変数

1.1 変数とは?

変数とは数値や文字などを入れることができる入れ物.

1.2 変数の名前

変数名 (変数の名前) にはルールがある.

1. 変数に使える文字は次の文字だけ
   - アルファベット (a ～ z, A ～ Z)
   - 数字 (0 ～ 9)
   - アンダースコア (_)

     ※変数名の先頭 (1 文字目) にアンダーバー (_) を使用できるが 避(さ)けた方が良い
2. 数字は変数名の先頭には使えない
3. 予約語 (Python ですでに使われている名前) は変数名にはできない

   予約語の例

   if, elif, else, False, True, try, for, continue, break, and, or, not など
4. アルファベットの大文字小文字は別の変数として区別される

変数名として使える名前の例

a, abc, ABC, a1, xxx, abc_xyz, _a

変数名として使えない名前の例

1a, 2b, $a, 1_, if, else, and, not

1.3 代入

コード 1.1 代入

```
1 a = 5
2 b = "ABC"
3 c
```

変数には数値や文字を入れることができる. このことを**代入**という.
= があるが数学とは違(ちが)い, 左の例では右辺の値 (5) を左辺の変数 (a) に代入することとなる. また, 代入をしていない変数は中身がなく, この状態のことを**未定義**という. (左の例では, 3 行目の変数 c は未定義)

数式

2.1 使える記号

記号	意味	例	結果
+	加算 (足し算)	5 + 8	13
-	減算 (引き算)	90 - 10	80
*	乗算 (掛け算)	4 * 7	28
/	除算 (割り算)	7 / 2	3.5
//	切り捨て除算	7 // 2	3
%	剰余 (あまり)	7 % 3	1
**	累乗	3 ** 4	81
()	括弧 (かっこ)	(2 + 4) * 4	24

2.2 優先順位

優先順位	記号	意味
1	()	括弧 (かっこ)
2	**	累乗
3	*	乗算 (掛け算)
	/	除算 (割り算)
	//	切り捨て除算
	%	剰余 (あまり)
4	+	加算 (足し算)
	-	減算 (引き算)

※優先順位が同じ場合は左から順に計算される．また、演算に使用できる記号はほかにもたくさんある．

型 (type)

3.1 データの型の種類

型名	意味	例
int	整数	1, -12, 2022, 10, など
float	小数	3.14..., 0.5, 12.53, など
str	文字列	'hello', " こんにちは", など

数学では整数は小数 (実数) の中に含まれるが, コンピューターの世界では小数と整数の扱(あつか)いが異なるので明確に別物．文字列は'(シングルクォーテーション), "(ダブルクォーテーション) 記号で囲んだ部分のこと．

3.2 型と四則演算

- int 型と float 型の四則演算
  数学の四則演算と同じ．
  int 型と float 型で四則演算を行うときは, int 型を float 型として型の変換(へんかん)がされ計算が行われる．
- str 型と四則演算
  - str 型では数学のような四則演算はできず, 引き算・割り算はできない
  - str 型同士の足し算は文字列の結合をする (文字列をくっつける)
    (掛け算と違い str 型と int 型の足し算はできない)
  - str 型と int 型の掛け算は文字列をかけた分だけ繰(く)り返す
    (足し算と違い str 型同士で掛け算はできない)

コード 3.2 str 型と四則演算

```
1 # str型の四則演算（足し算と掛け算）
2 a = "1"
3 b = "2"
4 print(a + b)  # str型の足し算
5 print(a * 3)  # str型の掛け算
```

出力

12
111

説明

str 型の変数 a と b が結合されたため出力が 12 となった.
a * 3 で 1 を 3 回繰り返すため, 出力が 111 となった.

3.3 str 型の数字

コード 3.3 int 型と str 型

```
1 # int型とstr型の出力
2 int_val = -30    # int_valはint型（整数）
3 str_val = '-30'  # str_valはstr型（文字列）
4 print(int_val)
5 print(str_val)
```

出力

-30
-30

説明

出力の見え方は全く同じでも, -30 が int 型か str 型かという違いがある.

if 文

もし (条件式) ならば○○ (処理) を実行する, というように条件に合った時だけ ○○ (処理) を行うためのもの.

