情報メディア基盤ユニット

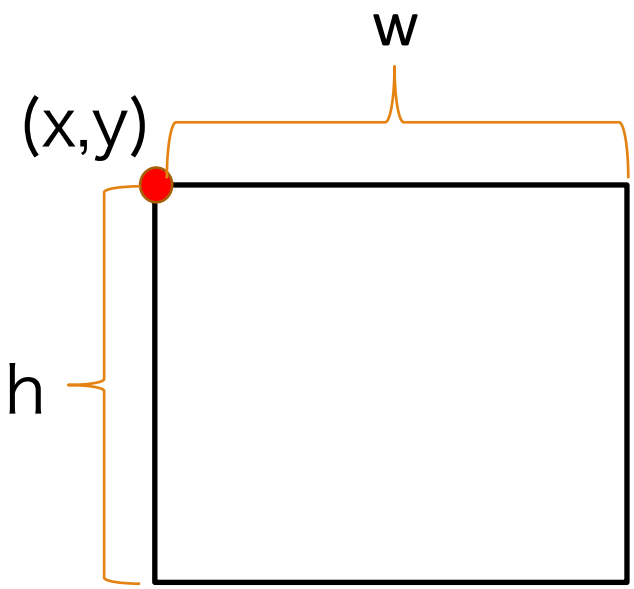
5月17日

繰り返し処理

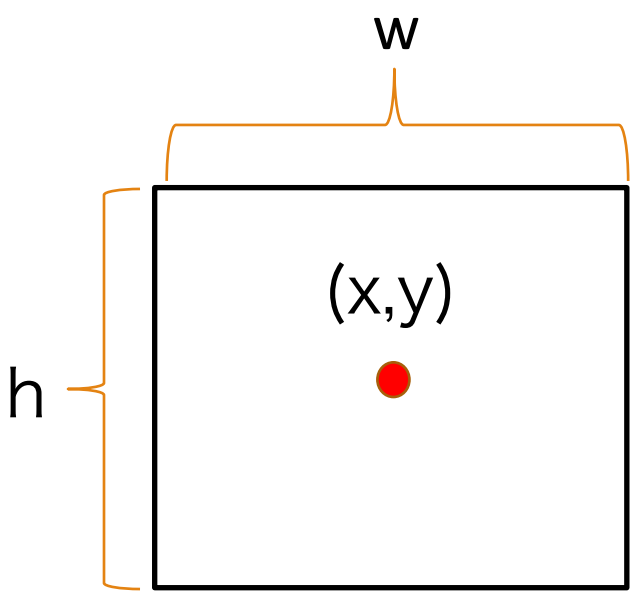
情報メディア学科佐藤尚

rect(x,y,w,h)

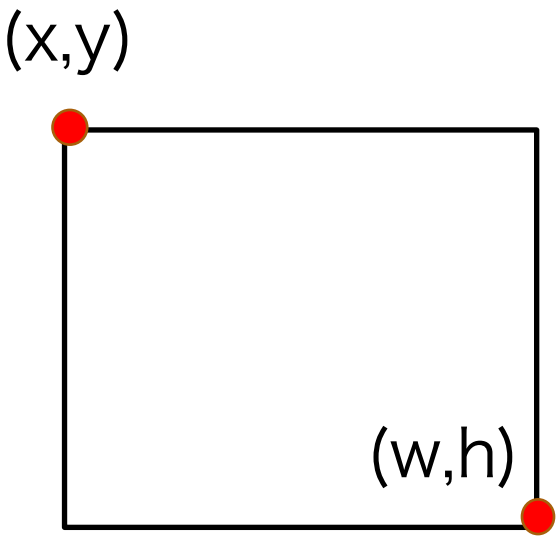
rectMode(CORNER)



rectMode(CENTER)



rectMode(CENTERS)

