

## 第 I 編 基本文法と制御処理

1 基礎事項	コードの入力と実行 プログラムの書き方・保存・実行 コメント文 標準出力・標準入力 文字列と数値
2 変数と式	
3 演算子	四則演算子 比較演算子 論理演算子 ビット演算子 代入演算子 条件演算子
4 特殊演算子	range（範囲） スライス
5 制御演算子	分岐演算子 繰り返し演算子

## 1 基礎事項

### (1) コードの入力と実行

実行方法は2通りがある。

(A) インタラクタモード

(B) スクリプトモード

#### (A) インタラクタモード

内容：電卓みたいに1行ずつ対話的に入力・実行する。

方法：コマンドプロンプト画面上で行う。

例題：「2 + 3」の結果、「ようこそpythonへ!」を表示するプログラム。

```

C:\> コマンドプロンプト - python
C:\>
C:\> python
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49)
t (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for
>>> 2+3
5
>>> 'ようこそpythonへ!'
'ようこそpythonへ!'
>>>
  
```

```

2+3
'ようこそPythonへ!'

x = 2
y = 3
z = x + y
str = 'ようこそPythonへ!'
print(z)
print(str)
  
```

参考として、上記はプログラムのようにも書くことができる。

```

C:\> コマンドプロンプト - python
C:\>
C:\> python
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49)
t (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for
>>> x=2
>>> y=3
>>> z=x+y
>>> str='ようこそpythonへ!'
>>> print(z)
5
>>> print(str)
ようこそpythonへ!
>>>
  
```

```

x = 30
z = x + y
print(z)

x = 30
z = x + y
print(z)
  
