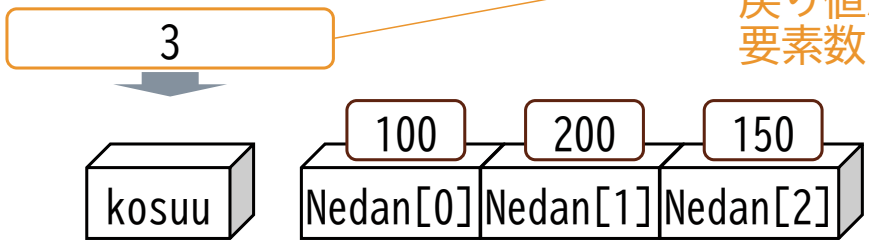


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
 └ 返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
 └ goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

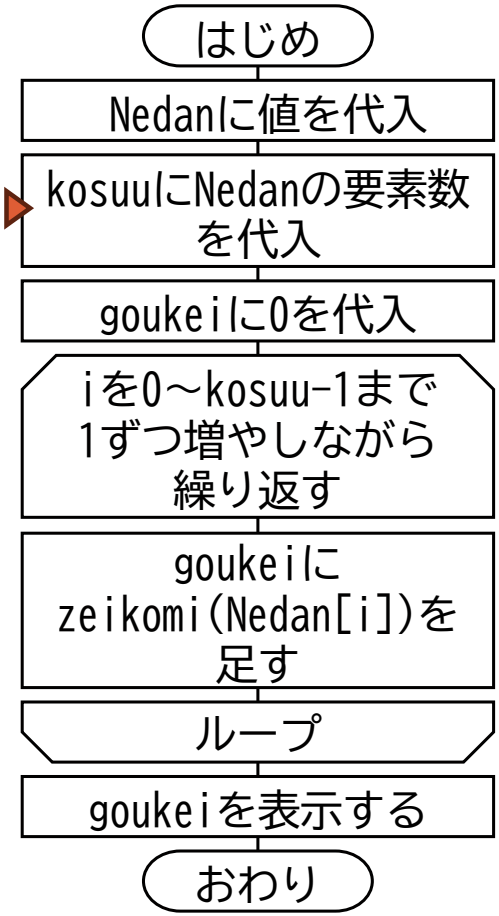


戻り値が3なので、
要素数(Nedan)が3になる



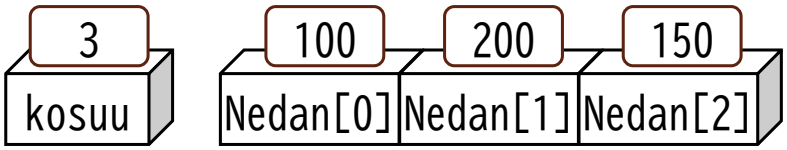
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する



共通テストで使用されるDNCL

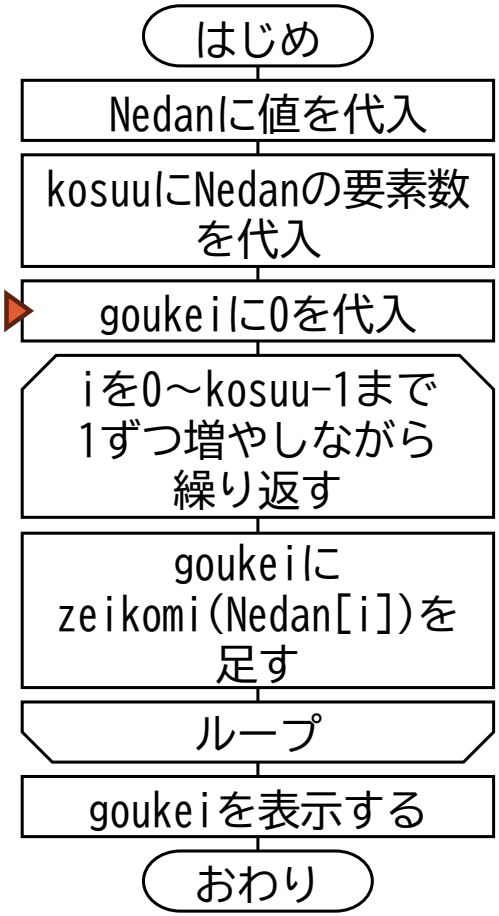
```
関数 zeikomi(x) を定義する：
 ┗ 返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
 ┗ goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



赤字：用語も意味も理解が必要、**緑字**：意味や概念の理解が必要

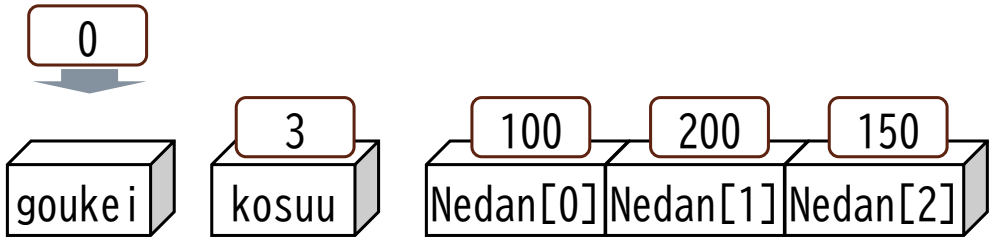
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する



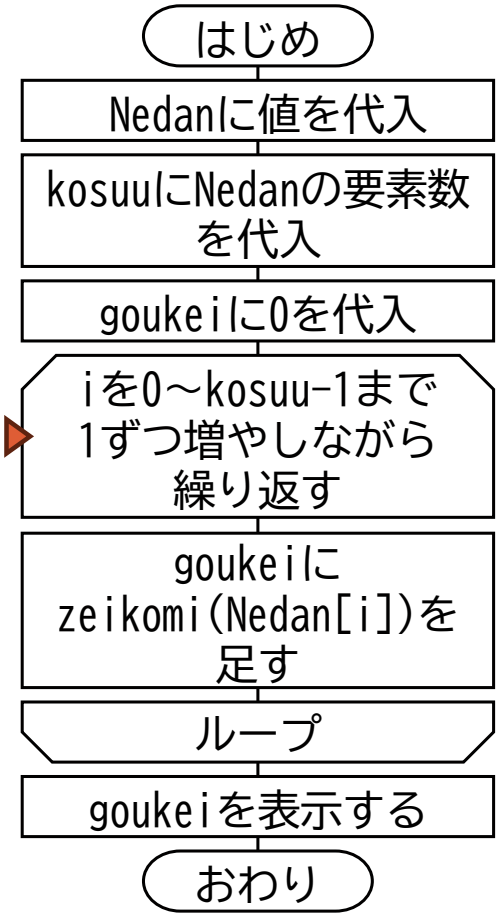
共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
 ┗ 返す( x * 1.1 )　# 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)　# 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
 ┗ goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



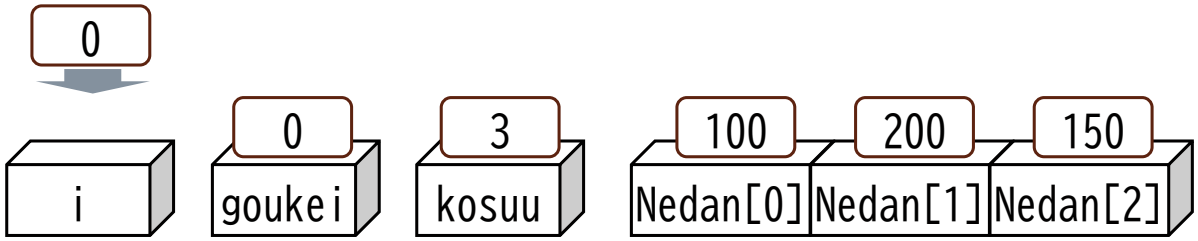
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する



共通テストで使用されるDNCL

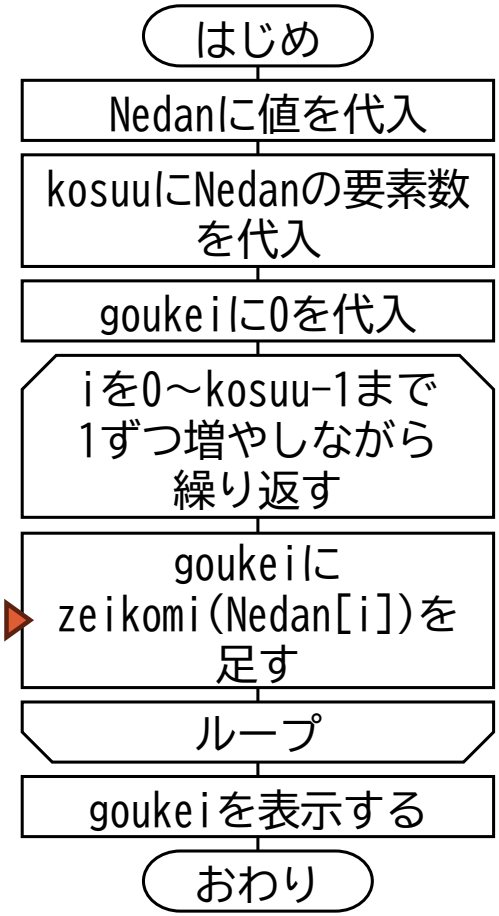
```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



赤字：用語も意味も理解が必要、**緑字**：意味や概念の理解が必要

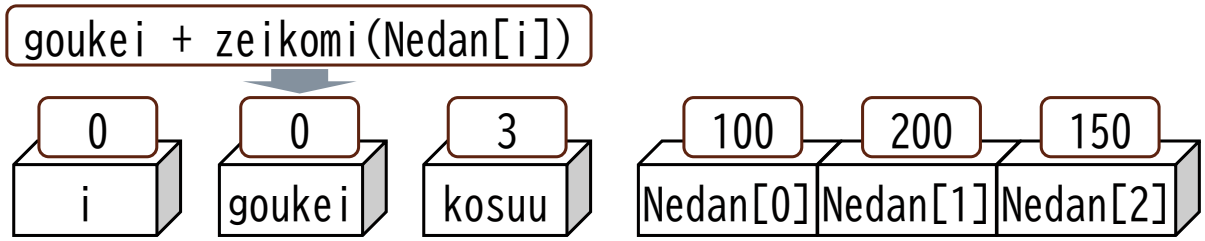
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する



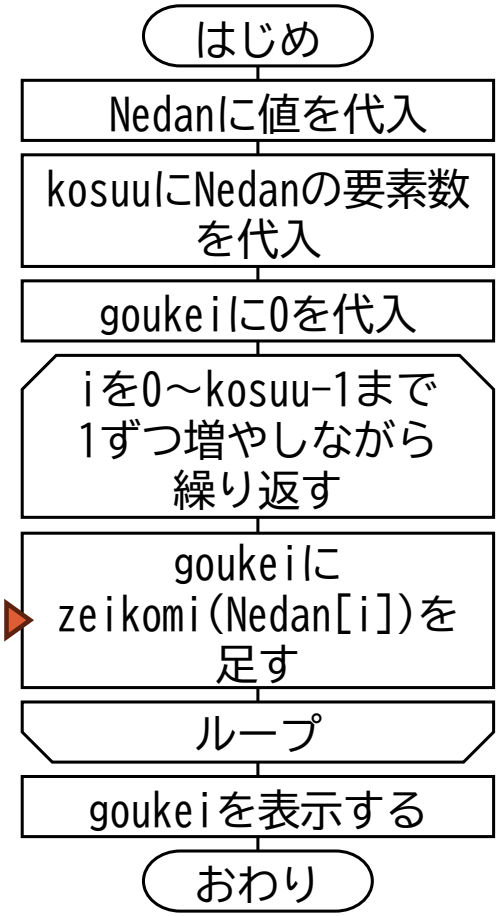
共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



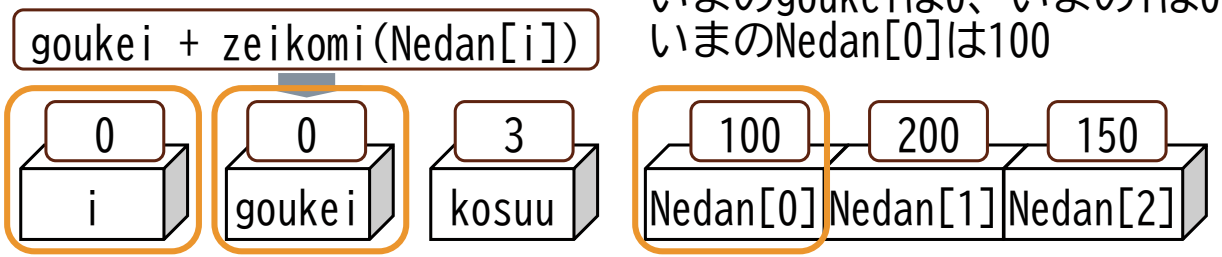
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する