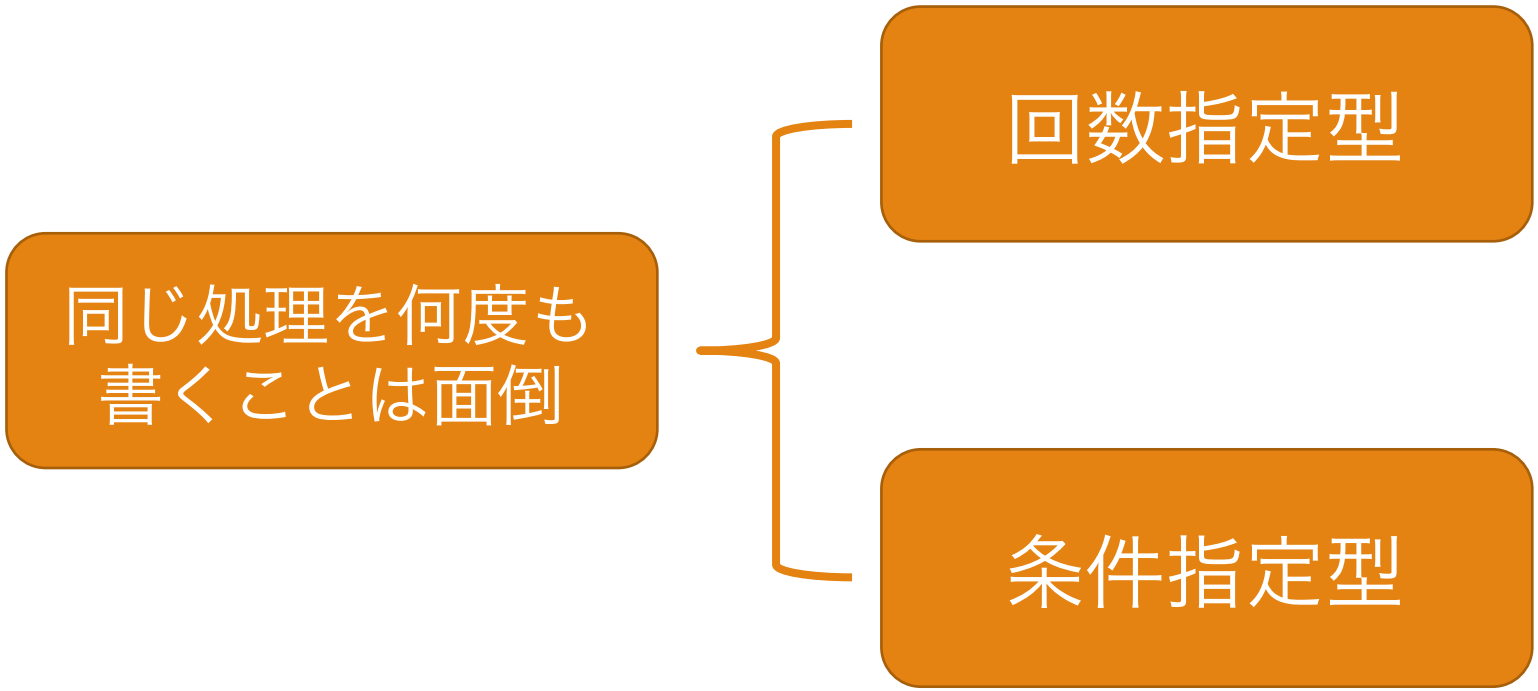


これがデフォルトの動作

プログラムの構成要素

逐次処理	前々回までやっていた、 上から順番に命令を実行
条件分岐処理	前回にやった条件に応じて 実行する命令を選ぶ
繰り返し処理	同じ命令（の塊）を 繰り返し実行
関数	処理の塊に名前をつけて、 その名前で処理を行う

繰り返し処理



繰り返し処理を使うためのこつ

繰り返しされているパターンを探し出す

```
line(25,20,25,180);
line(50,20,50,180);
line(75,20,75,180);
line(100,20,100,180);
line(125,20,125,180);
line(150,20,150,180);
line(175,20,175,180);
line(200,20,200,180);
line(225,20,225,180);
line(250,20,250,180);
```

```
line( 0*25+25,20, 0*25+25,180);
line( 1*25+25,20, 1*25+25,180);
line( 2*25+25,20, 2*25+25,180);
line( 3*25+25,20, 3*25+25,180);
line( 4*25+25,20, 4*25+25,180);
line( 5*25+25,20, 5*25+25,180);
line( 6*25+25,20, 6*25+25,180);
line( 7*25+25,20, 7*25+25,180);
line( 8*25+25,20, 8*25+25,180);
line( 9*25+25,20, 9*25+25,180);
```

少し複雑な回数指定処理

カウンタ変数の利用を
利用して処理を行う

7ページ目から

```
line( 0*25+25,20, 0*25+25,180);
line( 1*25+25,20, 1*25+25,180);
line( 2*25+25,20, 2*25+25,180);
line( 3*25+25,20, 3*25+25,180);
line( 4*25+25,20, 4*25+25,180);
line( 5*25+25,20, 5*25+25,180);
line( 6*25+25,20, 6*25+25,180);
line( 7*25+25,20, 7*25+25,180);
line( 8*25+25,20, 8*25+25,180);
line( 9*25+25,20, 9*25+25,180);
```

```
for(int i=0;i<10;i++){
  line(i*25+25,20,
       i*25+25,180);
}
```

繰り返しパターンの別の見方

```
line(25,20,25,180);
line(50,20,50,180);
line(75,20,75,180);
line(100,20,100,180);
line(125,20,125,180);
line(150,20,150,180);
line(175,20,175,180);
line(200,20,200,180);
line(225,20,225,180);
line(250,20,250,180);
```

```
line(  0+25,20,  0+25,180);
line( 25+25,20, 25+25,180);
line( 50+25,20, 50+25,180);
line( 75+25,20, 75+25,180);
line(100+25,20,100+25,180);
line(125+25,20,125+25,180);
line(150+25,20,150+25,180);
line(175+25,20,175+25,180);
line(200+25,20,200+25,180);
line(225+25,20,225+25,180);
```

繰り返しパターンの別の見方

```
line(  0+25,20,  0+25,180);
line( 25+25,20, 25+25,180);
line( 50+25,20, 50+25,180);
line( 75+25,20, 75+25,180);
line(100+25,20,100+25,180);
line(125+25,20,125+25,180);
line(150+25,20,150+25,180);
line(175+25,20,175+25,180);
line(200+25,20,200+25,180);
line(225+25,20,225+25,180);
```

```
x = 0;
line(x+25,20,x+25,180);
x = x+25;
line(x+25,20,x+25,180);
x = x+25;
line(x+25,20,x+25,180);
x = x+25;
line(x+25,20,x+25,180);
x = x+25;
line(x+25,20,x+25,180);
x = x+25;
```