```

**(B) スクリプトモード**

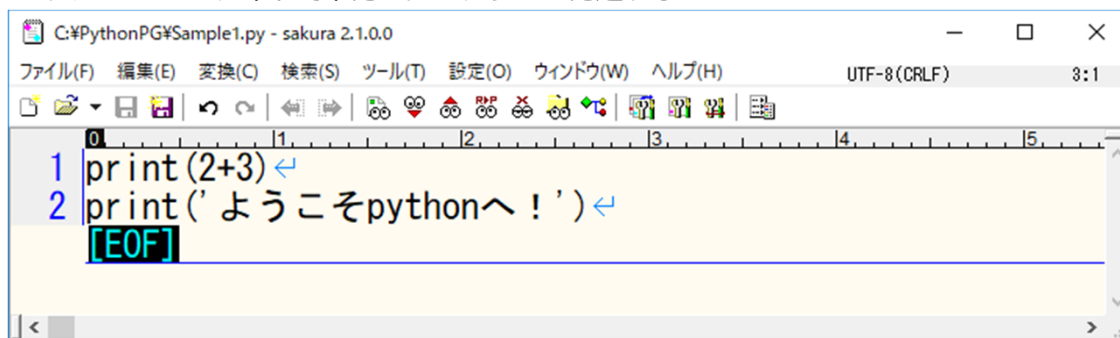
内容：普通のプログラム言語のようにプログラムをファイルに保存して、実行する。

方法：コマンド画面で、C:¥pythonPG>python 保存ファイル名.py [リターン] で実行する。

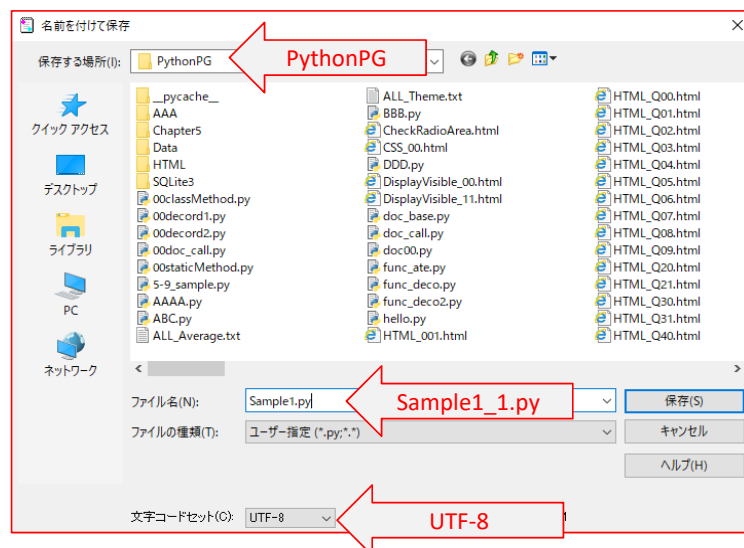
例題：

プログラムを文字コード[UTF-8]、ファイル名[C:¥PythonPG¥Sample1\_1.py]で保存する。

まず、Sakuraエディトで下記のプログラムを記述する。



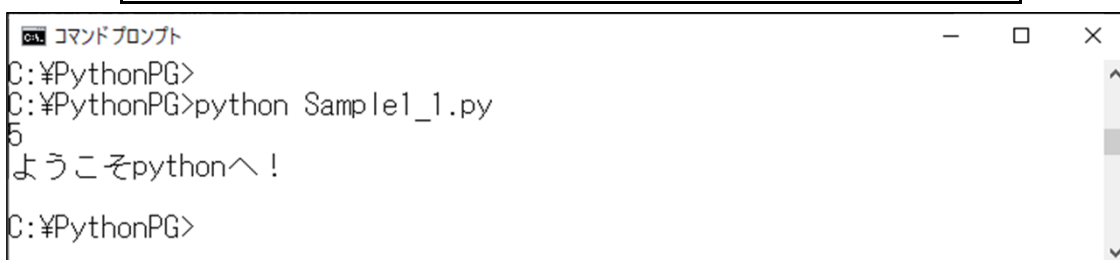
つぎに「ファイル」「名前を付けて保存」で下記のように保存する。



**注：**何故、UTF-8で保存するかというと、インストール時に文字コードを「UTF-8」に設定したためである。なお、このことは後で確認する。

最後にコマンドプロンプト画面で実行する。

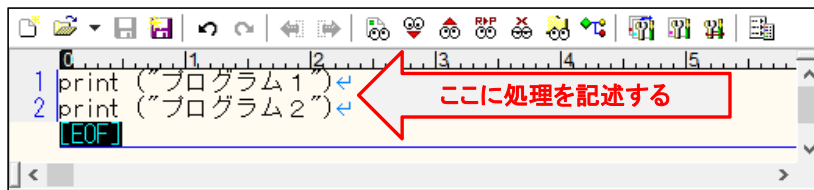
**実行の命令語：** C:¥PythonPG>python Sample1\_1.py



## (2) プログラムの書き方・保存・実行

## (A) プログラム

単に画面に文字列を表示するのみのプログラムは



'プログラム 1'でもよい  
アポストロフィ  
シングルクォーテーション

のようになる。行の先頭に空白なし、最後尾にセミicolon[;]なし。  
サクラエディットを開いて、上記のプログラムを作成しなさい。

## (B) 保存

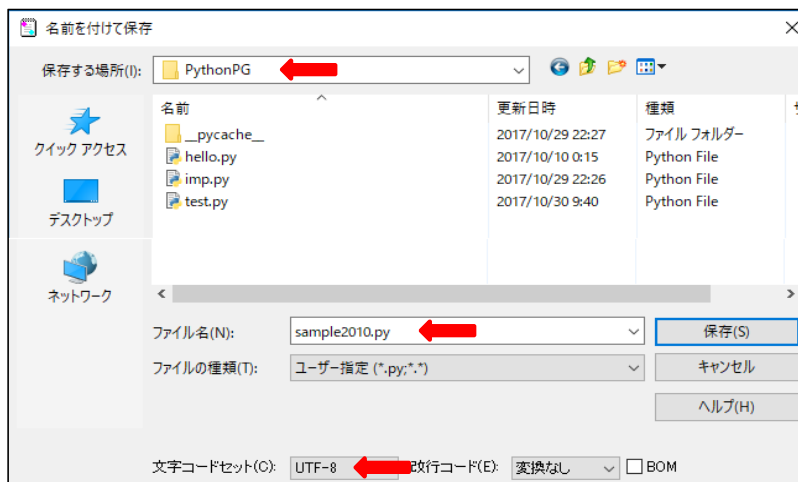
保存の書式は下記である。

書 式 任意のファイル名.py # 拡張子.py

作成したプログラムをフォルダ[C:¥pythonPG¥]に sample2010.py として保存する。

ただし、pythonをインストールするときに、文字コード「UTF-8」としたので、

保存するときに文字コード「UTF-8」を設定する。



## (C) 実行

ファイルが C:¥pythonPG¥sample2010.py にあり、インストール時  
フォルダ[C:¥pythonPG¥] (全て) にPathを通してあるので

書 式 C:¥pythonPG>python sample2010.py  
または  
C:¥>python C:¥pythonPG¥sample2010.py

で実行できる。

```

コマンドプロンプト
C:¥PythonPG>python sample2010.py
プログラム 1
プログラム 2

C:¥PythonPG>cd ..

C:¥>python C:¥PythonPG¥sample2010.py
プログラム 1
プログラム 2