4.1 if 文の書き方

コード 4.4 if 文の基本構文-その 1

```
1 if 条件式 :
2     処理
```

コード 4.5 if 文の基本構文-その 2

```
1 if 条件式 :
2     処理1
3 else :
4     処理2
```

コード 4.6 if 文の基本構文-その 3

```
1 if 条件式1 :
2     処理1
3 elif 条件式2 :
4     処理2
5 else :
6     処理3
```

コード 4.4
条件が一つのとき

コード 4.5
条件が一つとその条件以外のときに処理をしたいとき

コード 4.6
条件が複数のときとそれらの条件以外のときに処理をしたいとき

注意

- 条件式の後ろには必ずコロン (:) をつける
- 条件式の後ろには必ずインデント (字下げ) をする
- else の後ろには条件式は書かない

4.2 条件式で使う記号

記号	意味	例	例の意味
==	等しい	$x == 5$	x は 5 と等しい
!=	等しくない	$x != 3$	x は 3 と等しくない
>	より大きい	$5 > 2$	5 は 2 より大きい
<	より小さい	$2 < 3$	2 は 3 より小さい
>=	以上	$a >= 0$	a は 0 以上
<=	以下	$b <= 0$	b は 0 以下

等しい事を示す記号は, 2 つの等号 (==) が使われている.
1 つの等号 (=) では代入になってしまうので注意
また, 数学では以上・以下は ≧・≦ と書くが, プログラミングではそのように書くことができないため代わりに >=・<= と書く.

4.3 if 文の例

コード 4.7 if 文の例-その 1

```
1 # 変数numの値が2の倍数かどうかの判定
2 if num % 2 == 0 :
3     print("num は2の倍数です")
4 else :
5     print("num は2の倍数ではありません")
```

コード 4.8 if 文の例-その 2

```
1 # 変数numが正の数か負の数か0かどうかの判定
2 if num > 0 :
3     print("num は正の数")
4 elif num < 0 :
5     print("num は負の数")
6 else :
7     print("num は0")
```

コード 4.7

num の値が 2 の倍数ならば,
”num は 2 の倍数です” と出力.
num の値が 2 の倍数でないならば,
”num は 2 の倍数ではありません” と出力.
2 の倍数の判定は num % 2 == 0 で行うことができる.

コード 4.8

num が 0 より大きいとき (num $>$ 0)
”num は正の数” と出力.
num が 0 より小さいとき (num $<$ 0)
”num は負の数” と出力.
それ以外のとき (つまり num が 0 のとき)
”num は 0” と出力.

複雑な if 文

4.4 if 文の仕組み

if 文の条件式からは True/False という値が返ってくる. これらは bool 型という「分類の値を持つ型」の値である.
条件式があっているときは True 条件式が間違っているときは False
if 文が実行されるときは条件式が True のときである.

4.5 論理演算

複数の条件があるときに使う.

4.5.1 and(論理積)

複数の条件がすべて合っていてほしいときに使う.
すべての条件が合っているときその条件式全体は True となる.

コード 4.9 and の例

```
1 if num % 3 == 0 and num % 5 == 0 :
2     print("3 と 5 の倍数です")
3 else :
4     print("3 と 5 の倍数でもありません")
```

説明

num が 3 の倍数かつ 5 の倍数ならば,
”3 と 5 の倍数です” と出力.
num が 3 の倍数かつ 5 の倍数でないならば,
”3 と 5 の倍数でもありません” と出力.

4.5.2 or(論理和)

複数の条件のどれか一つでも合っていてほしいときに使う.
どれか一つでも条件が合っているときその条件式全体は True となる.

コード 4.10 or の例

```
1 if str == "中学生" or str == "小学生" :
2     print("中学生か小学生です")
```

説明

str が ”中学生” か ”小学生” ならば,
”中学生か小学生です” と出力.

4.5.3 not(否定)

True ならば False, False ならば True となる.

コード 4.11 not の例

```
1 if not (num % 2 == 0) :
2     print("奇数です")
3 else :
4     print("偶数です")
```

説明

num が偶数でないならば, ”奇数です” と出力.
num が偶数ならば, ”偶数です” と出力.

4.6 計算の優先順位 (再び)

優先順位が高いほど先に計算が行われる. 同じ優先順位の場合は左から右へ順に計算される. (数学の計算と同じ)

優先順位		同じ演算のときの優先順位
高	括弧 (かっこ)	
	算術演算	累乗 (**)
		乗算 (*), 除算 (/), 切り捨て除算 (//), 剰余 (%)
		加算 (+), 減算 (-)
	比較演算	>, >=, <, <=
	論理演算	not(否定)
		and(論理積)
低		or(論理和)

※この表の同じ枠内での優先順位は同じ.

繰り返し文

5.1 for 文

決められた回数繰り返したいときに使う.

