C言語講習会

#7 関数

1. 関数とは

これまで学んできた 条件分岐 や 繰り返し処理 、continue文、break文を使うことで、
ある程度のプログラムを書くことができるようになりました。

これからは上の基本的な処理に加えて、さらに高度なプログラムを簡潔に記述する
方法を学んでいきます。

1. 関数とは

(復習) 数学での関数とは

$$f(x) = 2x + 3$$

入力

$$f(2) = 2 \cdot 2 + 3 = 7$$

入力　出力

関数とは

変数x, yについて、xの値が決まるとyの値がただ一つに定まるとき、yをxの関数という。

1. 関数とは

(復習) 数学での関数とは

$$f(x) = 2x + 3$$

$f(3) = 9$

$f(3) = 12$ ~~12~~

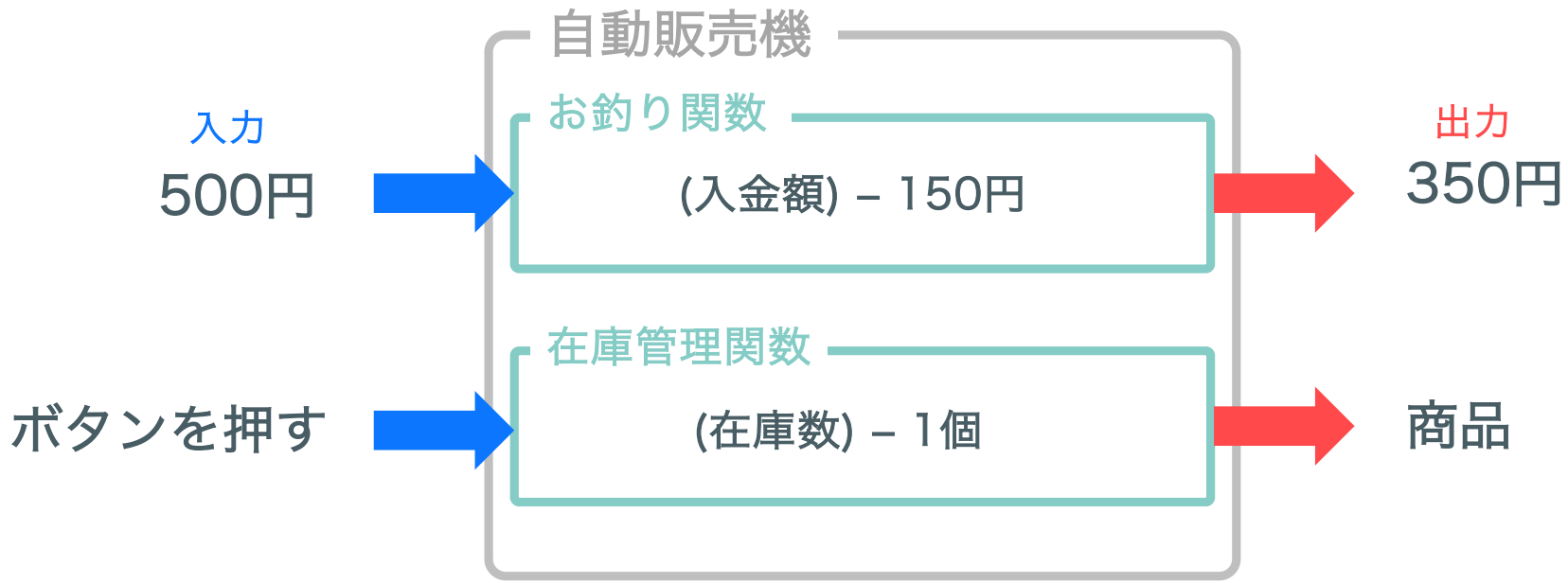
同じ入力に対して
異なる出力にはならない

関数とは

変数x, yについて、xの値が決まるとyの値がただ一つに定まるとき、yをxの関数という。

1. 関数とは

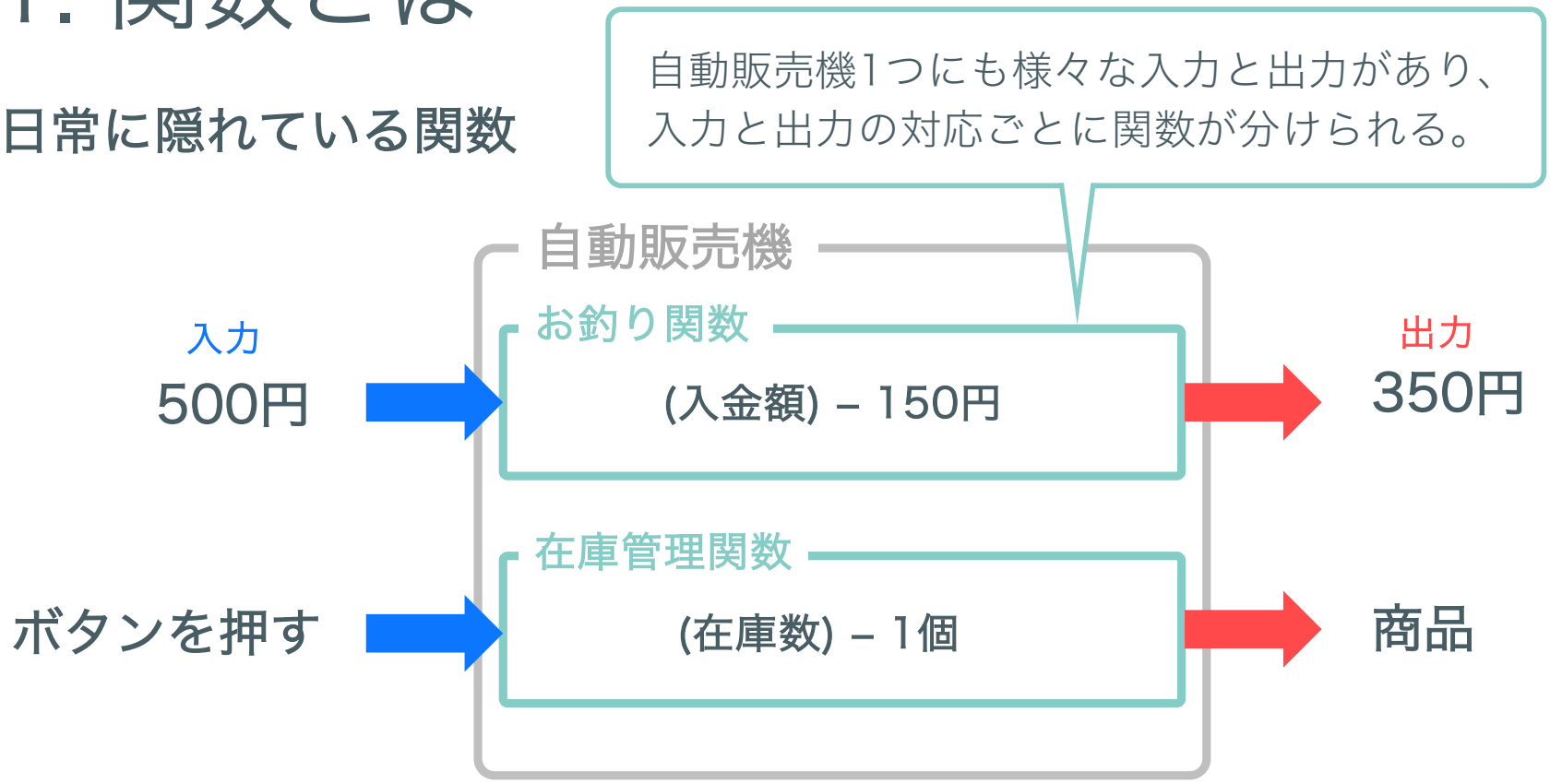
日常に隠れている関数



図：処理機構を簡略した自動販売機