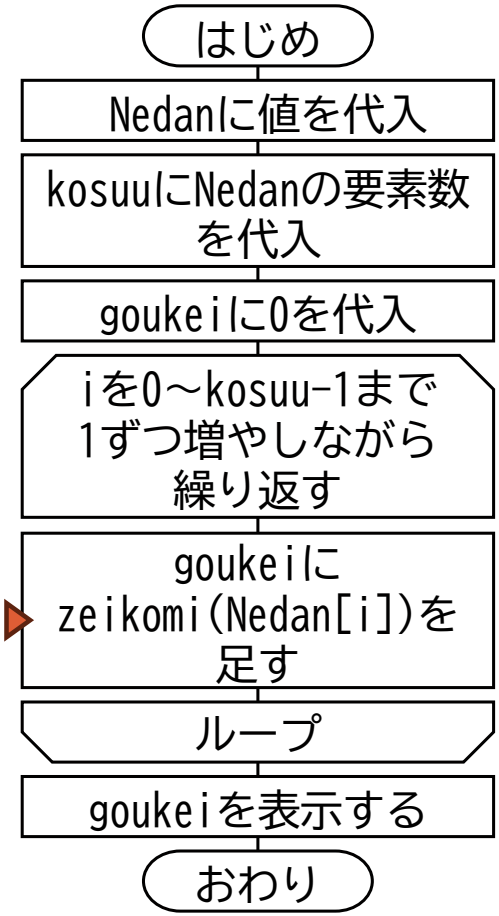
共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



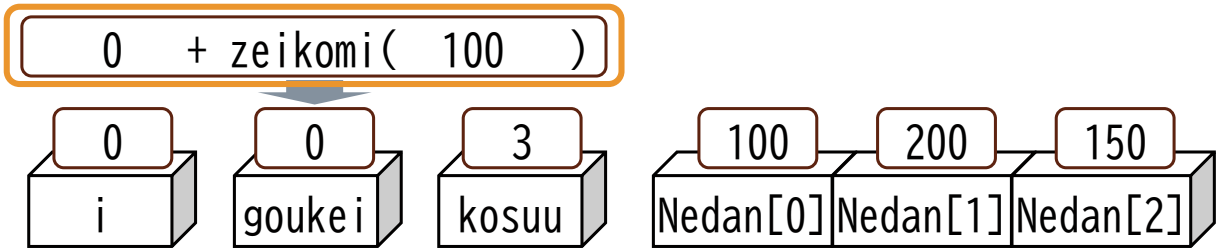
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える