以下繰り返す

```
line(x+25,20,y+25,180);
x = x+25;
```

繰り返しパターンの別の見方

```
x = 0;
line(x+25,20,x+25,180);
x = x+25;
line(x+25,20,x+25,180);
x = x+25;
line(x+25,20,x+25,180);
x = x+25;
line(x+25,20,x+25,180);
x = x+25;
line(x+25,20,x+25,180);
```

以下繰り返す

```
x = x+25;
line(x+25,20,y+25,180);
```

```
x = 25;
line(x,20,x,180);
x = x+25;
line(x,20,x,180);
x = x+25;
line(x,20,x,180);
x = x+25;
line(x,20,x,180);
x = x+25;
line(x,20,x,25,180);
```

以下繰り返す

```
x = x+25;
line(x,20,x,180);
x = x+25;
```

一番最後はxの値は275になっている。

繰り返しパターンの別の見方

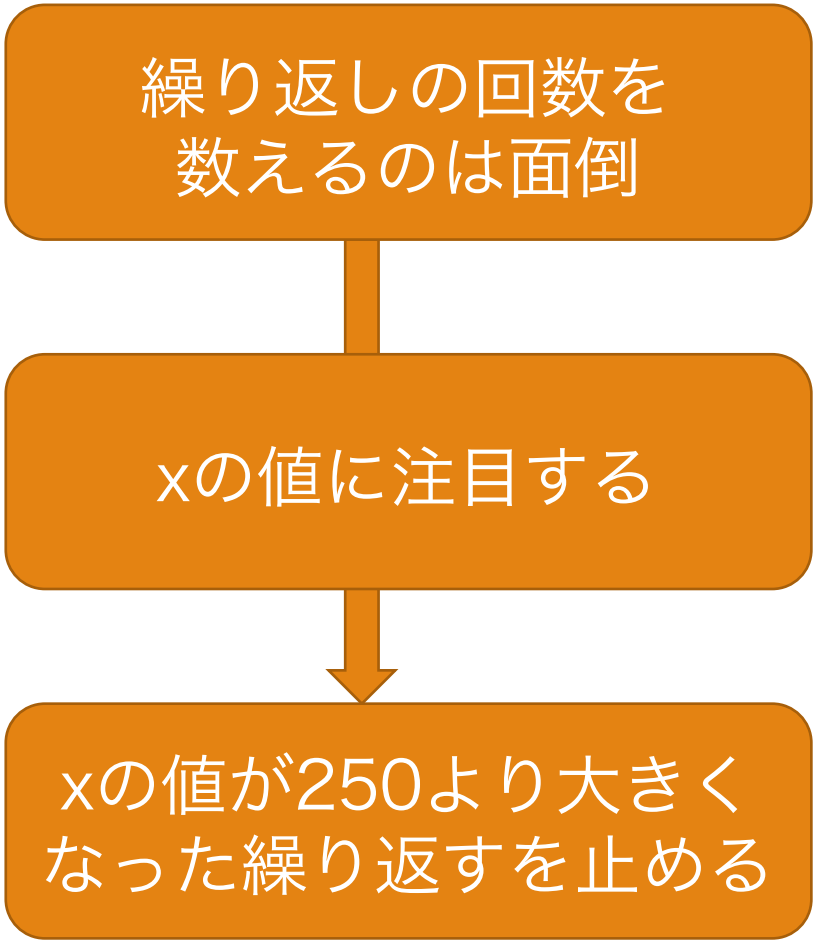
```
x = 0;
for(int i=0;i<10;i++){
  line(x+25,20,x+25,180);
  x = x+25;
}
```

```
x = 25;
for(int i=0;i<10;i++){
  line(x,20,x,180);
  x = x+25;
}
```