5.2 for 文の書き方

コード 5.12 for 文の基本構文

```
1 for 変数名 in 繰り返す回数 :
2     繰り返したい処理1
3     繰り返したい処理2
```

注意

- for 文の後ろには必ずコロン (:) をつける
- for 文の処理はインデント (字下げ) をする (インデントの有無で繰り返す処理の範囲が変わってしまう)

5.3 for 文の例

コード 5.13 for 文の例

```
1 # 0 から 5 までのすべて足したときの値
2 s = 0
3 for i in range(6) :
4     print(i)        # 繰り返し部分
5     s = s + i  # 繰り返し部分
6
7 print(s) # for文の範囲外
```

出力

0
1
2
3
4
5
15

説明

繰り返し：1 回目
：2 回目
：3 回目
：4 回目
：5 回目
：6 回目
7 行目の print 関数の出力

5.4 whlie 文

for 文と同じく繰り返しが必要な処理を行うために使うもの.
条件式が True の間繰り返しが続き, False のとき繰り返しが終了する.

5.5 while 文の書き方

コード 5.14 while 文の基本構文

```
1 while 条件式 :
2     条件式が'True'のとき繰り返したい処理
```

注意

- while 文の後ろには必ずコロン (:) をつける
- while 文の処理はインデント (字下げ) をする
  (インデントの有無で繰り返す処理の範囲が変わってしまう)

コード 5.15 while 文の注意点

```
1 while True : # もしくは条件式がずっとTrue
2     繰り返したい処理  # 処理が永遠に繰り返される
```

while 文の注意点

while 文は条件式が True のとき処理を繰り返すので, 「while True:」のときは永遠に処理を繰り返す. また, 指定した条件式を間違えたり, 条件式で変数を使っているときに繰り返し処理のなかで変数を条件式を満たすように変化させなかった場合も条件式の結果が True だと永遠に処理を繰り返してしまう.

コード 5.16 while 文の注意点-例 1

```
1 # 1 から 10 までの数字を出力 (n への加算操作がないため永遠に繰り返す)
2 n = 1
3 while n < 10 :
4     print(n)
```

コード 5.17 while 文の注意点-例 2

```
1 # a が 5 のとき Buzz を出力して終了 (break を忘れているため永遠に繰り返す)
2 a = 1
3 while True :
4     a = a + 1
5     if a == 5 :
6         print("Buzz")
```

5.6 break

for 文や while 文の繰り返し処理の中で繰り返しを中断したいときに使う.
break を使うと for 文や while 文の**繰り返し処理自体が終了**する.

コード 5.18 break の使い方

```
1 n = 0
2 while True :
3   n = n + 1
4   print(f'n = {n}')
5   if n == 5:
6     break
```

出力

n = 1
n = 2
n = 3
n = 4
n = 5

説明

while True で永遠に繰り返しが実行されそうだが, n が 5 のときに break によってループが終了するため, 5 まで出力された.

5.7 continue

繰り返し処理の中で特定の処理のときだけ, その処理をスキップする場合に使う.
continue を使っても for 文や while 文の**繰り返し自体は終了しない**.

コード 5.19 continue の使い方

```
1 n = 0
2 while n < 5 :
3   n = n + 1
4   if n == 3:
5     continue
6   print(f'n = {n}')
```

出力

n = 1
n = 2
n = 3
n = 4
n = 5

説明

n が 3 のときに continue によってループがスキップされたため,
3 のときだけ結果が出力されなかった.

5.8 条件式と True と break

コード 5.20 条件式による繰り返しの終了

```
1 n = 0
2 while n < 5 :
3   n = n + 1
4   print(f'n = {n}')
```

コード 5.21 break による繰り返しの終了

```
1 n = 0
2 while True :
3   n = n + 1
4   print(f'n = {n}')
5   if n == 5:
6     break
```

while の条件式による終了か break による終了かの違いだけで, どちらも, n が 5 のときに繰り返し処理が終了する同じ処理である. ただし, break での処理の方では, break を忘れてしまうと処理が永遠に繰り返すので注意が必要となる.

関数

6.1 関数とは

関数とは, 決められた処理を実行してその結果を返す
プログラムの部品のようなもの.
関数には値を入れることができ, 入れる値のことを**引数**
(**ひきすう**) と言い, 値を処理して返ってきた結果を
返り値/**戻り値** (**かえりち**/**もどりち**) と言う.
関数には Python ですでにあるものや自分で作れるものがある.
関数の引数に関数を入れたり, 関数の返り値を変数に代入する
ことが出来る.