共通テストで使用されるDNCL

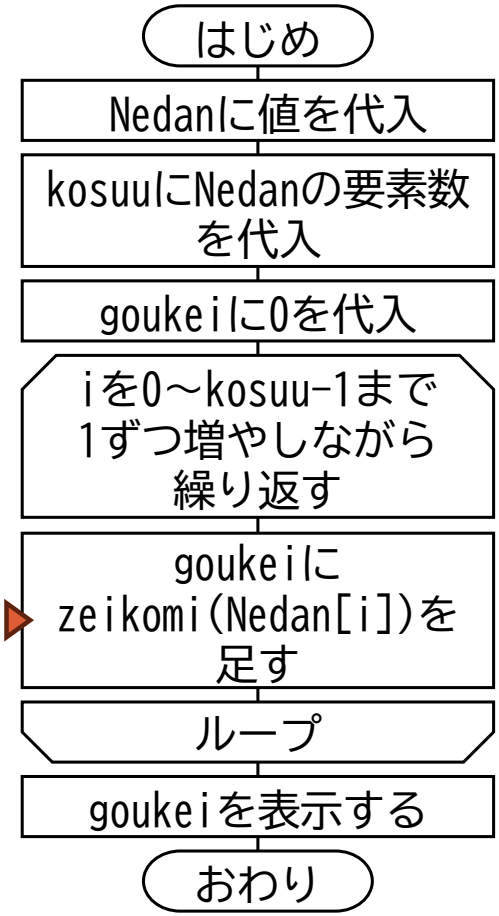
```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



赤字：用語も意味も理解が必要、緑字：意味や概念の理解が必要

関数は呼び出されたときだけ処理される

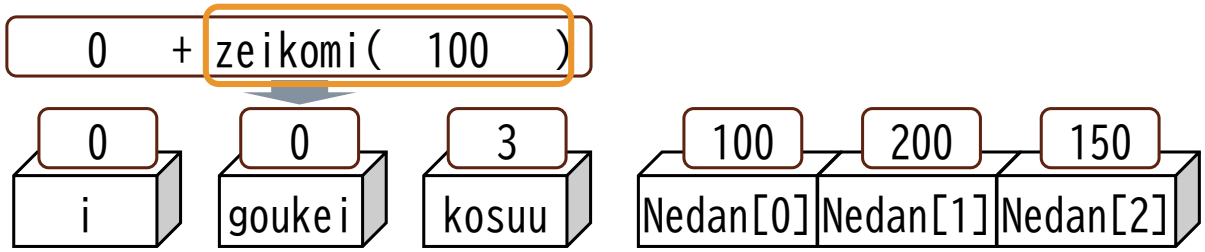
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える



共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

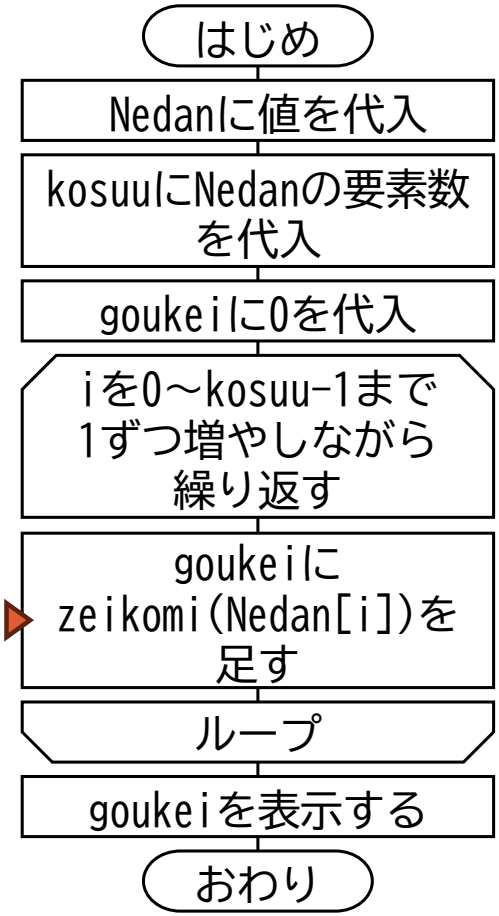
zeikomi



赤字：用語も意味も理解が必要、緑字：意味や概念の理解が必要

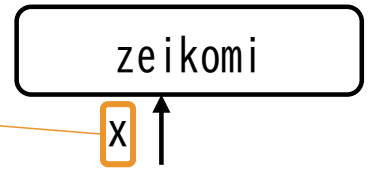
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

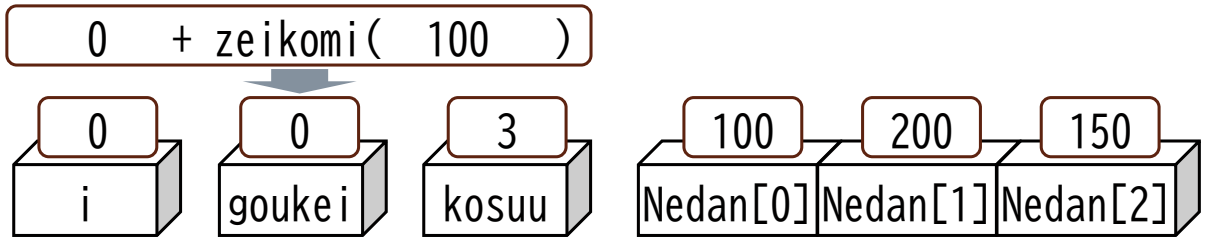


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



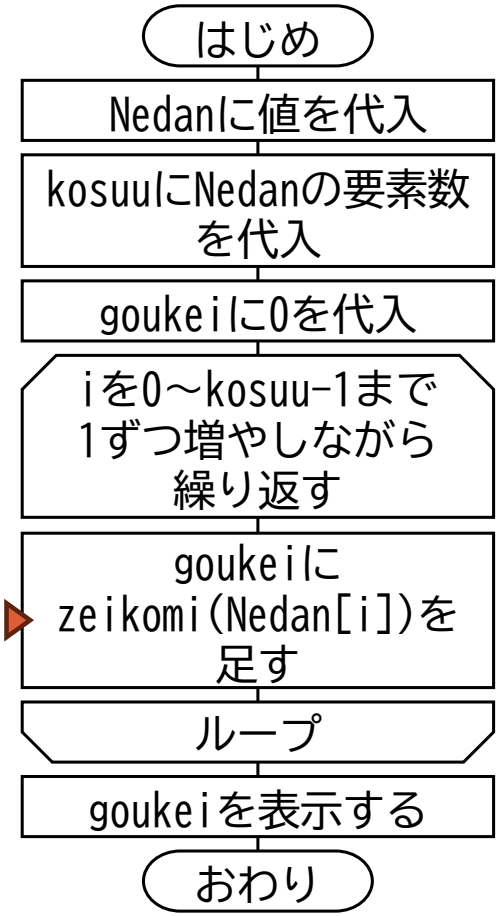
引数（インプットする値）がx



赤字：用語も意味も理解が必要、緑字：意味や概念の理解が必要

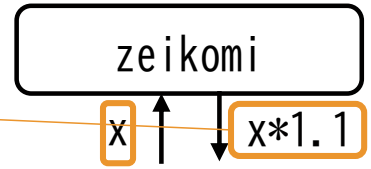
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

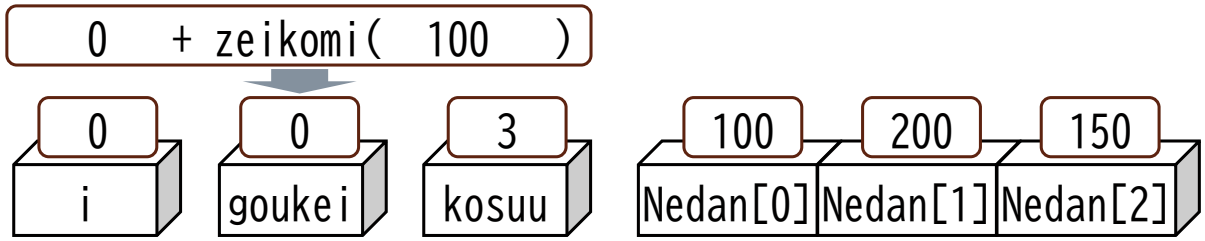


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



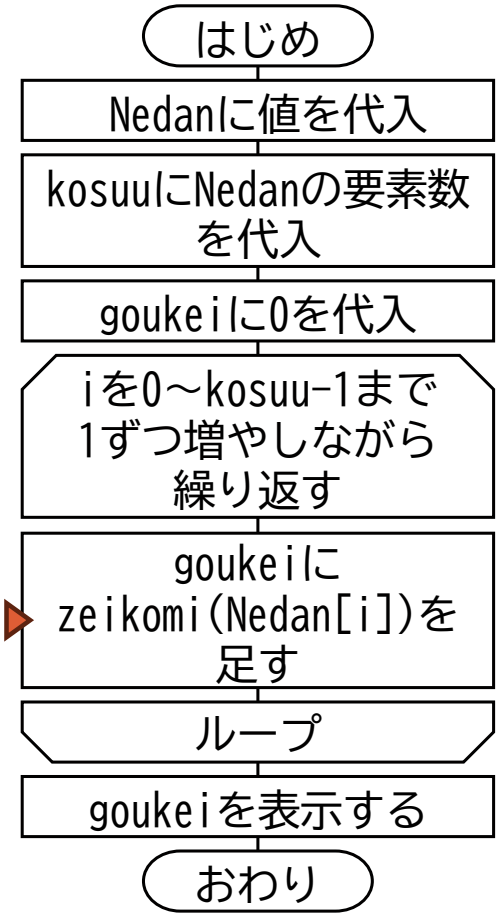
戻り値（アウトプットする値）がx*1.1



赤字：用語も意味も理解が必要、緑字：意味や概念の理解が必要

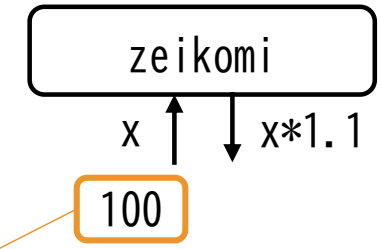
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

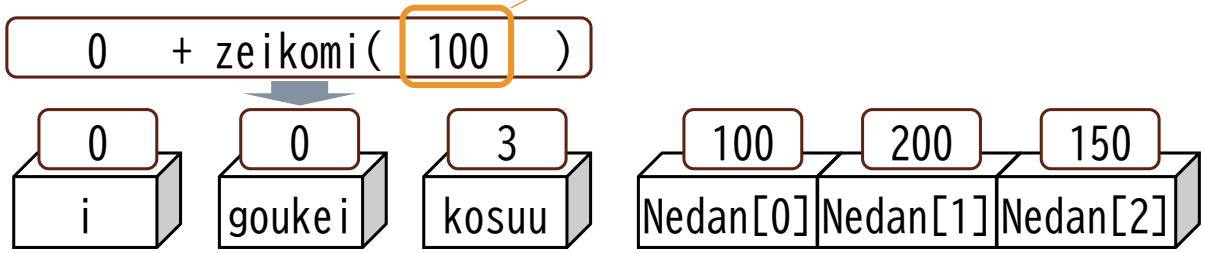


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

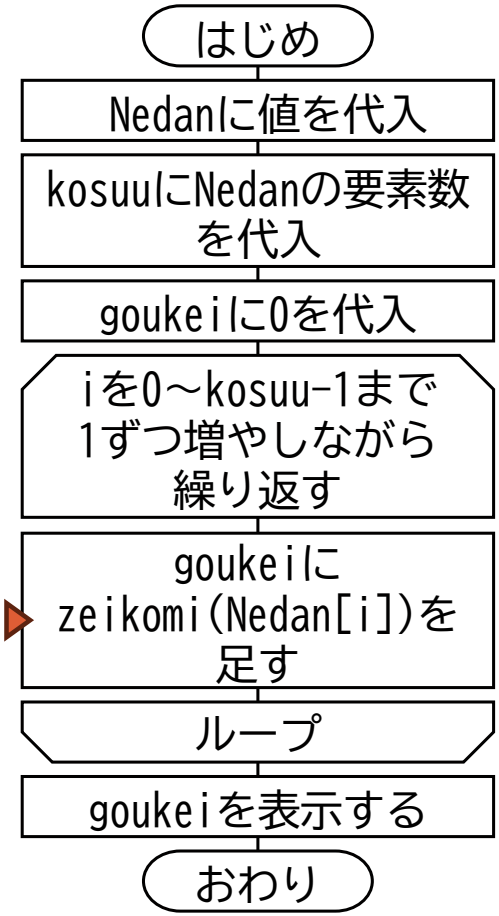


実際の引数（インプットする値）は100



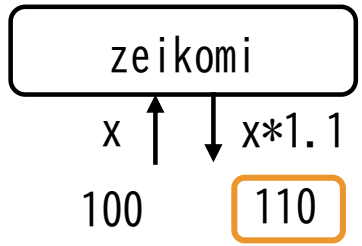
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

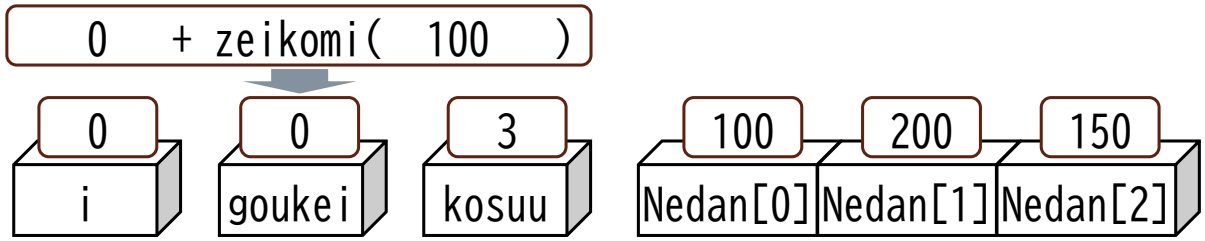


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



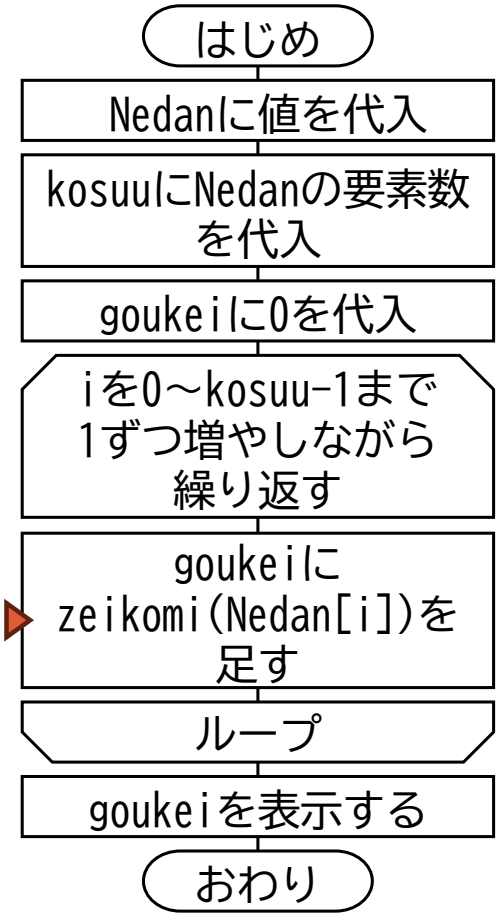
実際の戻り値
（アウトプットする値）
は100×1.1＝110



赤字：用語も意味も理解が必要、緑字：意味や概念の理解が必要

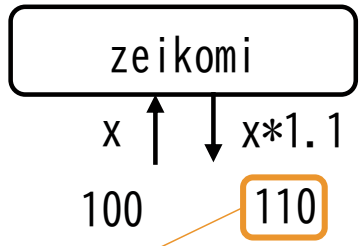
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

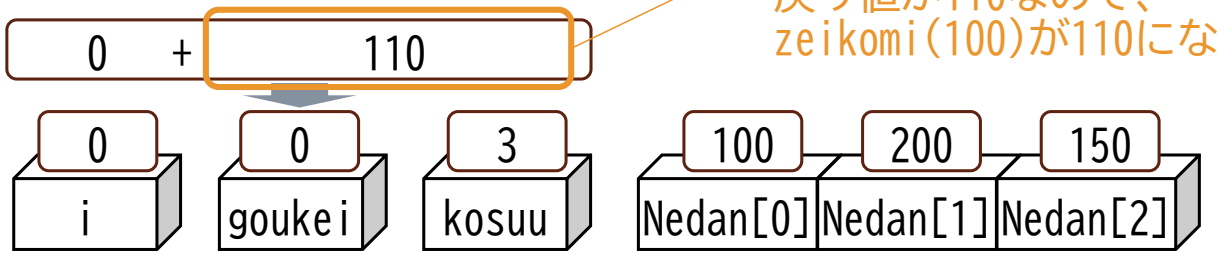


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



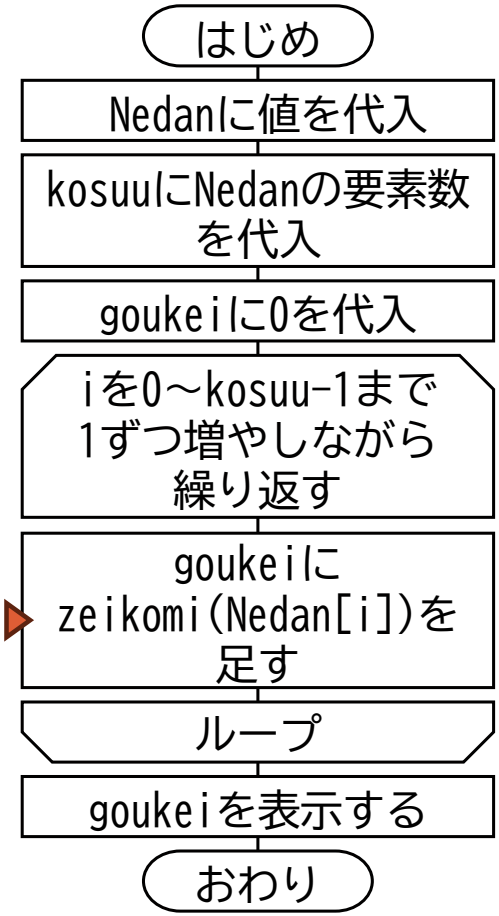
戻り値が110なので、
zeikomi(100)が110になる



赤字：用語も意味も理解が必要、**緑字**：意味や概念の理解が必要

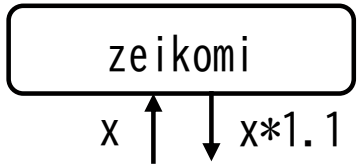
関数は呼び出されたときだけ処理される

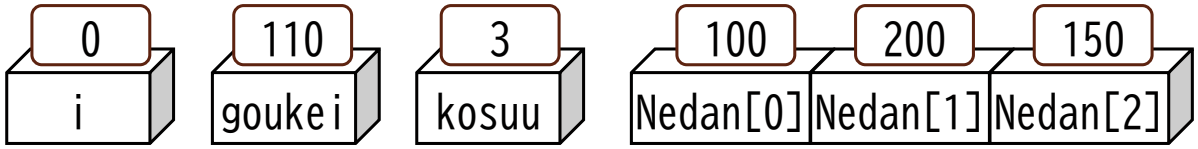
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える



共通テストで使用されるDNCL

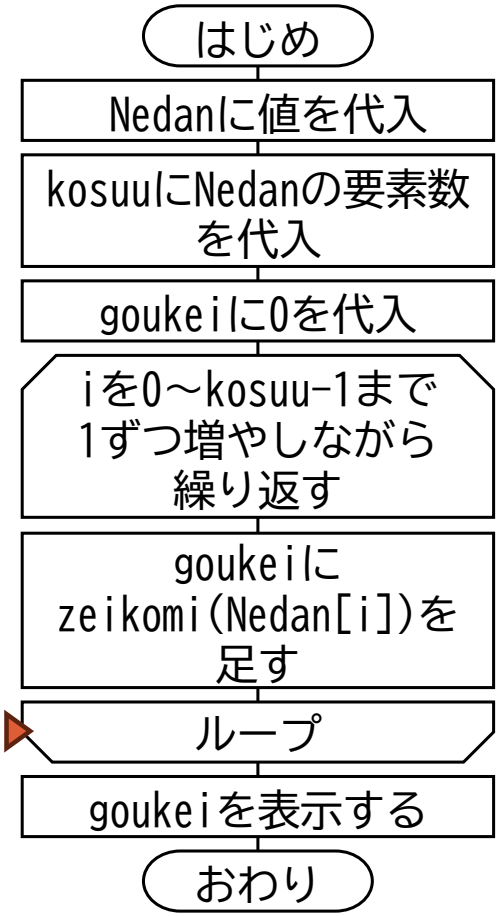
```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```





関数は呼び出されたときだけ処理される

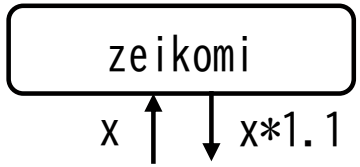
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

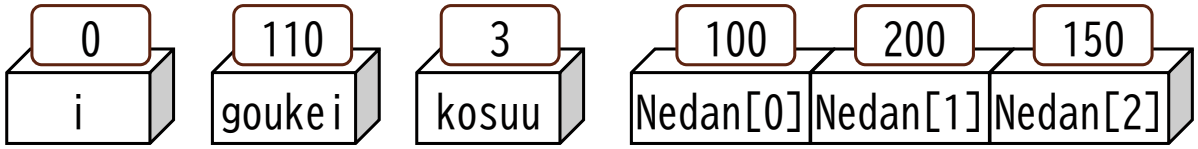


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

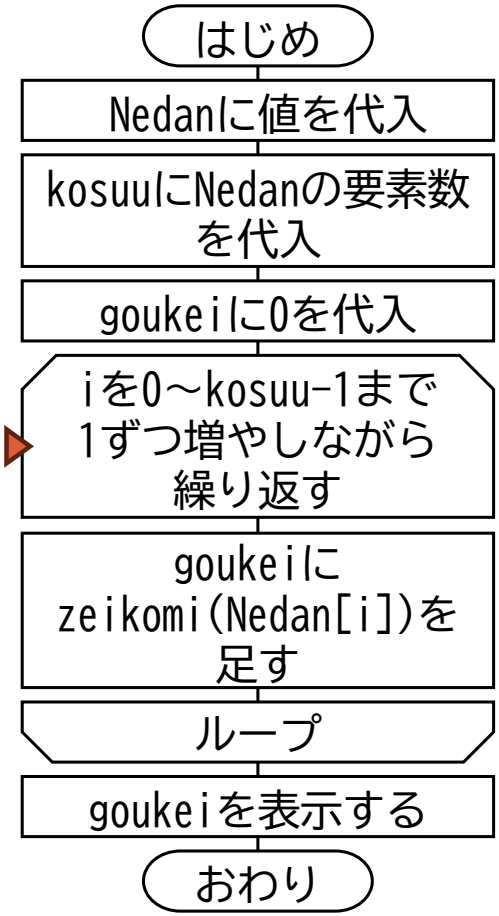
反復処理の終了部分にきたら開始部分に戻る





関数は呼び出されたときだけ処理される

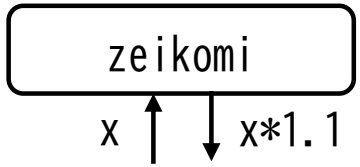
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

