くりかえし. (python)

k^2 を表示.

```
for k in range(1,11):
    print(k*k)
```

k は 1, 2, ..., 10

次がインデントで block になってる.

インデントで, { } のかわり.

1
4
9
⋮
100

← お手軽. google colaboratory (webより使う)

C言語

```
#include <stdio.h>
int main(){
    int k;
    for (k=1; k<11; k++){
        printf("%d\n", k*k);
    }
}
```

関数 (procedure, subroutine, ...) ← 数. "関数" に似てるけどちがうもの.

処理をひとまとまりにしたもの.

$\sum_{k=a}^{b} k$ を計算する関数.

python.

```
def mySum(from2, to):
    S=0;
    for k in range(from2, to+1):
        S=S+k;
    return S
print(Sum(1,100))
```

変数.

type(S)
Sのデータの型を表示.

k=2;

print(k) 2と表示.

引数の型

C言語.

```
int sum(int from, int to){
    int S;
    int k;
    for(k=from; k<=to; k++){
        S=S+k;
    }
    return S;
}
int main(){
    printf("%d\n", sum(1,100));
}
```

1-sum.c

関数中で使う局所変数を宣言.

10進で表示 dはdecimal.

① 関数を使うと処理を分割できる.

配列(array)

C言語. int x[3]; 長さ3の配列. つまり.

x[0], x[1], x[2]　ココの変数ができる。
（ベクトルみたいなもの）

内積計算の関数　2-inner.c

a,bの内積　　長さ

```
int inner(int a[], int b[], int n){
    int s;
    int k;
    s=0;
    for(k=0; k<n; k++){
        s=s+a[k]*b[k];
    }
    return s;
}
int main(){
    int x[3]={1,2,3};
    int y[3]={5,4,3};
    int z;
    z=inner(x,y,3);
    printf("%d\n",z);
}
```

宣言時しか使えない。

```
x[0]=1;
x[1]=2;
x[2]=3;
```

と同じこと。

python版

```
def inner(a,b,n):
    s=0
    for k in range(0,n):
        s=s+a[k]*b[k]
    return s
```

n=len(a)

```
x=[1,2,3]
y=[5,4,3]
z=inner(x,y,3)
print(z)
```

ベクトルの和.　　3-vsum.c

Cでは、配列を戻り値にはできない.　(やまには malloc.等を使う. 中級)

ベクトル a+b を cに代入.

関数値はなし.

```
void vsum(int a[], int b[], int c[], int n){
    int k;
    for (k=0; k<n; k++) {
        c[k]=a[k]+b[k];
    }
}
int main() {
    int x[3]={1,2,3};
    int y[3]={5,4,3};
    int z[3];
    vsum(x,y,z,3);
    for(k=0; k<3; k++) { printf("%d ",z[k]);}
    printf("\n");
}
```

int k;

練習　配列を表示する関数をつくる.

python版

```
def vsum(a,b):
    n=len(a)
    c=list(range(0,n))
    for k in range(0,n):
        c[k]=a[k]+b[k]
    return c
a=[1,2,3]; b=[5,4,3]; z=vsum(a,b)
print(z)
```

◎ python. 便利なライブラリ

線を描く. lines.py

```
import matplotlib.pyplot as plt
import numpy as np.
x = (0, 1, 1, 0, 0, 1)
y = (0, 0, 1, 1, 0, 1)
plt.plot(x, y)
plt.show()
```

中級編 (object 指向プログラム)

インスタンス変数

(正体は、メモリのどこかにまとまった領域へのポインター. たとえば構造体などへのポインター)

例. C言語.

Java "点"

```
struct point {
    int x;
    int y;
};
```

色付きの点

```
struct cpoint {
    int x;
    int y;
    int color;
};
```

```
struct point *p;
p = (struct point *)
    malloc(sizeof(struct point));
```

4×2=8 byte.

```
struct cpoint *q;
q = (struct cpoint *)
    malloc(sizeof(struct cpoint));
```

4×3=12 byte.

Java言語.

```
public class point extends Object {
    int x;
    int y;

    public static void main(String[] args) {
        point p = new point();
    }
}
```

Object が superclass

method のていぎ ①

② → point p ... ; ← インスタンスの生成

new point() : malloc に対応

p : クラス point の instance 変数

```
public class cpoint extends point {
    int color;

    public static void main(String[] args) {
        cpoint q = new cpoint();
    }
}
```

④ → (method の位置)

③ → cpoint q ... ; ← 実体は

int x
int y
int c..

int x, int y : point より

q : クラス cpoint の instance 変数

method ← 関数みたいなもの

①

```
public void output() {
    int p, q;
    p = this.x;
    q = this.y;
    System.out.print("x=" + p + " y=" + q
                                   + "\n");
}
```

② p.output()

p を this として method output をよびだす

- . はいろんな意味あり(多義的)
  C言語では . は構造体のメンバーの意味のみ

object p にメッセージ output をおくる。

③ q.output() とすると何がおきるか？

class cpoint には method output がないので、super class の method をよびにいく。class point の method output がよばれる。つまり x=0, y=0

(c は表示されない)

④

```
public void output() {
    super.output();
    System.out.print("c=" + colon);
}
```

← class point の output をよぶ。

this.colon

サンプルの cpoint.java

コンパイル　javac cpoint.java

実行　java cpoint ↵

クラス名＝ファイル名 がルール。

サンプルでは point は point2 になる。

---

python.

インスタンス.method

インスタンス.メンバー

モジュール.メンバー関数

モジュール.モジュール変数