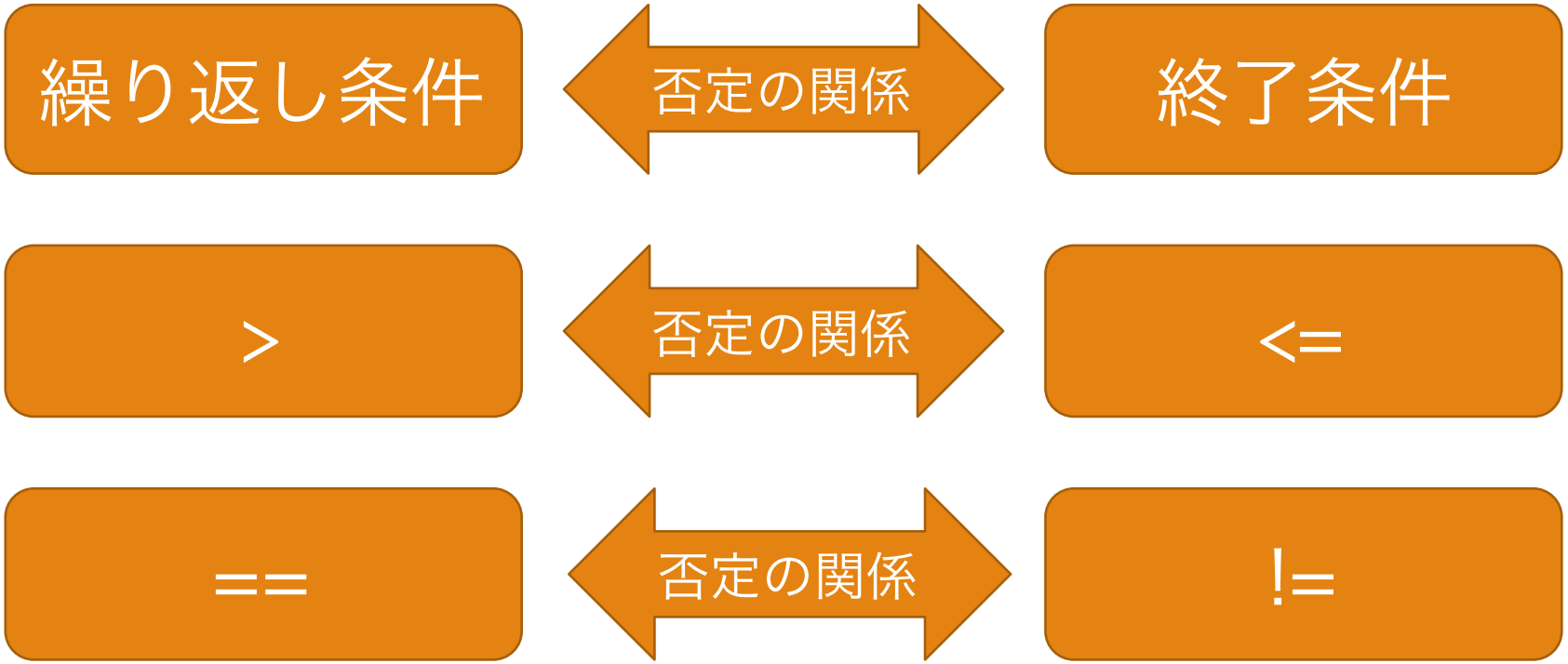
繰り返しパターンの別の見方

```
x = 25;
line(x,20,x,180);
x = x+25;
line(x,20,x,180);
x = x+25;
line(x,20,x,180);
x = x+25;
line(x,20,x,180);
x = x+25;
line(x,20,x,25,180);
以下繰り返す
x = x+25;
line(x,20,x,180);
x = x+25;
```