6.2 関数を使うときの例

コード 6.22 関数

```
1 # 関数の使い方
2
3 関数名(引数) # 引数あり
4 関数名()     # 引数なし
```

6.3 Python でよく使う関数

- print 関数
  画面に文字列や変数の値を出力する関数. print 関数の引数には文字列や変数を入れることができる.
  引数：文字列や数値, 計算結果

  ◦ print 関数の使い方

コード 6.23 print 関数

```
1 print("こんにちは")
2 print(1000)
3 a = 10
4 b = 30
5 print(a)
6 print(a + b)
```

出力

こんにちは
1000
10
40

説明

引数の文字列や数値, 変数の値, 計算式の計算結果を出力する

  ◦ print 関数で変数と文字を一緒に使う

コード 6.24 f フォーマット

```
1 name = "タロウ"
2 print(f'{name}さん、こんにちは')
```

出力

タロウさん、こんにちは

説明

文字列を f" で囲み (f' 文字列' となるように), { } の中に変数を入れることで変数と文字列を一緒に出力することができる.

- range 関数

  一定の間隔での数値の並びを生成する関数

  range 関数の引数の与え方は, range(開始値, 終了値, ステップ幅) ある.

  開始値とステップ幅はそれぞれ省略でき, 省略した場合は (開始値：0, ステップ幅：1) が初期値となる.

  生成できる値は開始値から**終了値より手前の整数**までの範囲である.

コード 6.25 range 関数

```
1 for i in range(5) :
2     # 0, 1, 2, 3, 4
3
4 for i in range(5, 10) :
5     # 5, 6, 7, 8, 9
6
7 for i in range(0, 10, 2) :
8     # 0, 2, 4, 6, 8
9
```

説明

3 行目では開始値, ステップ幅が省略されているため, 0 から始まり, 1 ずつ上がっていく

6 行目では 5 から始まり, 9 までの値を生成する

9 行目ではステップ幅が 2 のため, 2 ずつ上がり, 終了値 10, つまり 9 までの範囲で値を生成する

- max 関数

  2 つ以上の引数でその中から最大の値を返す関数

  引数：数値, 文字列, リスト　　返り値：最大値

コード 6.26 max 関数

```
1 max_num = max(1, 3, 4, 2)
2 max_str = max("abc", "xyz", "ijk")
3 print(max_num)
4 print(max_str)
```

出力

```
4
xyz
```

説明

max_num には (1, 3, 4, 2) の中で一番大きい値である 4 が代入される.

max_str には (”abc”, ”xyz”, ”ijk”) の中で一番大きい値である xyz が代入される.

- min 関数

  2 つ以上の引数でその中から最小の値を返す関数

  引数：数値, 文字列, リスト　　返り値：最小値

コード 6.27 min 関数

```
1 min_num = min(1, 3, 4, 2)
2 min_str = min("abc", "xyz", "ijk")
3 print(min_num)
4 print(min_str)
```

出力

```
1
abc
```

説明

min_num には (1, 3, 4, 2) の中で一番小さい値である 1 が代入される.

min_str には (”abc”, ”xyz”, ”ijk”) の中で一番小さい値である abc が代入される.

- input 関数
  変数に任意 (好き) な文字列を代入する事が出来る関数
  引数：なし

コード 6.28　input 関数

```
1 value = input()
2 print(value)
```

入力

abcdef

出力

abcdef

説明

入力した abcdef が value に代入され, print 関数で出力される

- type 関数
  データの種類を調べる関数
  引数：データ　　返り値：種類名

コード 6.29　type 関数

```
1 int_val = 3
2 float_val = 3.14
3 print(type(int_val))
4 print(type(float_val))
```

出力

<class 'int'>
<class 'float'>

説明

int_val の値は int 型, float_val の値は float 型とデータの種類が出力された

- len 関数
  引数の値の長さを返す関数

コード 6.30　len 関数

```
1 s = "abcde"
2 l = len(s)
3 print(l)
```

出力

5

説明

変数 s の値 "abcde" の長さが 5 のため, len 関数の戻り値は 5 となる

- sum 関数
  引数の値の合計を返す関数

コード 6.31　sum 関数

```
1 l = [1, 2, 3, 4, 5]
2 s = sum(l)
3 print(s)
```

出力

15

説明

変数 l の値 [1, 2, 3, 4, 5] の合計が 15 のため, sum 関数の戻り値は 15 となる

- int 関数
  引数に与えられた値を int 型に変換する関数

コード 6.32 int 関数

```
1 float_val = 3.1415
2 int_val   = int(float_val)
3 print(int_val)
4 print(type(int_val))
```

出力

```
3
<class 'int'>
```

説明

int 型は整数型のため小数が整数に変換され, 3.1415 が 3 となった.
type 関数の返り値からも int 型になったことが分かる.