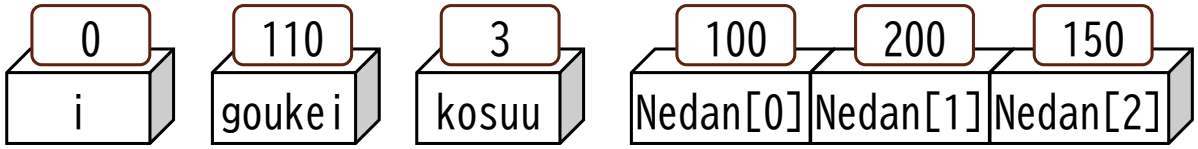


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

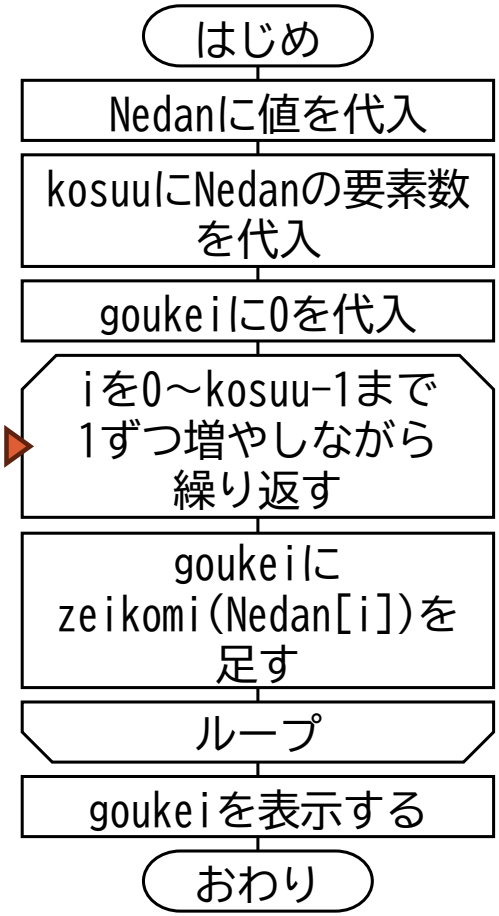
反復処理の終了部分にきたら開始部分に戻る





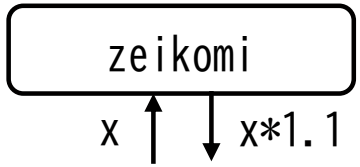
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える



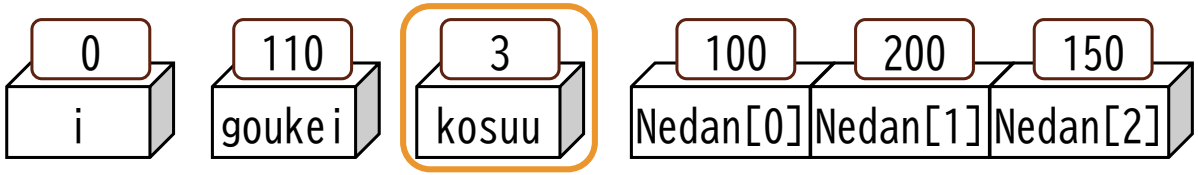
共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



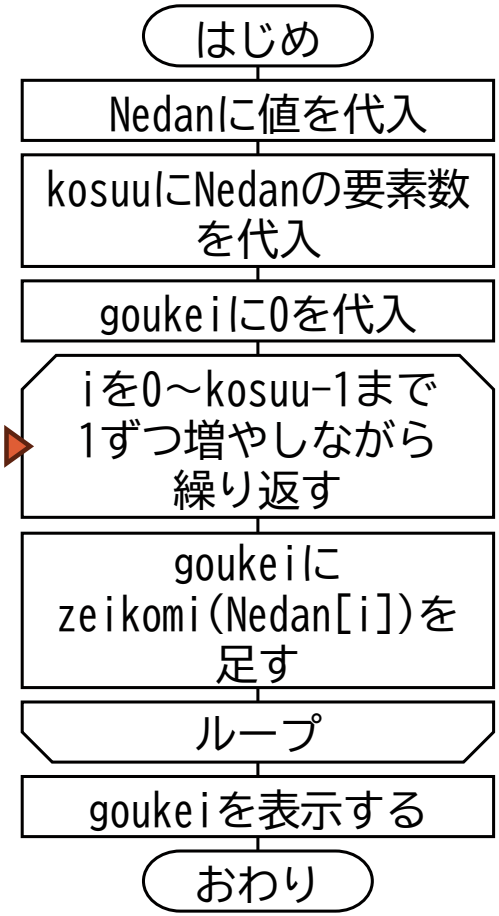
iを0から2まで1ずつ増やす

いまのkosuu-1は2



関数は呼び出されたときだけ処理される

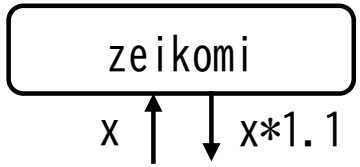
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

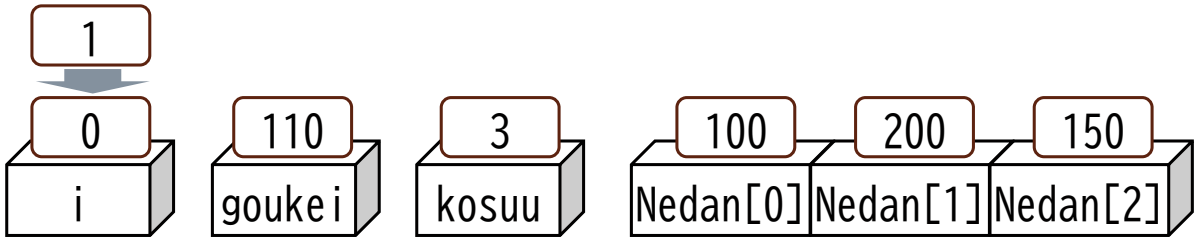


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

iを0から2まで1ずつ増やす

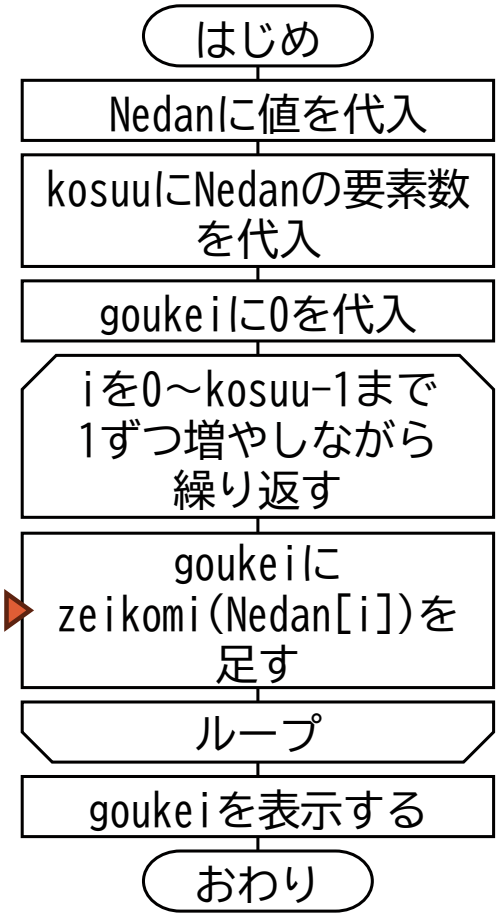




赤字：用語も意味も理解が必要、緑字：意味や概念の理解が必要

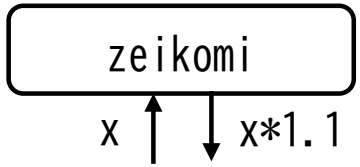
関数は呼び出されたときだけ処理される

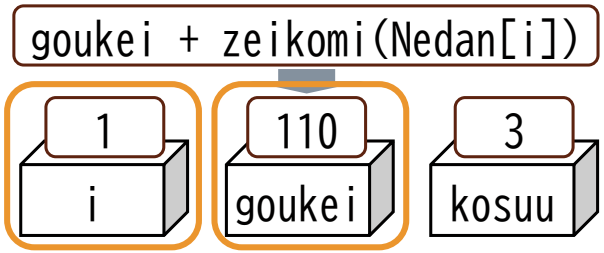
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える



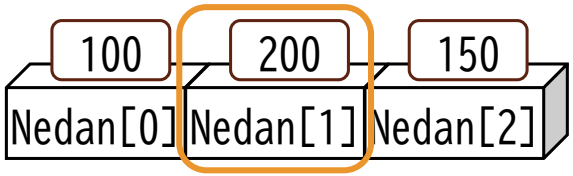
共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



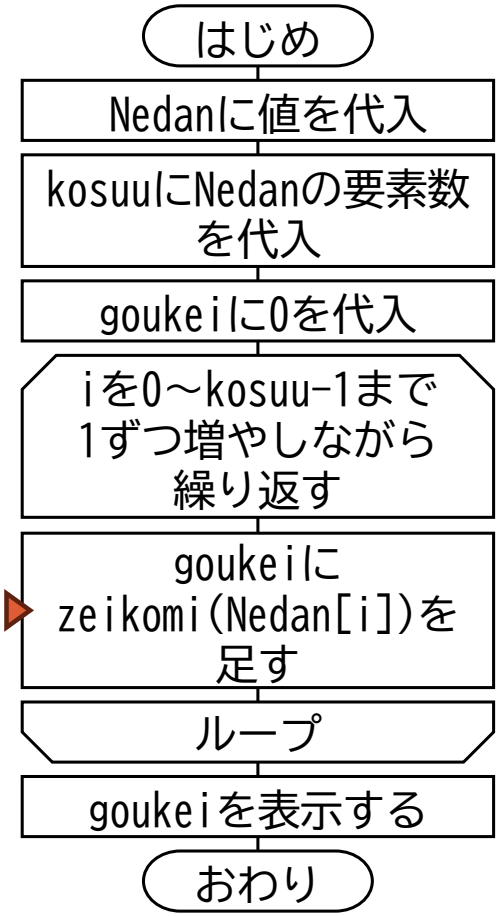


いまのgoukeiは110、いまのiは1
いまのNedan[1]は200



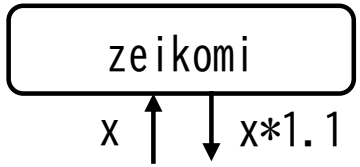
関数は呼び出されたときだけ処理される

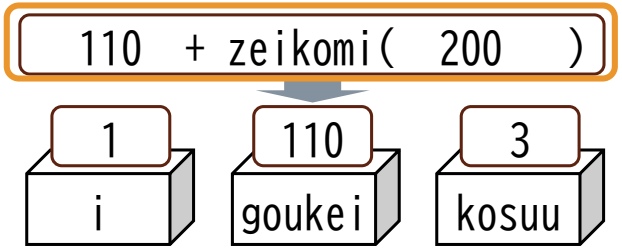
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える



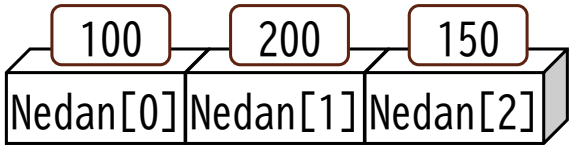
共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```