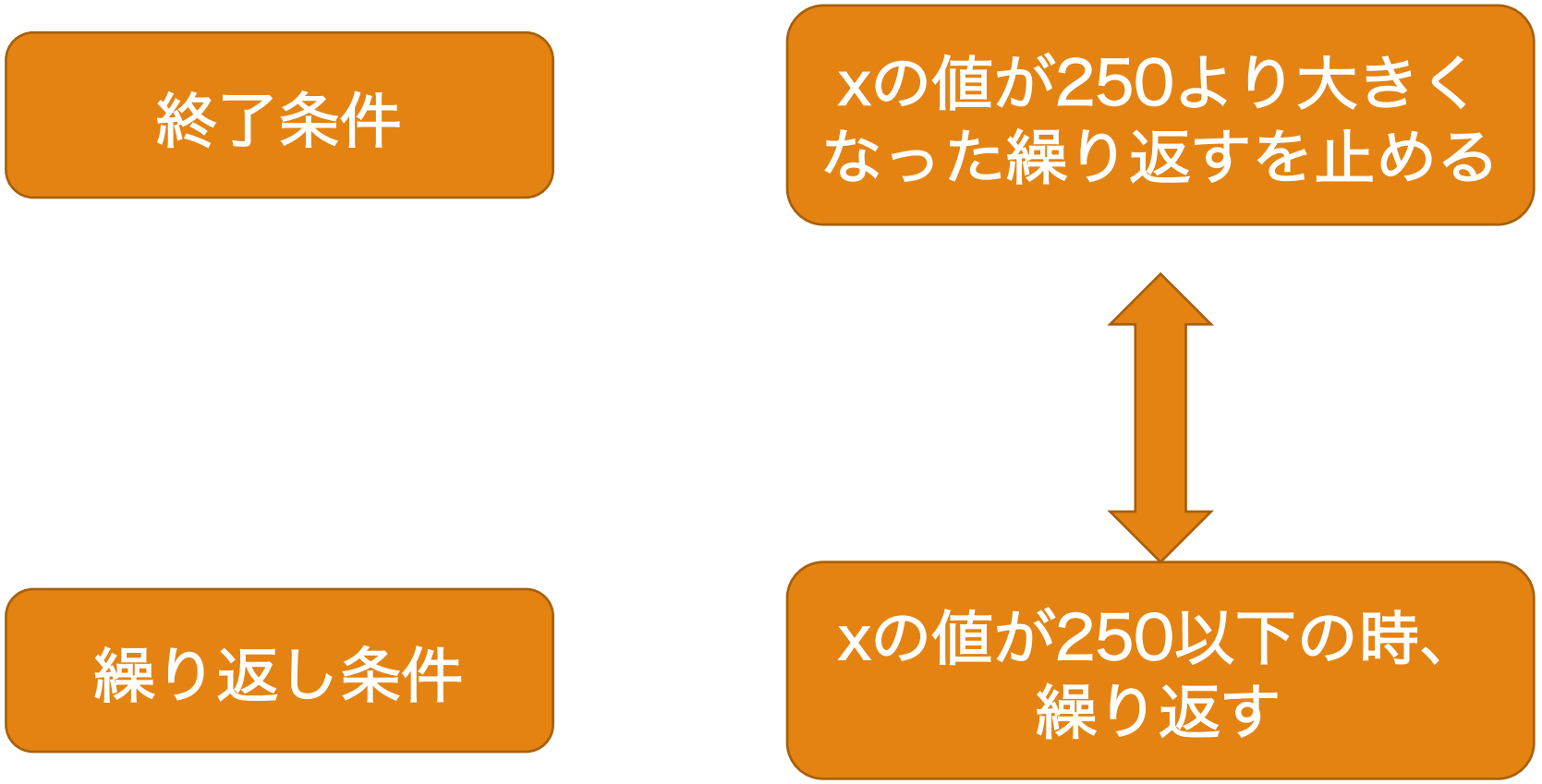
一番最後はxの値は275になっている。



繰り返し条件と終了条件

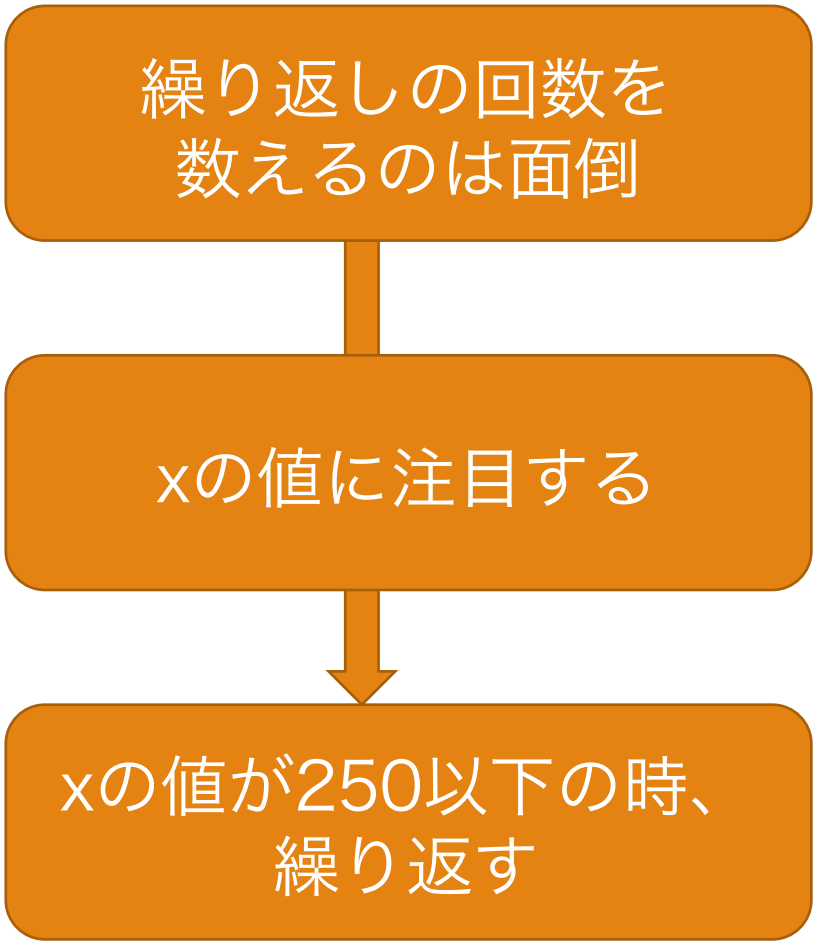


などなど


終了条件
xの値が250より大きくなった繰り返すを止める
繰り返し条件
xの値が250以下の時、繰り返す

繰り返しパターンの別の見方

```
x = 25;
line(x,20,x,180);
x = x+25;
line(x,20,x,180);
x = x+25;
line(x,20,x,180);
x = x+25;
line(x,20,x,180);
x = x+25;
line(x,20,x,25,180);
以下繰り返す
x = x+25;
line(x,20,x,180);
x = x+25;
```



条件指定の繰り返し処理

while命令を使う

```
while(繰り返し条件){
    繰り返し処理内容
}
```

```
x = 25;
while(x <= 250){
  line(x,20,x,180);
  x = x+25;
}
```

同じ動作をするプルグラム3種

```
size(300,200);
background(255);
stroke(0);

line(25,20,25,180);
line(50,20,50,180);
line(75,20,75,180);
line(100,20,100,180);
line(125,20,125,180);
line(150,20,150,180);
line(175,20,175,180);
line(200,20,200,180);
line(225,20,225,180);
line(250,20,250,180);
```