- float 関数
  引数に与えられた値を float 型に変換する関数

コード 6.33 float 関数

```
1 int_val   = 100
2 float_val = float(int_val)
3 print(float_val)
4 print(type(float_val))
```

出力

```
100.0
<class 'float'>
```

説明

float 型は小数の型のため整数が小数に変換され, 100 が 100.0 となった.
type 関数の返り値からも float 型になったことが分かる.

- str 関数
  引数に与えられた値を str 型に変換する関数

コード 6.34 sum 関数

```
1 int_val = 100
2 str_val = str(int_val)
3 print(str_val)
4 print(type(str_val))
```

出力

```
100
<class 'str'>
```

説明

見た目ではわからないが int 型から str 型に変換されたことが type 関数によってわかる

list(リスト)

7.1 リストとは

複数の値に順番をつけてまとめて扱うためのデータ型の一つ.

7.2 リストの生成

コード 7.35 リストの生成

```
1 a = [5, 1, 3, 4]
2 b = ['abc', 'def', 'ghi']
3
4 t = 5
5 c = [t, 1, 3, 4]
```

説明

リストの生成には [ ] の中に , で区切り具体的にリストの中身を書くことで生成できる.
4,5 行目のように変数を入れてもよい

7.3 リストの使い方

リストの個々のデータのことを**要素**という言い, 要素の順番を指定する値のことを**添え字**という言い方をする.
また, 要素の順番は 0 から始まる.

7.3.1 要素へのアクセス方法

コード 7.36 要素へのアクセス方法 1

```
1 #    要素の順番
2 #    0  1  2  3
3 a = [5, 1, 3, 4]
4 print(a[0])
5
6 # 正の添え字と負の添え字
7 print(a[3], a[-1])
```

出力

5
4 4

説明

リストの 0 番目である値 5 が出力される.
添え字には負の値を指定することができ, 負の値を指定した場合はリストの後ろの要素から指定されていく.

コード 7.37 要素へのアクセス方法 2

```
1 #    要素の順番
2 #    0  1  2  3
3 a = [5, 1, 3, 4]
4 print(a[0:3])
```

出力

[5, 1, 3]

説明

添え字に [先頭番号: 終了番号] を与えると, リストの一部だけを取り出すことができる.
このとき, 終了番号の手前までの値が取り出せることに注意する. このような添字の指定の仕方を**スライス**という.

7.3.2 リストの基本操作

- 要素の追加
  リストの末尾に要素を追加する

コード 7.38 要素の追加

```
1 a = [1, 2, 3, 4]
2 a.append(5)
3 print(a)
```

出力

[1, 2, 3, 4, 5]

説明

リストの末尾に指定した値が追加される. append() を使うとき () の中は**追加したい値**を書く.
リストへの追加は append() 以外にも insert() 関数を使うことで任意の位置に追加することができる.

- 要素の削除
  リストの末尾の要素を削除する

コード 7.39 要素の削除

```
1 a = [1, 2, 3, 4]
2 a.pop()
3 print(a)
4 a.pop(0)
5 print(a)
```

出力

[1, 2, 3]
[2, 3]

説明

引数を指定しない場合はリストの末尾を, 指定した場合は指定した添え字の値が削除される.
pop() を使うときは () の中は, **削除したい値の添え字**にする.

7.4 スライス

リストの一部を取り出すことができる.
スライスの書き方は [先頭番号: 終了番号] である.
このとき, 終了番号の手前までの値が取り出せることに注意する.

コード 7.40 スライス

```
1 a = [1, 2, 3, 4, 5]
2 print(a[0:3])
3 print(a[1:2:4])
4 print(a[2:])
5 print(a[:3])
```

出力

```
[1, 2, 3]
[2, 3, 4]
[3, 4, 5]
[1, 2, 3]
```

説明

1 行目ではリストの 0 番目から 2 番目までの値が取り出されている.
2 行目ではリストの 1 番目から 3 番目までの値が取り出されている.
3 行目ではリストの 2 番目から最後までの値が取り出されている.
4 行目ではリストの 0 番目から 2 番目までの値が取り出されている.