1. 関数とは

日常に隠れている関数



図：処理機構を簡略した自動販売機

1. 関数とは

C言語での関数

```
戻り値の型　関数名( 引数 ){
    処理の内容;
    return　値;
}
```

1. 関数とは

「~~いんすう~~」ではなく「ひきすう」

C言語での関数

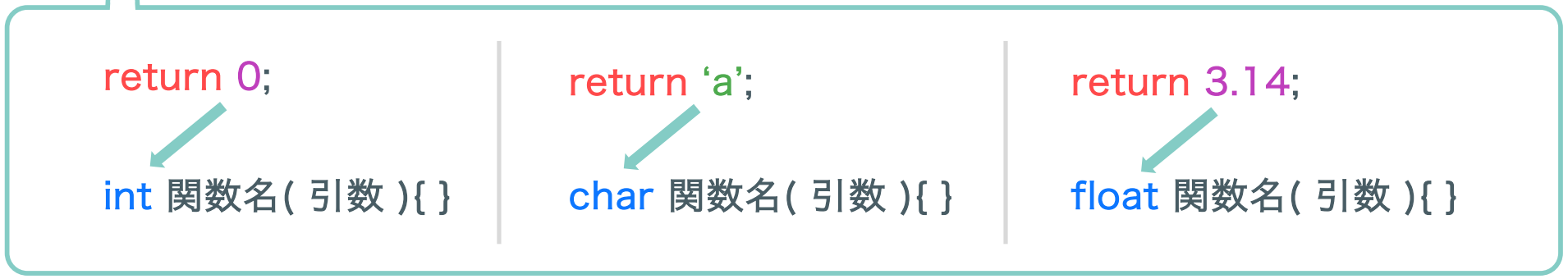
```
戻り値の型　関数名( 引数 ){
    処理の内容;
    return　値;
}
```

インデント

1. 関数とは

C言語での関数

```
戻り値の型　関数名( 引数 ){
    処理の内容;
    return　値;
}
```



1. 関数とは

```c
#include <stdio.h>

int sum(int a, int b){
    int c = a + b;
    return c;
}
```

```c
int main(void){
    int x = 10;
    int y = 5;
    int z;

    z = sum(x, y);

    printf("%d + %d = %d¥n", x, y, z);
    return 0;
}
```