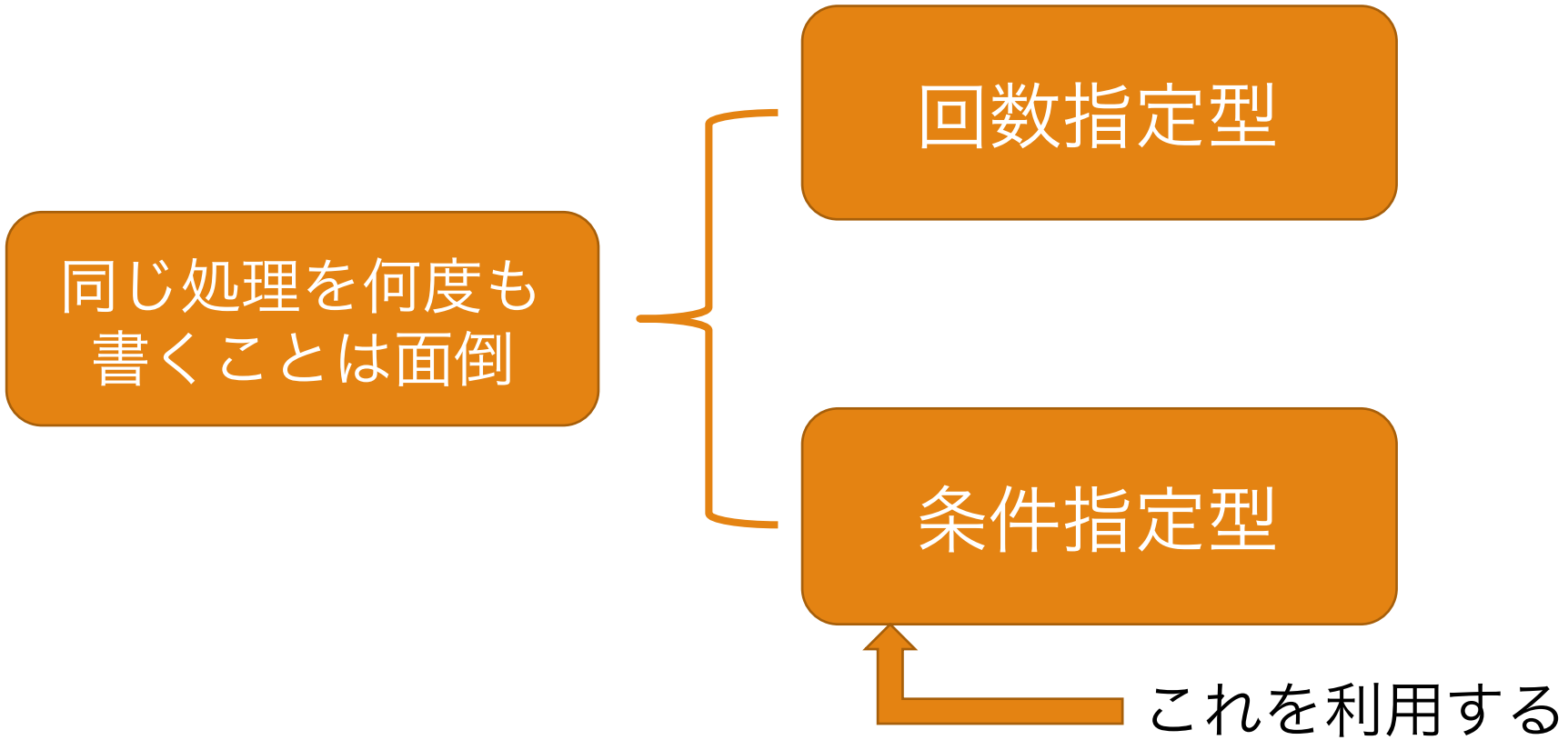
```
size(300,200);
background(255);
stroke(0);

for(int i=0;i<10;i++){
  line(25*i+25,20,
       25*i+25,180);
}
```

```
int x;
size(300,200);
background(255);
stroke(0);

x = 25;
while(x <= 250){
  line(x,20,x,180);
  x = x+25;
}
```

プログラム作成時には繰り返し回数がわからない場合



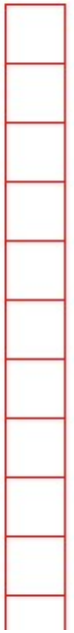
例えば、

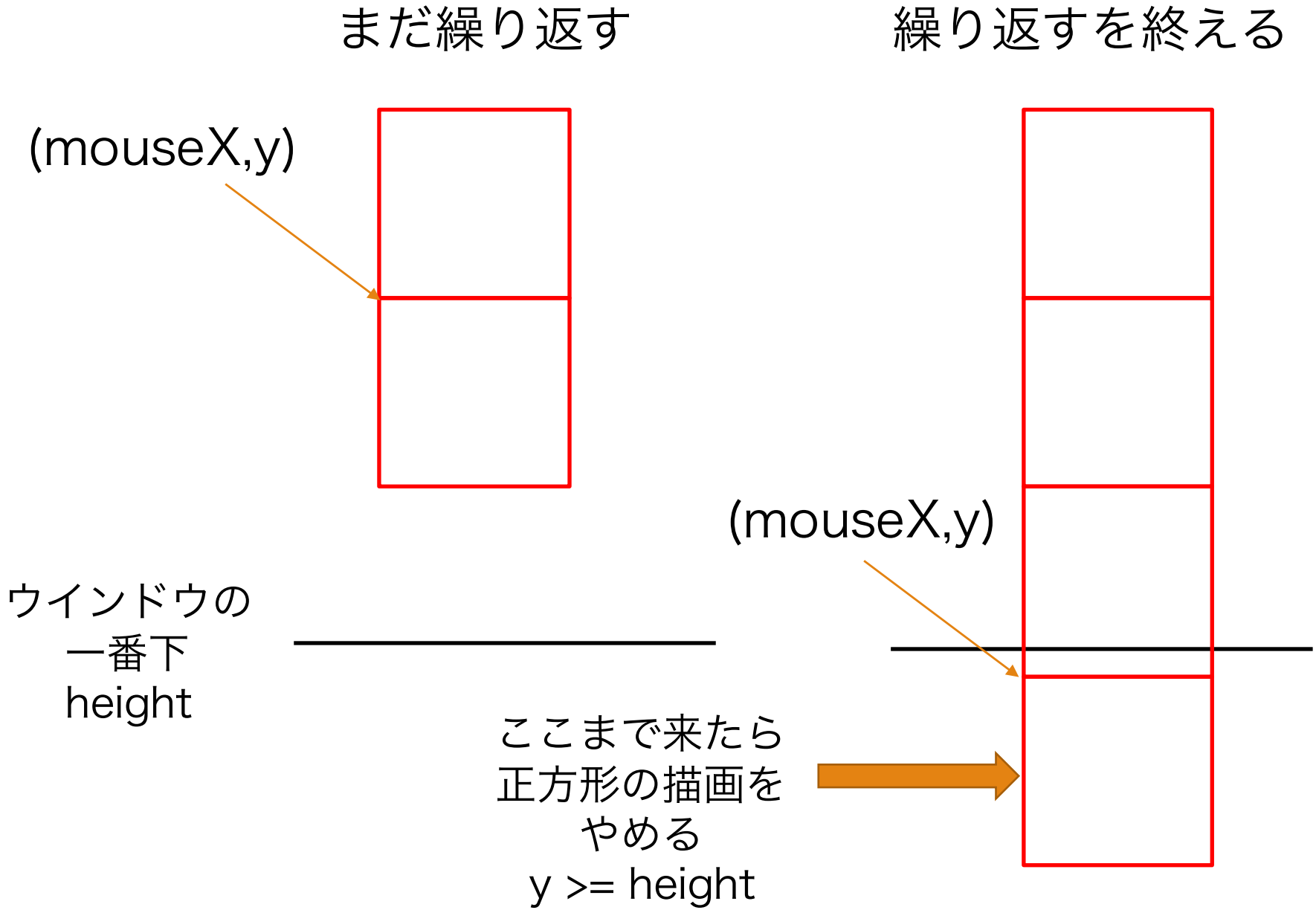
現在のマウスの位置から、下の方に向けて正方形を描く。ただし、塗りつぶしをしません。一辺の長さは30とする。