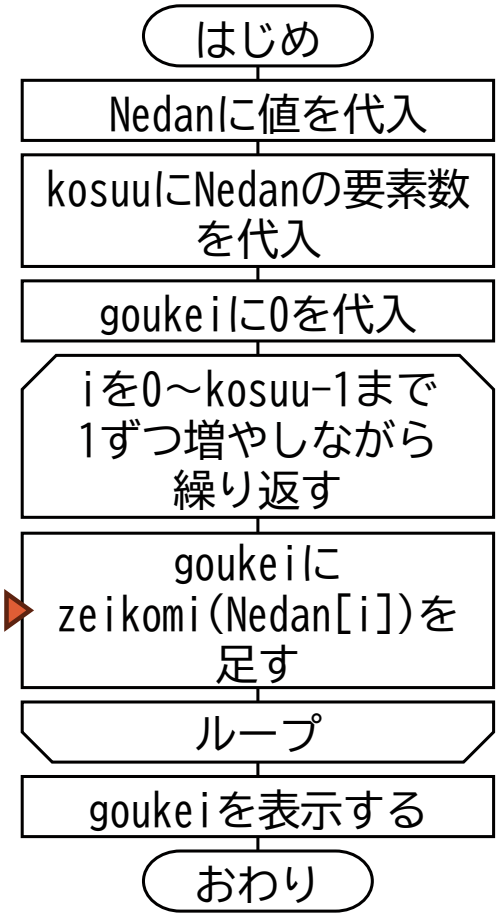
いまのgoukeiは110、いまのiは1
いまのNedan[1]は200



赤字：用語も意味も理解が必要、緑字：意味や概念の理解が必要

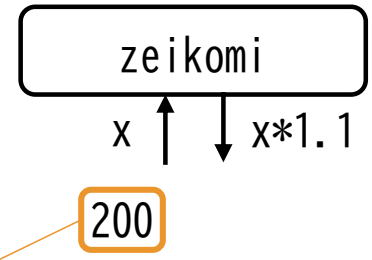
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

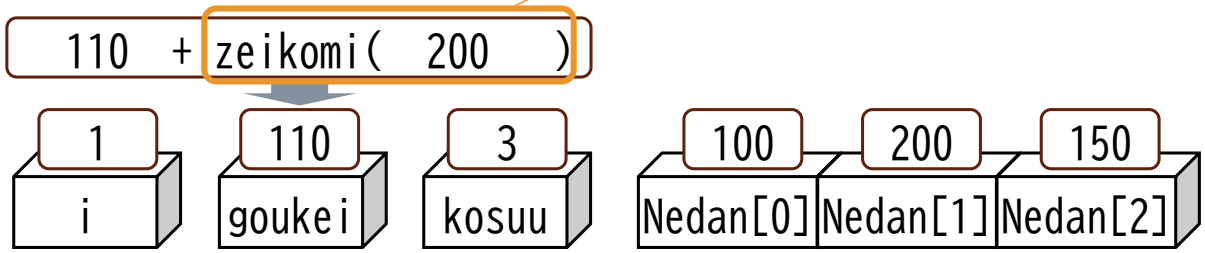


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

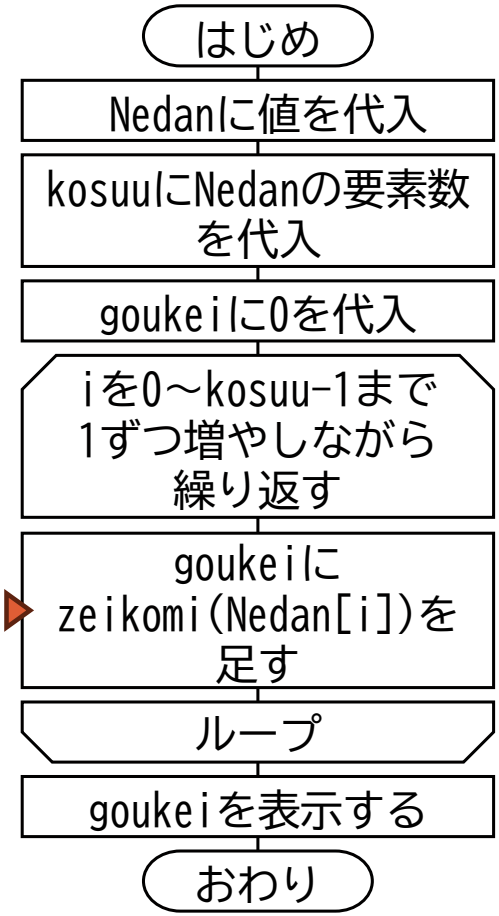


実際の引数（インプットする値）は200



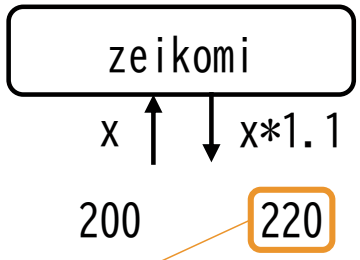
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

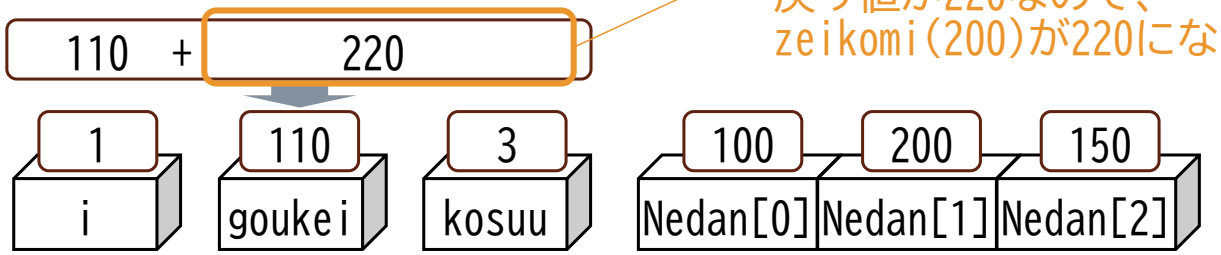


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



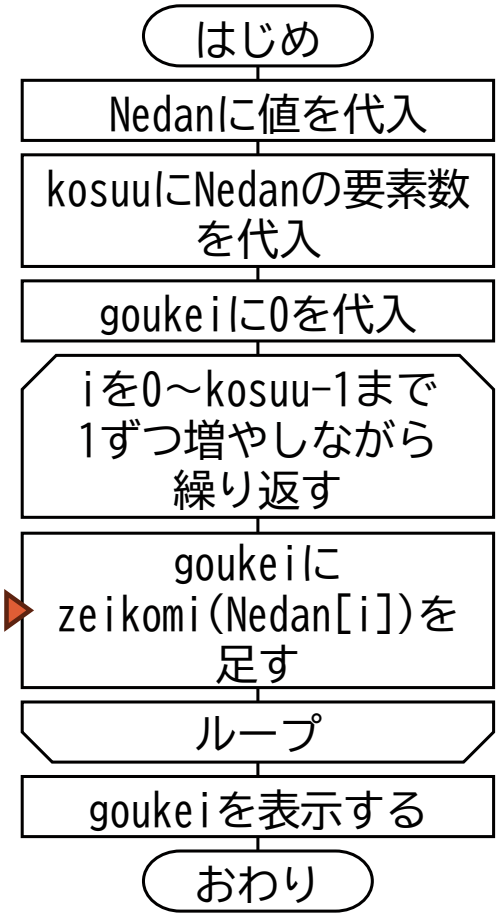
戻り値が220なので、
zeikomi(200)が220になる



赤字：用語も意味も理解が必要、**緑字**：意味や概念の理解が必要

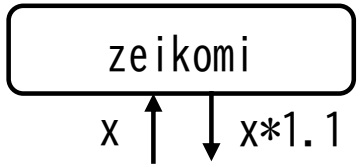
関数は呼び出されたときだけ処理される

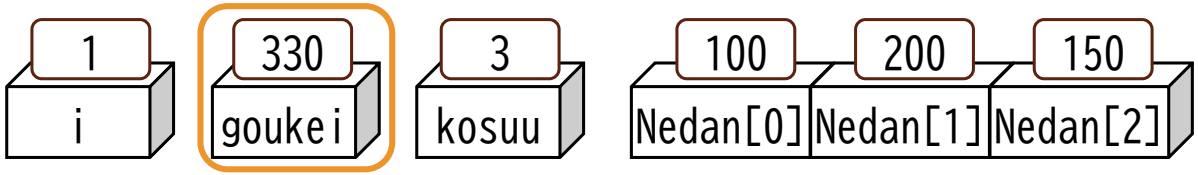
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える