(実行結果)

10 + 5 = 15

1. 関数とは

#include <stdio.h>

main関数

プログラムを実行すると、
まず初めにmain関数から実行される

```
}
```

(実行結果)
10 + 5 = 15

```
int main(void){
    int x = 10;
    int y = 5;
    int z;

    z = sum(x, y);

    printf("%d + %d = %d¥n", x, y, z);
    return 0;
}
```

1. 関数とは

```
#include <stdio.h>

int sum(int a, int b){
}

int main(void){
    int x = 10;
    int y = 5;
    int z;

    z = sum(x, y);

    printf("%d + %d = %d¥n", x, y, z);
    return 0;
}
```

変数の宣言

int型の変数 x, y, zを宣言する

(実行結果)

10 + 5 = 15

1. 関数とは

```c
#include <stdio.h>

int sum(int a, int b){
    int c = a + b;
    return c;
}

int main(void){
    int x = 10;
    int y = 5;
    int z;

    z = sum(x, y);

    printf("%d + %d = %d¥n", x, y, z);
    return 0;
}
```

関数の呼び出し

```
関数名( 引数 );
```

で関数を呼び出すことができる

1. 関数とは

```
#include <stdio.h>

int sum(int a, int b){
    int c = a + b;
    return c;
}
```

sum(x, y); が実行されると、

sum(int a , int b);

引数のaとbに、それぞれxとyが代入される

つまり, int a = x = 10; int b = y = 5;

```
    int z;

    z = sum(x, y);

    printf("%d + %d = %d¥n", x, y, z);
    return 0;
}
```

(実行結果)

10 + 5 = 15

1. 関数とは

```
#include <stdio.h>

int sum(int a, int b){
    int c = a + b;
    return c;
}

int main(void){
    z = sum(x, y);

    printf("%d + %d = %d¥n", x, y, z);
    return 0;
}
```

今, `int a = x = 10;` `int b = y = 5;` より、

```
int c = a + b;
```

は、`c = 15;` となる

(実行結果)

10 + 5 = 15

1. 関数とは

```
#include <stdio.h>

int sum(int a, int b){
    int c = a + b;
    return c;
}
```

(実行結果)

10 + 5 = 15

変数aやbは宣言しなくてもいいの？

```
int sum( int a, int b ){ }
```

関数を定義する際に、引数の部分で変数aとbを既に宣言しているため、新たに別途で

```
int sum( int a, int b ){
    int a, b;
    int c = a + b;
    return c;
}
```

とする必要はない

, x, y, z);

}

1. 関数とは

```
#include <stdio.h>

int sum(int a, int b){
    int c = a + b;
    return c;
}

int main(void){
    int x = 10;
    int y = 5;
    int z;

    z = sum(x, y);

    printf("%d + %d = %d¥n", x, y, z);
    return 0;
}
```

return c;

で c = 15; を呼び出し元に返す (同時に関数sumを終了)

(実行結果)

10 + 5 = 15

1. 関数とは

```c
#include <stdio.h>

int sum(int a, int b){
    int c = a + b;
    return c;
}
```

```c
int main(void){
    int x = 10;
    int y = 5;
    int z;

    z = sum(x, y);

    printf("%d + %d = %d¥n", x, y, z);
    return 0;
}
```

(実行結果)

10 + 5 = 15

1. 関数とは

```c
#include <stdio.h>

int sum(int a, int b){
    int c = a + b;
    return c;
}
```

```c
int main(void){
    int x = 10;
    int y = 5;
    int z = sum(x, y);
    printf("%d + %d = %d¥n", x, y, z);
    return 0;
}
```

return 0;

でmain関数を終了する。
(つまりプログラムを終了する)

(実行結果)

10 + 5 = 15

2. main関数の return 0; は何をしてるの?

main関数のreturn先

main関数で return 0; をすると「プログラムが正常に終了した」というシグナルをOS側に返します。
また、main関数で return 1; にすると「異常終了」のシグナルをOS側に返します。

```
$ cat hoge.c
  int main(void){return 0;}

$ gcc hoge.c -o hoge

$ ./hoge && echo “OK”
  OK
```

```
$ cat hoge.c
  int main(void){return 1;}

$ gcc hoge.c -o hoge

$ ./hoge && echo “OK”
```

← OKと表示されない

2. main関数の return 0; は何をしてるの?

main関数のreturn先

main関数で return 0; をすると「プログラムが正常に終了した」というシグナルをOS側に返します。

また、main関数で return 1; にすると「異常終了」のシグナルをOS側に返します。

Unix/Linux コマンドでの && の意味

$ ここが正常に終了したら && ここのコマンドを実行する

```
$ cat h
  int m                                    n 1;}

$ gcc hog .c -o hoge

$ ./hoge && echo “OK”
  OK
```

```
$ gcc hoge.c -o hoge

$ ./hoge && echo “OK”
```

← OKと表示されない

参考文献

・大川内隆郎, 大原竜男, かんたん C言語 [改訂2版], 技術評論社, 2017.

・筧捷彦, 高田大二 他, 入門 C 言語, 実教出版株式会社, 2019.