mouseYの値から正方形を描き始めて、描く**正方形**のY座標の値を30増やすことを繰り返しながら、**正方形**を描いていく。

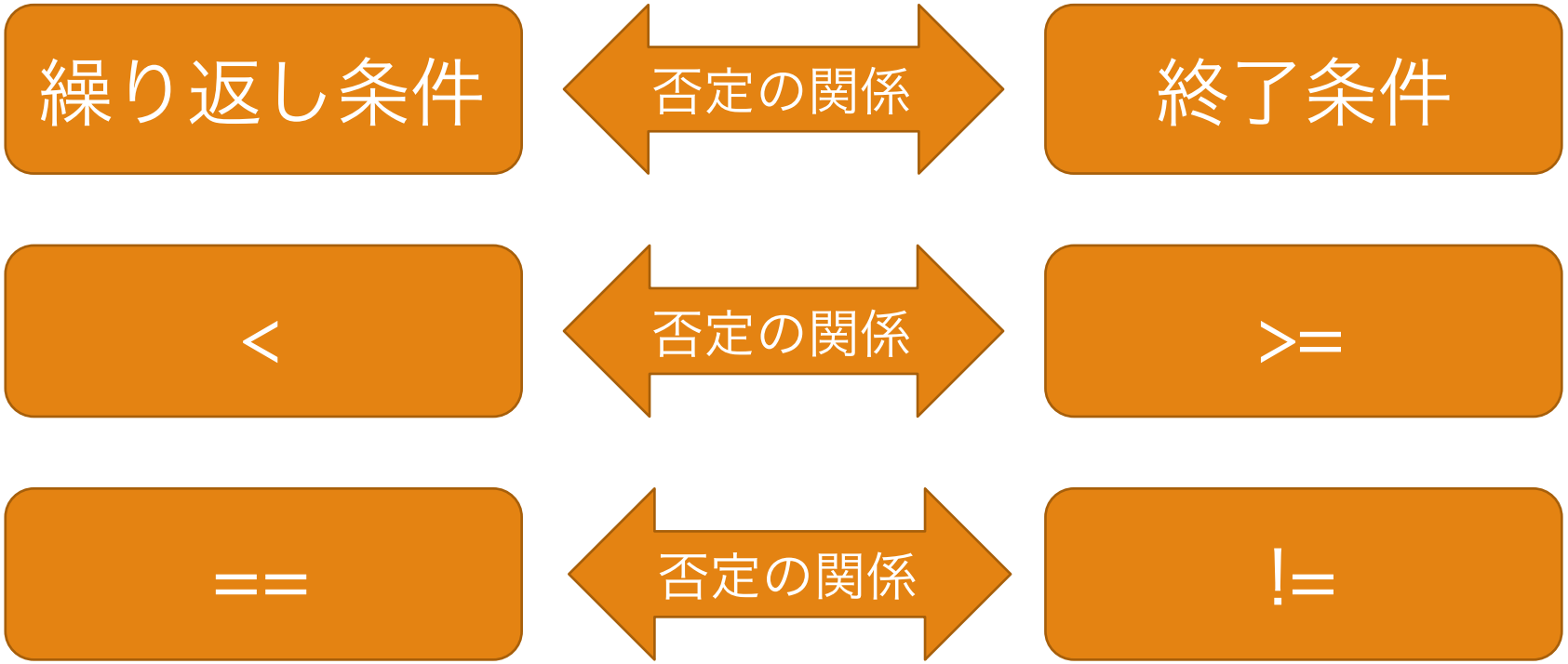
1. 描く正方形のX座標の値はmouseX
2. 描く正方形のY座標の値を変数yに保存する
3. 変数yの値は最初はmouseY
4. 正方形を描く
5. yの値を30増やす
6. 4と5を繰り返す