```

ファイルのある場所：

[c:¥PythonPG]

Sample2010.py

ルート・ディレクトリ

カーソルのあるフォルダ

C:¥Pythnpg>

または

C:¥>

**(D) 半角スペース** 文の先頭に同じインデント(字下げ)すること

プログラムにおいて、行の最初(文頭)の半角空白(タブ可)は重要な意味を持つ。

逆に、(後述するが) for, if, function, class文 などでは先頭から書くと、

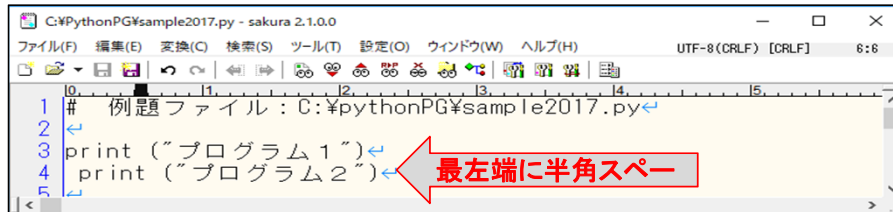
正常に処理せず、エラーも出ないので特に注意が必要である。

一般には行の最初からプログラムを書くことに心がける。

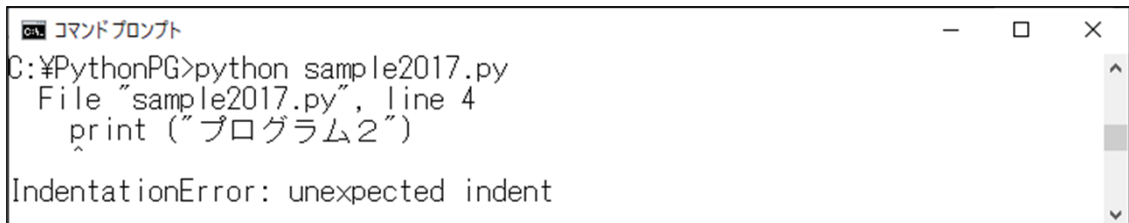
まず、半角スペースを入れたときにエラーになることを確認するために

つぎのプログラムを C:\pythonPG\sample2017.py として作成・保存して実行する。

[ C:\pythonPG\sample2017.py]



このプログラムをコマンドプロンプトで実行すると、下記のようなエラー結果がでる。



さらに、半角スペースを削除すると正常になる。

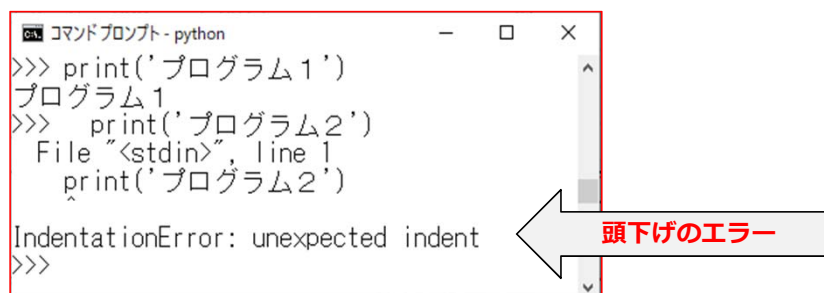
**インタラクタモード：**

また、このことは**インタラクタモード**でも同様である。

```

>>> print('プログラム 1')
>>> print('プログラム 2')

```



## (2) コメント文 ( # )

プログラムに、日本語の説明などを付加すると見やすくなる。これがコメント文( # )である。

書 式	# コメント	# 以降の行のみコメント
説 明	1行すべてコメントにしたり、式の後にコメントをつめる。	複数行は？
例 題	<pre># 例題ファイル：C:¥pythonPG¥sample2020.py # 長方形の面積を求める x=2.0 # 縦の長さ y=3.0 # 横の長さ # 面積=縦 x 横 z=x*y print( '面積(%s)=縦(%s) x 横(%s)' %(str(z), str(x), str(y)) )</pre>	<pre>..... 例題ファイル・・・ 長方形の・・・・ ..... とすればよい。(正規でないよ)</pre>

上をサクラエディットを用いて、C:¥pythonPG¥sample2020.py 名でプログラムしなさい。  
実行すると下記の結果が得られ、コメント文は表示されないことがわかる。

```

コマンドプロンプト
C:¥PythonPG>python sample2020.py
面積(6.0)=縦(2.0) x 横(3.0)