共通テストで使用されるDNCL

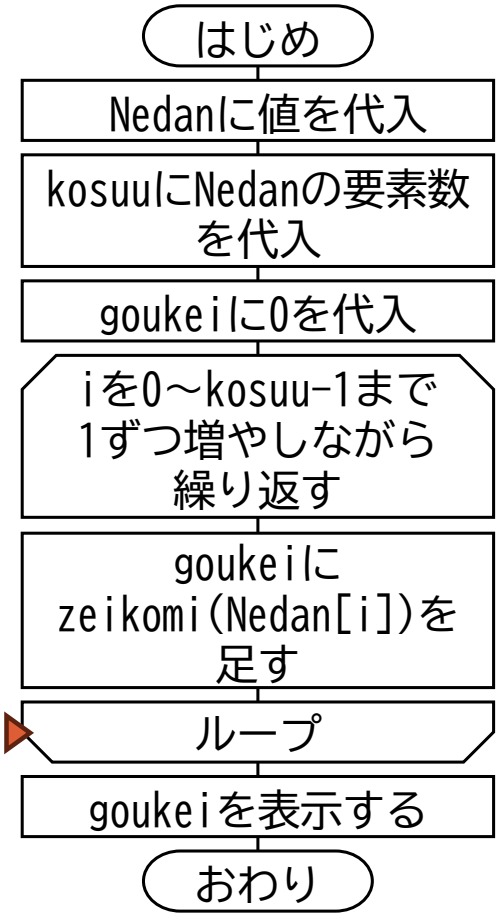
```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```





関数は呼び出されたときだけ処理される

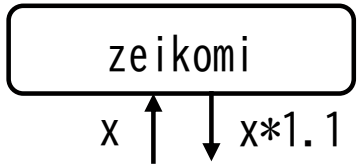
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

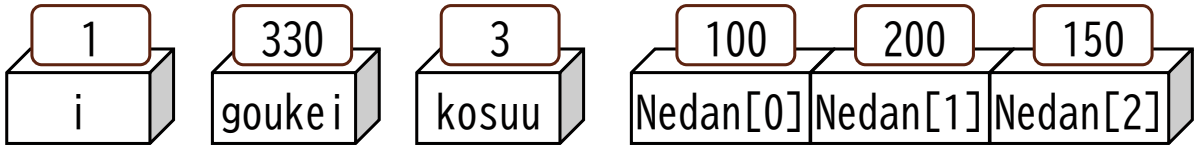


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

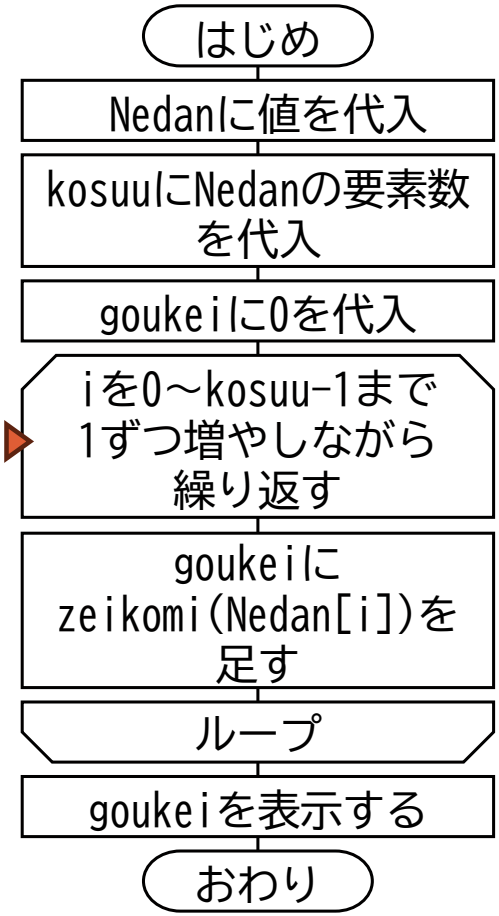
反復処理の終了部分にきたら開始部分に戻る





関数は呼び出されたときだけ処理される

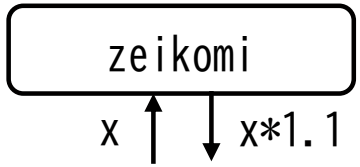
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

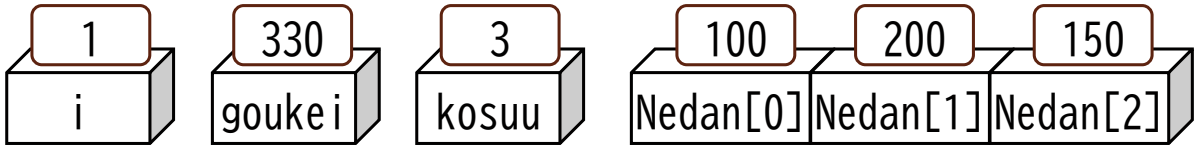


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

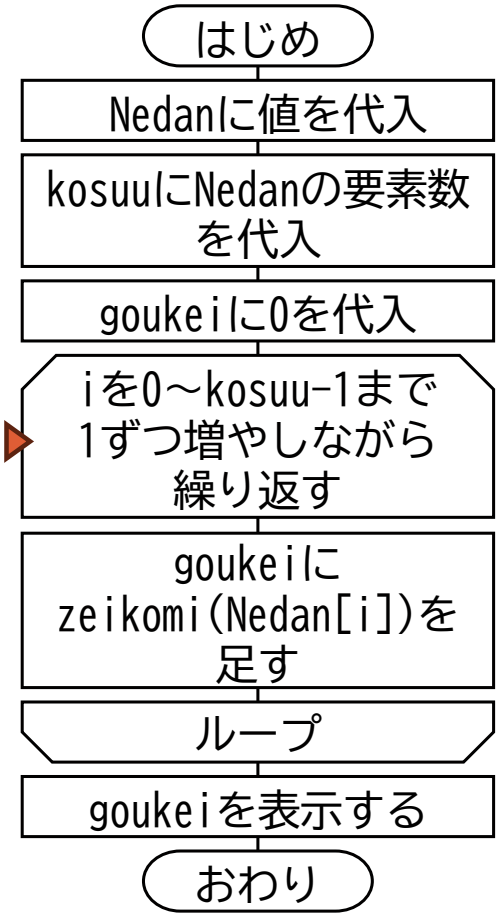
反復処理の終了部分にきたら開始部分に戻る





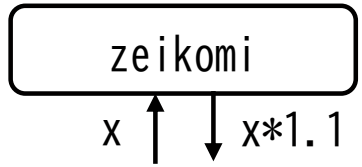
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える



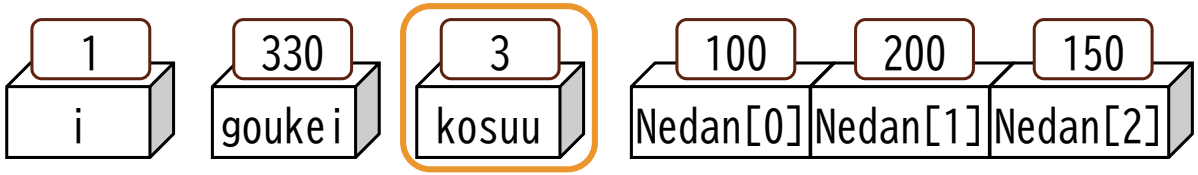
共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



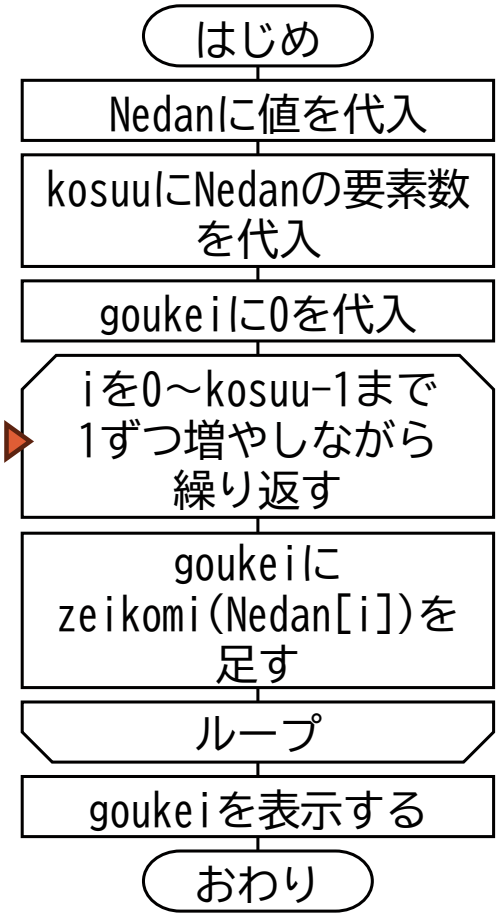
iを0から2まで1ずつ増やす

いまのkosuu-1は2



関数は呼び出されたときだけ処理される

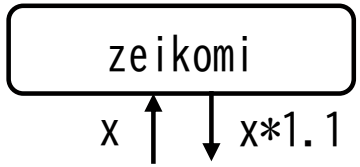
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

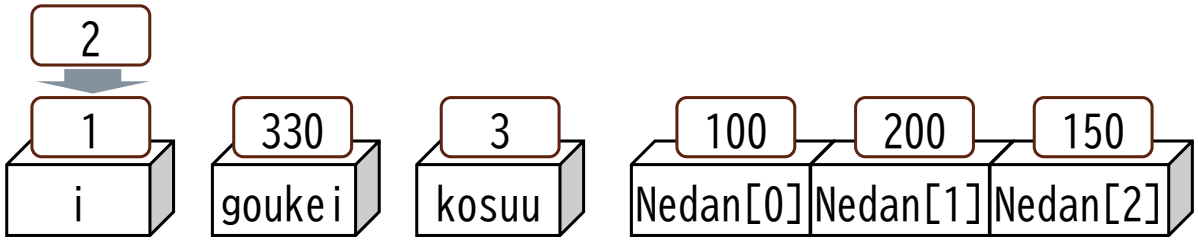


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

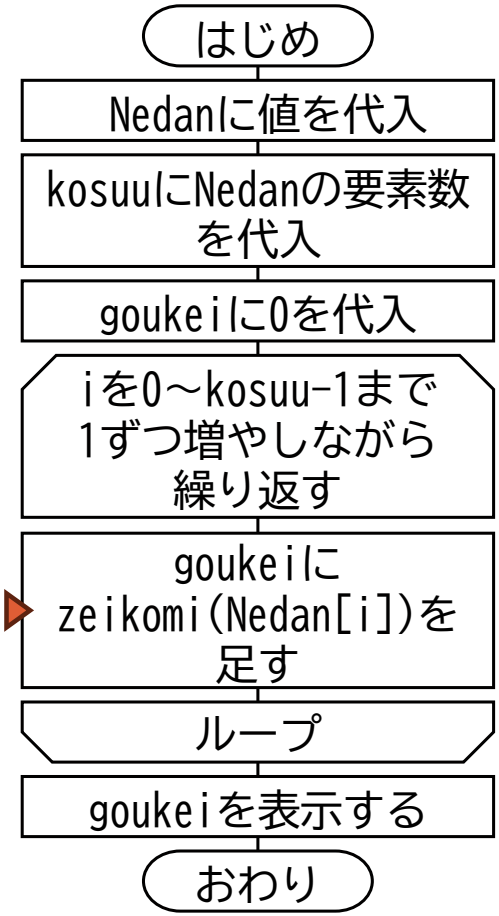
iを0から2まで1ずつ増やす





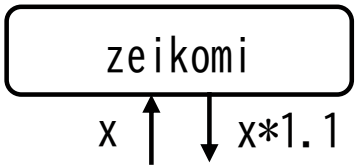
関数は呼び出されたときだけ処理される

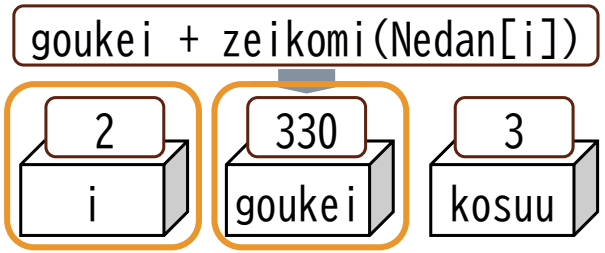
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える



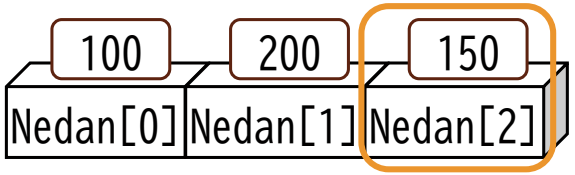
共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



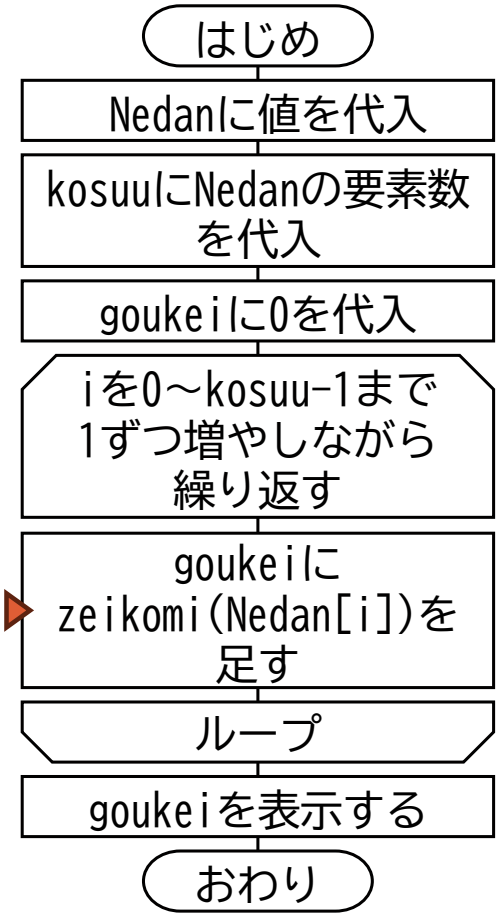


いまのgoukeiは330、いまのiは2
いまのNedan[2]は150



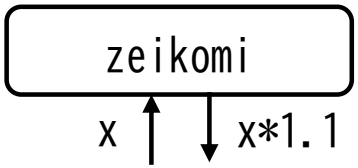
関数は呼び出されたときだけ処理される

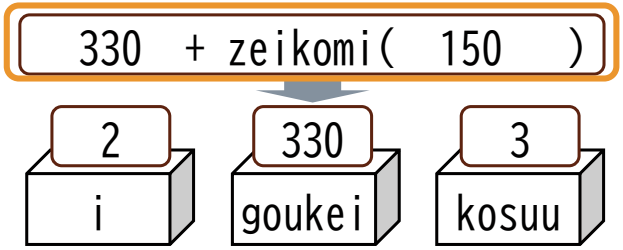
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える



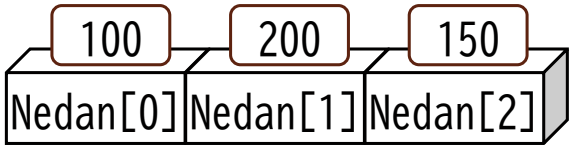
共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



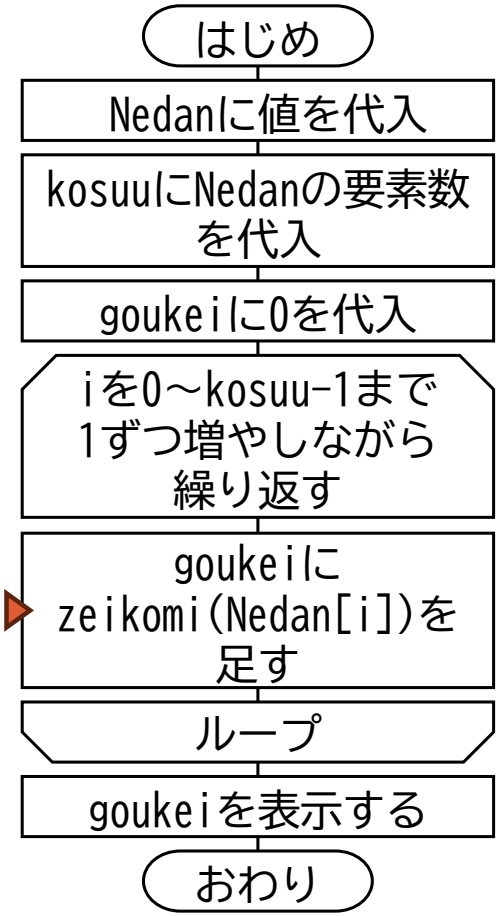


いまのgoukeiは330、いまのiは2
いまのNedan[2]は150



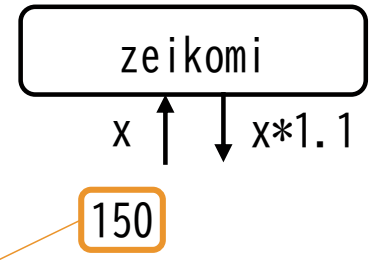
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

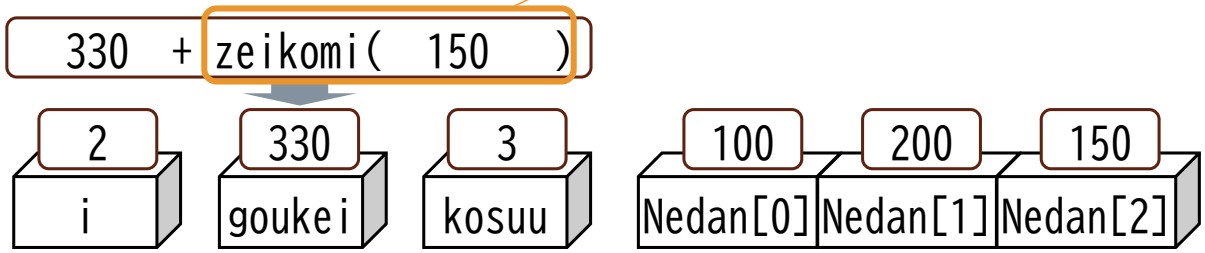


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

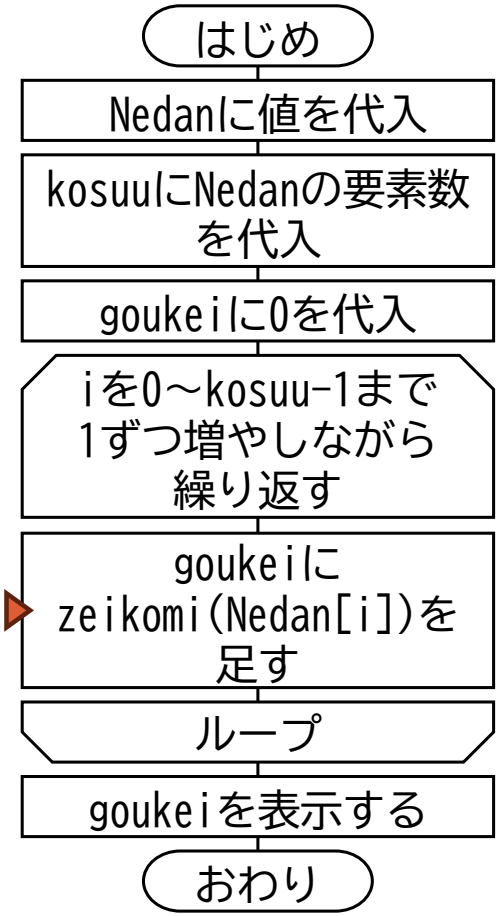


実際の引数（インプットする値）は150



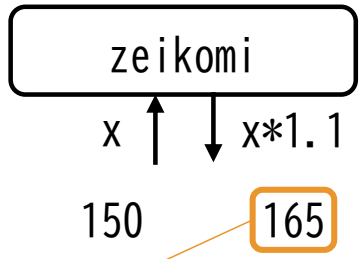
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

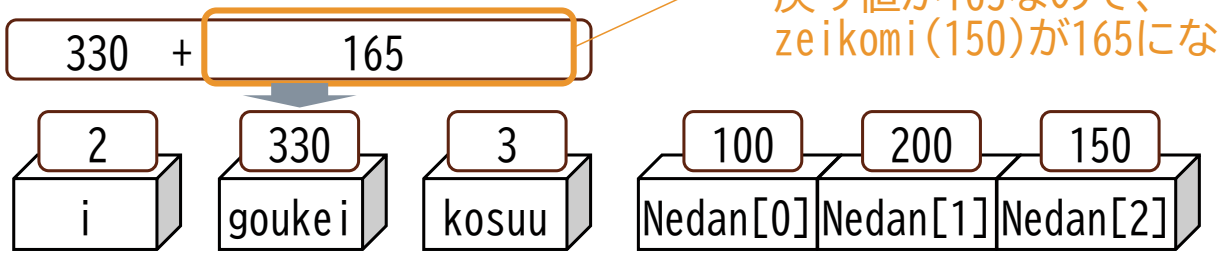


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



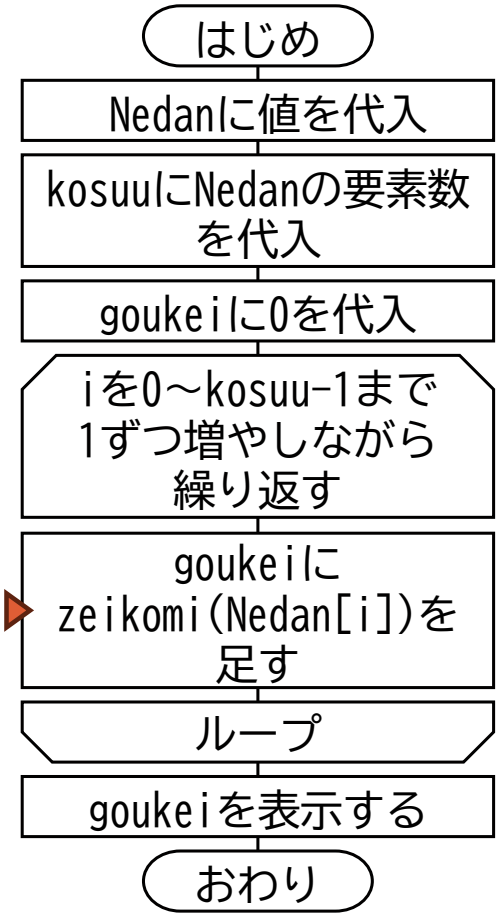
戻り値が165なので、
zeikomi(150)が165になる



赤字：用語も意味も理解が必要、**緑字**：意味や概念の理解が必要

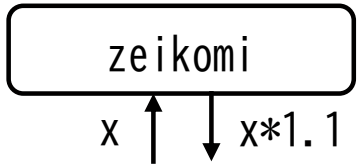
関数は呼び出されたときだけ処理される

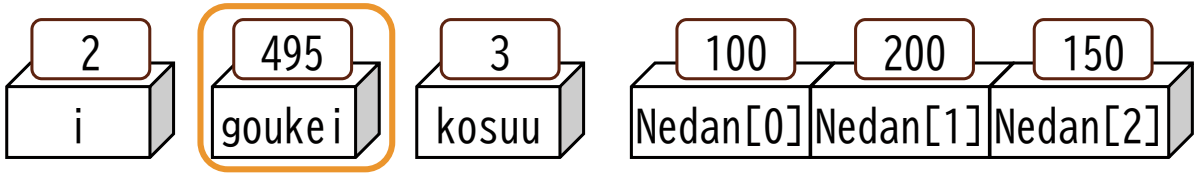
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える



共通テストで使用されるDNCL

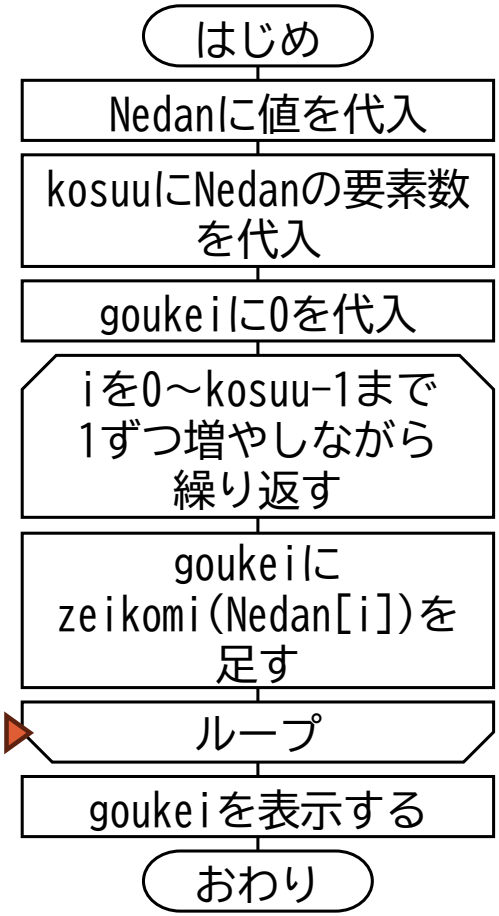
```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```