自作関数

8.1 自作関数とは

自分で関数を作ることができる.
関数を作ることで, 何度も同じ処理を書く必要がなくなり, プログラムの見通しが良くなる.

8.2 自作関数の書き方

コード 8.41 自作関数の書き方

```
1 def 関数名(引数):
2     実行したいプログラム
3     return 返り値          # 返り値がある場合
```

関数を作るときのルール

- 複数の関数を作るときは, 関数名が被らないように注意をすること. 関数名のルールは変数名の時と同じ.
- 引数は 2 つ以上でもよい. そのときは , で引数と引数を区切る.
  また, 引数はなくてもよい. しかし, その場合でも関数名の後ろには () をつけなければいけない
- return 文は関数定義の最後以外の場所, if 文の中などに書いてもよい.
  返り値がない, いらない場合は return 文を書かなくてもよい.
- 関数の返り値には計算式を書いてもよい.
- 関数を作る時の def 後の ( ) 内の引数名と実際に呼び出して使う時の引数名は一致している必要はない.
- 関数の中身はインデントを揃えて書く.

スコープ

9.1 スコープとは

スコープとは, 変数が有効な範囲のことである. スコープには大きく分けて 2 つの種類がある.

- ローカル変数　：限られた範囲で使われる変数
  関数内で定義された（代入された）変数は関数内でのみ利用可能で, 関数の実行ごとに関数の実行が終了すれば失われる
- グローバル変数：プログラム全体で使える変数
  関数外で定義されている変数は値を読み取ることのみ可能. 関数内で global 宣言された変数のみ, グローバル変数に代入可能. この global 宣言をせずに代入をするとグローバル変数がローカル変数になってしまう.

モジュール

10.1 モジュールとは

モジュールとは, Python でよく使われる関数や定数をまとめたもの.
モジュールを使うことで, そのモジュールに定義されている関数や定数を使うことができる.
モジュールも自作することができる.

10.2 モジュールのインポート方法

モジュールのインポート方法は大きく分けて以下の 2 つ

- import モジュール名
- from モジュール名 import 機能名

10.2.1 import と from について

- import〜
  モジュール全体を利用する, という宣言
  モジュールの関数を使うときは, 関数の前にモジュール名を付ける必要がある.
- from〜
  モジュールの一部を利用する, という宣言
  1 つ 1 つ呼び出す代わりに関数を使うときに関数の前にモジュール名は付けない.
  from モジュール名 import * はそのモジュールの関数や変数をすべて直接呼び出している.

import〜と from〜の使い分け

基本は自由. 自分やチームのルールがある場合はそれに従って書く.
変数名や関数名はかぶる可能性が有るため, 少数しかモジュールや関数を使わないなら from〜を
たくさんのモジュールを使うときは import〜を使う方がいい場合が多い.

10.3 自作モジュールを使用したインポートの紹介

自作のモジュールを使うときは, モジュールのファイルが使いたいプログラムのファイルと同じ場所にあり, モジュール名と同じファイルがないように気を付ける.
次のような自作モジュール (sample.py) があるとする. このときの, インポート方法とモジュールの変数へのアクセス方法を紹介する.

コード 10.42 sample.py

```
1 hello = "こんにちは"
2 color = ["red", "green", "黒"]
3
4 def right(var_list) :
5     ret = var_list[-1]
6     print(ret)
7     return ret
```

コード 10.43 インポートの仕方 1

```
1 import sample
2
3 sample.color
4 sample.hello
```

import モジュール名

モジュール名. というのを頭につけて変数や関数の名前を書く. 今回であれば sample. をつける必要がある.

コード 10.44 インポートの仕方 2

```
1 import sample as sp
2
3 sp.color
4 sp.hello
```

import モジュール名 as 別の名前

モジュール名を別の名前に変更して, 変更した名前を頭につけて変数や関数の名前を書く. 今回であれば sample. ではなく sp. をつける必要がある.
この方法はモジュール名が長いものを省略して書きたいときによく使われる方法である.

コード 10.45　インポートの仕方 3

```
1 from sample import color, right
2
3 color
4 hello
```

from モジュール名 import モジュールの関数や変数 (複数可)

モジュール名. というのを頭につけて関数の名前や変数を書く必要がなくなる.
しかし, 宣言していない関数や変数は使えないため注意 (今回では hello は宣言していないため使えない)

コード 10.46　インポートの仕方 4

```
1 from sample import *
2
3 color
4 hello
```

from モジュール名 import *

モジュールの関数や変数を import の後ろに
書かなくてもモジュールの関数や変数が使えるようになる.