何回繰り返すの？




まだ繰り返す
繰り返すを終える
(mouseX,y)
(mouseX,y)
ウインドウの
一番下
height
ここまで来たら
正方形の描画を
やめる
y >= height

繰り返し条件と終了条件



などなど

```
int y;

voidsetup(){
  size(200,400);
}

void draw(){
  background(255);
  stroke(255,10,10);
  noFill();
  y = mouseY;
  while(y < height){
   rect(mouseX,y,30,30);
   y = y + 30;
  }
}
```

```
int y;

voidsetup(){
  size(200,400);
}

void draw(){
  background(255);
  stroke(255,10,10);
  noFill();
  y = mouseY;
  while(y < height){
   rect(mouseX,y,30,30);
   y += 30;
  }
}
```

複合代入演算子とインクリメント・デクリメント演算子

ある変数の値を増やしたり、減らしたりする処理を書くことが多いので簡易的な書き方が用意されている。

```
x = x + 25; //意味は変数xの値を25増やす
x += 25; //意味は変数xの値を25増やす
x -= 25; //意味は変数xの値を25減らす
x --; //意味は変数xの値を1だけ減らす
--x;
x++; //意味は変数xの値を1だけ増やす
++x;
```

割り算/に注意

float：実数
- 1.0/2.0 -> 0.5
- 9.0/2.0 -> 4.5
- 9.0/2 -> 4.5
- 9/2.0 -> 4.5

int：整数
- 1/2 -> 0
- 9/2 -> 4
- 商を求める
- %をつかうと余りが求まる
- 1 % 2 -> 1
- 4 % 2 -> 0
- 偶数か奇数の判定：2で割ったあまりが0なら偶数、1なら奇数

A	B	A/Bの結果
int	int	int
int	float	float
float	Int	float
float	float	float

演算子の優先順位

演算子の優先順位 → 数式のどの部分から先に計算すべきかの規則

1. 括弧の中
2. かけ算と割り算
3. 足し算と引き算

優先順位	演算子	説明
1	[] . ++ --	配列要素、メンバへのアクセスなど
2	! ~ - +	前置の単項演算子など
3	* / %	乗法、除法、剰余
4	+ -	加法、減法
5	<< >>	ビット単位のシフト
6	< <= > >=	大小比較
7	== !=	等価、非等価比較
8	&	ビット単位の論理積
9	^	ビット単位の排他的論理和
10	|	ビット単位の論理和
11	&&	かつ
12	||	または
13	= += -= *= /= %= &= |= ^= <<= >>=	代入、複合代入演算子

演算子の優先順位を考慮すると

mouseX + 10 < width/2

(mouseX + 10) < (width/2)

width/3 < mouseX && mouseX < 2*width/3

((width/3)< mouseX) && (mouseX < (2*width/3))

演算の順番に自信が無いときは

積極的に括弧（）を使う

```
width/3 < mouseX && mouseX < 2*width/3
```

↓

```
(width/3 < mouseX) && (mouseX < 2*width/3)
```

色相、彩度、明度による色指定

コンピュータから見るとRGBによる指定が自然

人間から見るとRGBによる指定は不慣れ

色選択

H:色相（Hue）

S:彩度(Saturation)

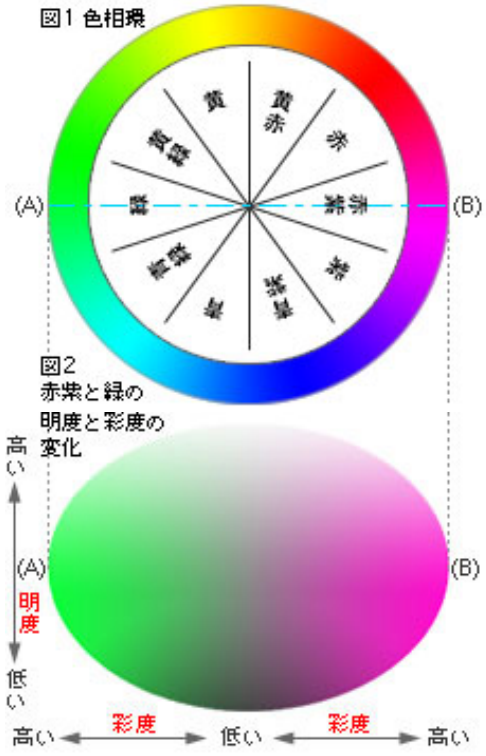
B:明度(Brightness)

色相、彩度、明度

色相(Hue)
- 赤、黄、青などの「色合い」
- 0～359

彩度(Chroma or Saturation)
- 「あざやかさ」の度合い
- 0～99

明度(Value or Brightness)
- 「明るさ」の度合い
- 0～99



colorMode(RGB) <-デフォルトの色指定

- RGBの値を0～255
- colorMode(RGB,100) <- 0～100の数値で指定

colorMode(HSB,359,99,99);

- H:0～359
- S,B：0～99
- colorSelectorの数値
- colorMode(HSB) <- HSBの値を0～255で指定

色を表すデータ型

color型

```
color c1 = color(10,100,255);
color c2 = color(10,100,255,90);
```

授業時に配布した資料

http://www.sato-lab.jp/imfu/index.html

においてあります。