関数は呼び出されたときだけ処理される

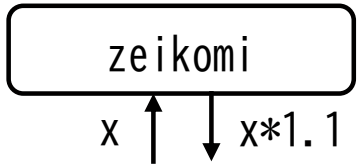
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

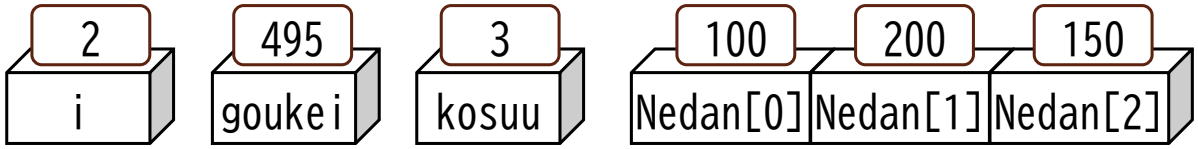


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

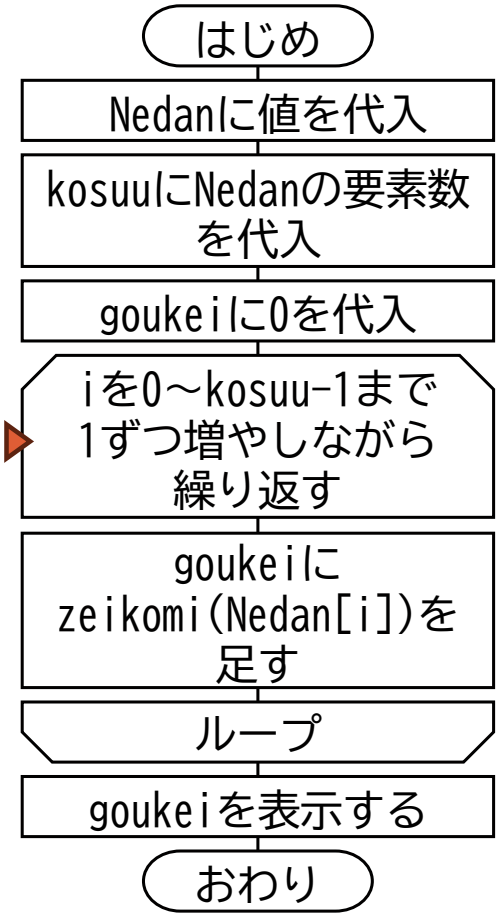
反復処理の終了部分にきたら開始部分に戻る





関数は呼び出されたときだけ処理される

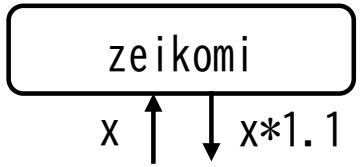
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える

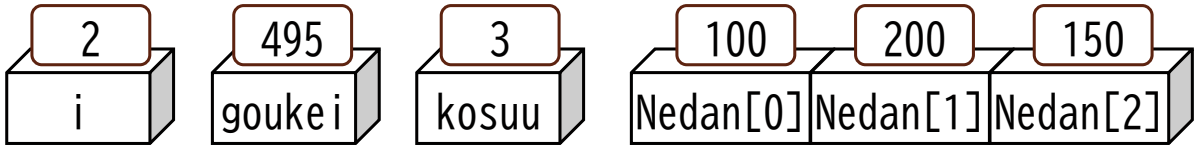


共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

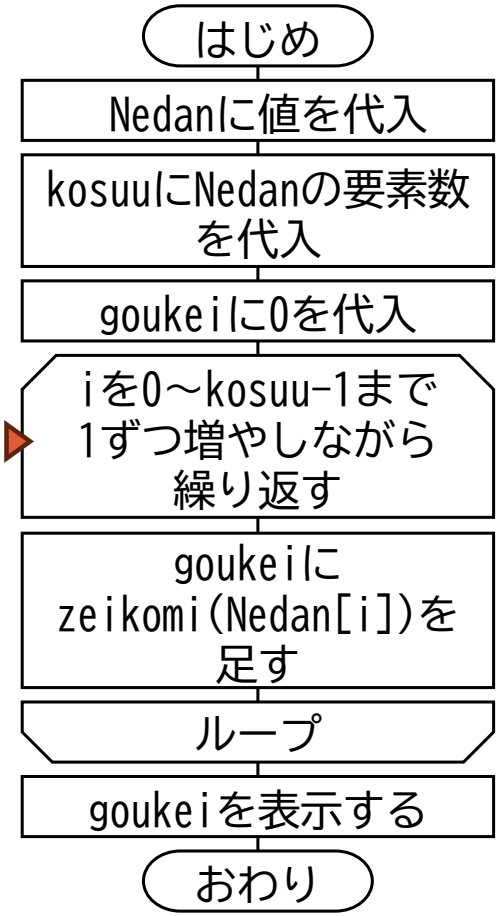
反復処理の終了部分にきたら開始部分に戻る





関数は呼び出されたときだけ処理される

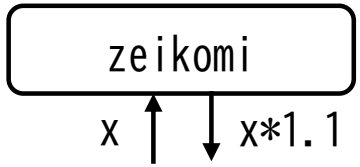
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える



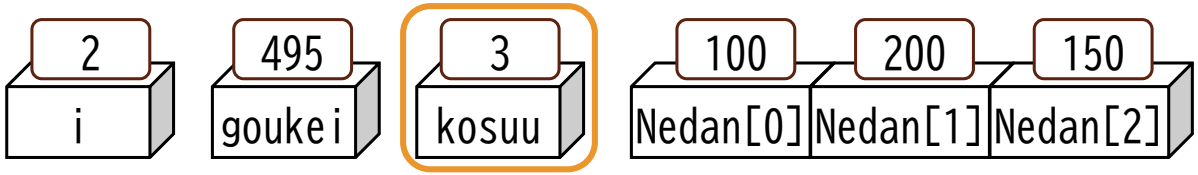
共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

iを0から2まで1ずつ増やす



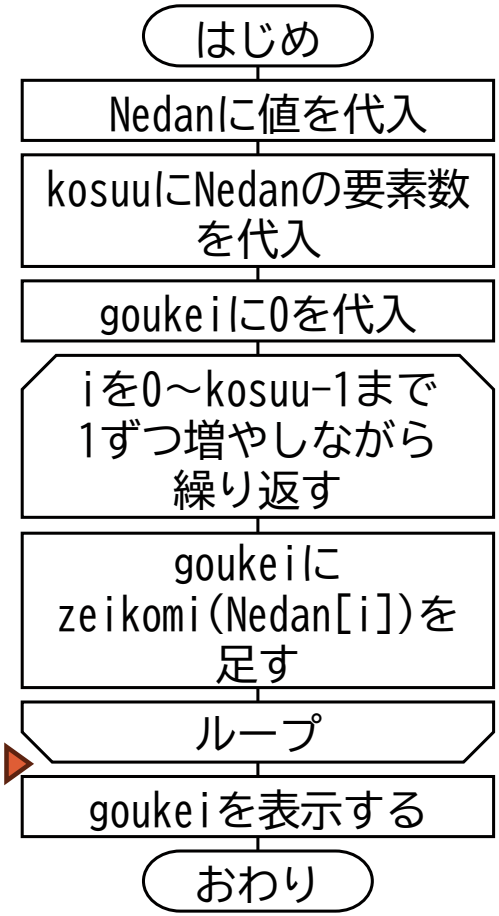
いまのkosuu-1は2



赤字：用語も意味も理解が必要、緑字：意味や概念の理解が必要

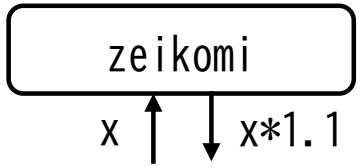
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える



共通テストで使用されるDNCL

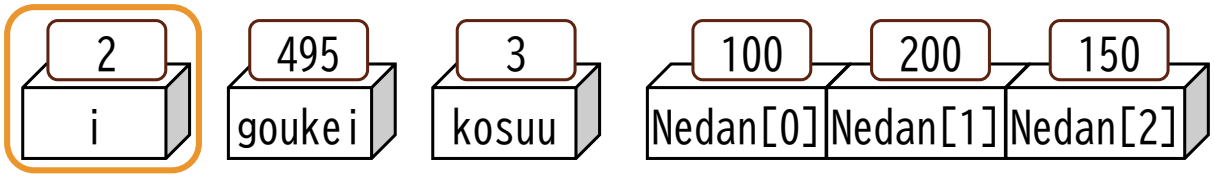
```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



iを0から2まで1ずつ増やす

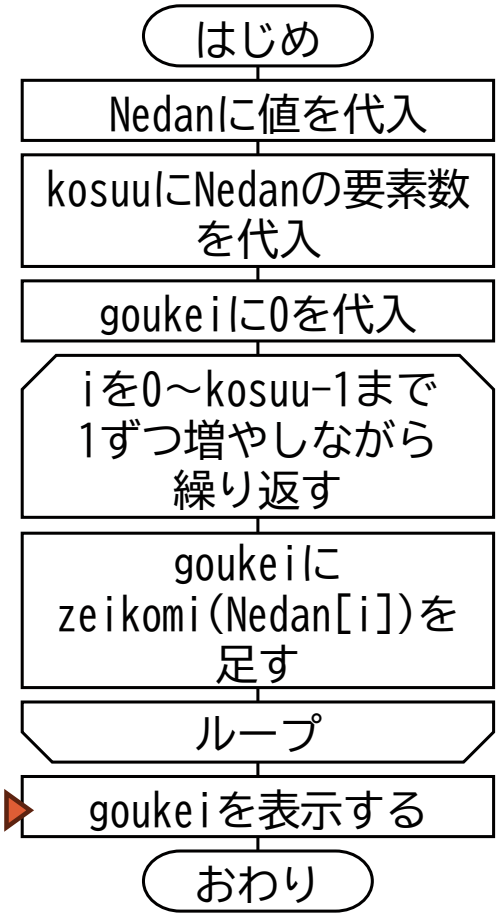
上限に達したので反復処理の次に進む

いまのiは2



関数は呼び出されたときだけ処理される

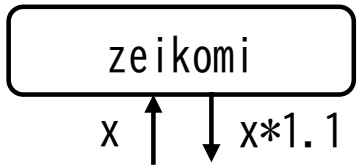
- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える



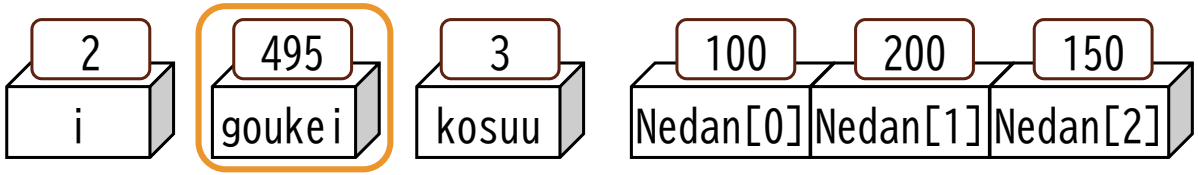
共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
    goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

「495」と表示される

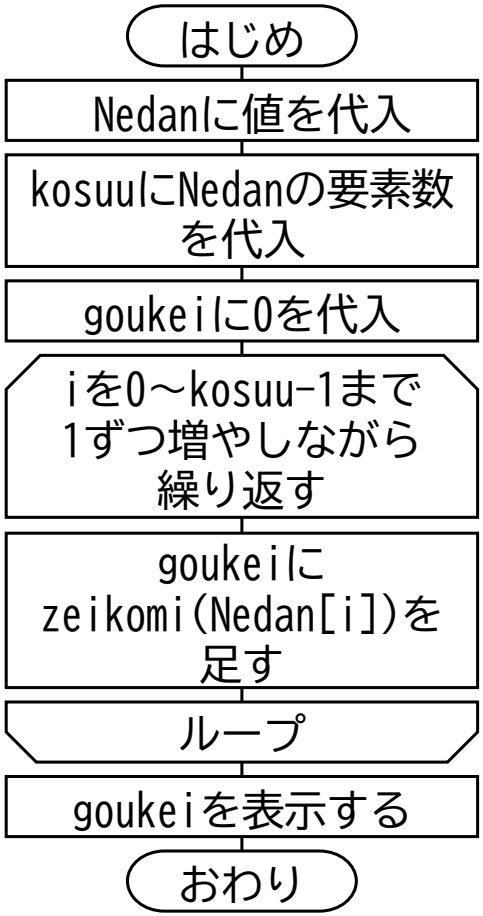


いまのgoukeiは495



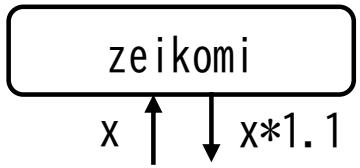
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える



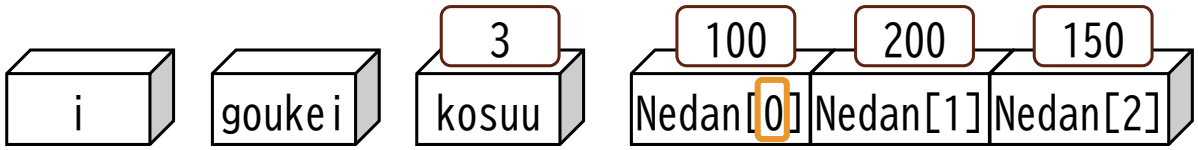
共通テストで使用されるDNCL

```
関数 zeikomi(x) を定義する：
  ┗ 返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
  ┗ goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



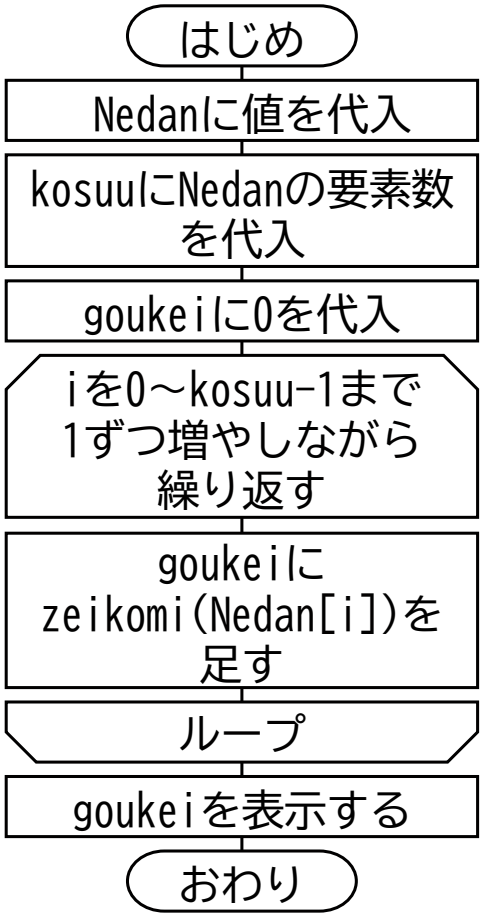
反復処理における注意点

0から始まる場合　　添字も0から始まる



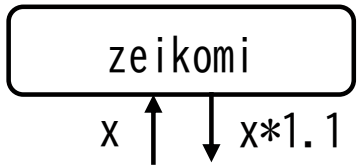
関数は呼び出されたときだけ処理される

- 「＝」は代入の意味　※等しいではない
- 計算をしてから変数に代入する
  - 変数は値に置き換える



共通テストで使用されるDNCL

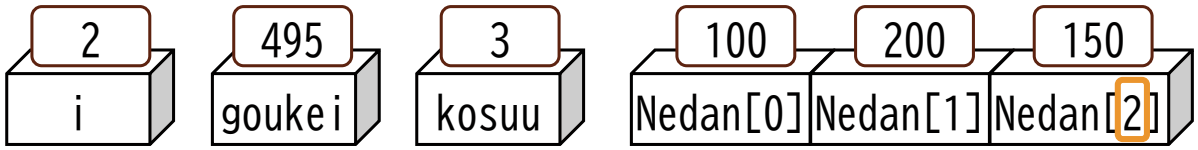
```
関数 zeikomi(x) を定義する：
  ┗ 返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
  ┗ goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```



反復処理における注意点

繰り返し回数は要素数-1

添字の最大値も要素数-1



「プログラミング(関数)」の要点

「プログラミング(関数)」の要点

関数：機能単位で切り出されたプログラム。
プログラムの理解しやすさの向上、再利用可能、修正やテストの負荷軽減といったメリットがある。
関数にインプットされる値を**引数**といい、呼び出し元に戻される値を**戻り値**または返り値という。

```
関数 zeikomi(x) を定義する：
    返す( x * 1.1 )  # 消費税10%

Nedan = [100, 200, 150]
kosuu = 要素数(Nedan)  # 配列の要素数を返す関数
goukei = 0
iを0からkosuu-1まで1ずつ増やしながら繰り返す：
 ┗ goukei = goukei + zeikomi(Nedan[i])
表示する(goukei)
```

- 数字が不規則(ランダム)に並んだ数を**乱数**という
- 乱数という関数は、実行ごとに異なる値を出力する

```
kazu = 乱数()
```

kazuに生成された乱数の値が代入される