```

## (3) 標準出力・標準入力

## (A) 標準出力

ここでは、標準出力とはコマンドプロンプト画面への出力(表示)を考える。

標準出力としては、つぎの**print()**関数を用いる。

書 式	print ( "文字列" )	# 文字列
	print ( 100 )	# 数値
	var=100	
	print ( var )	# 変数varの値100 を出力
	print( "var=", var)	# var= 100 と出力される。
書式つき出力 (下記の <b>%演算子</b> を用いる)		
	print ( "文字列=%s" %"おはよう")	# 文字列=おはよう と出力される

(%演算子)

%演算子	意 味
d	符号付き10進整数
i	符号付き10進整数
x	符号付き16進数(小文字)
X	符号付き16進数(大文字)
e	指数表記の浮動小数点数(小文字)
E	指数表記の浮動小数点数(大文字)
f	10進浮動小数点数
F	10進浮動小数点数
c	文字一文字
r	文字列(repr())で変換
s	文字列(str())で変換

特に使う演算子

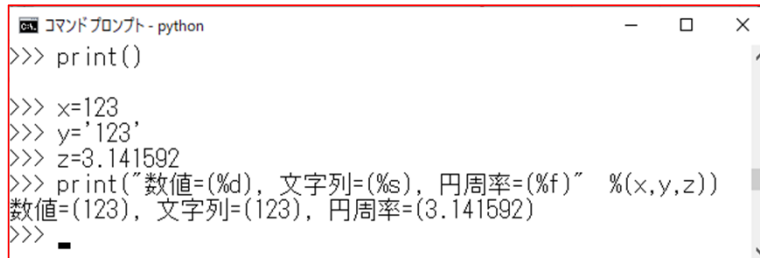
%d	整数の場合
%f	実数の場合
%s	文字列の場合

他言語と同じ

%4.2f	
x=3.141592	
print("%4.2f" %x)	# 3.14

**インタラクタモード：**

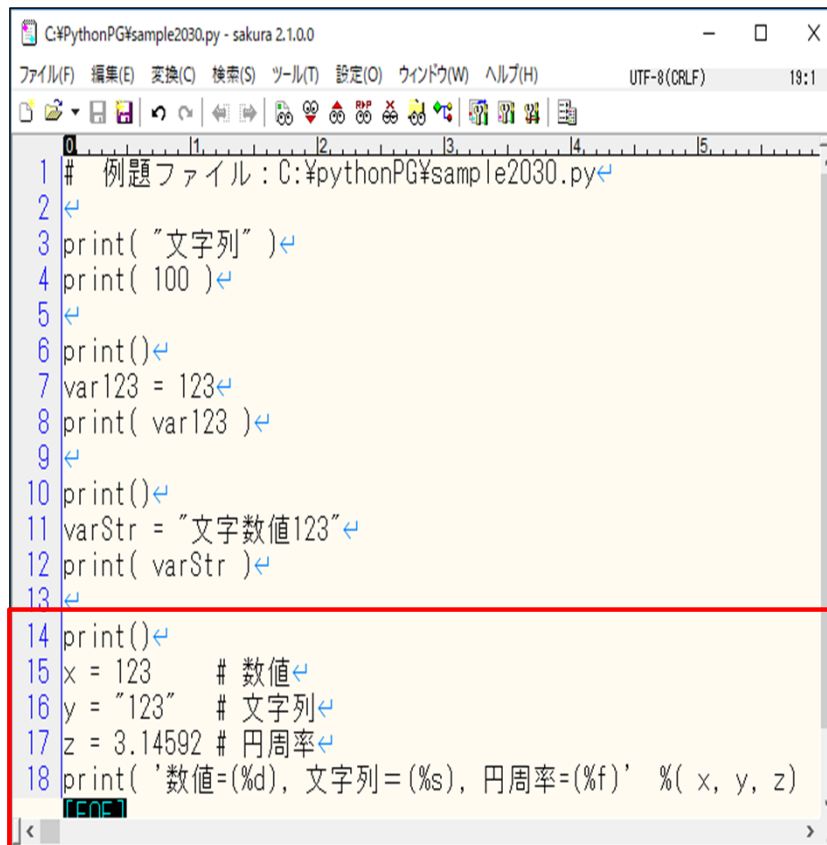
```
>>> print()
>>> x=123
>>> y='123'
>>> z=3.141592
>>> print("数値=(%d), 文字列=(%s), 円周率=(%f)" %(x,y,z))
```



```
コマンドプロンプト - python
>>> print()
>>> x=123
>>> y='123'
>>> z=3.141592
>>> print("数値=(%d), 文字列=(%s), 円周率=(%f)" %(x,y,z))
数値=(123), 文字列=(123), 円周率=(3.141592)
>>>
```

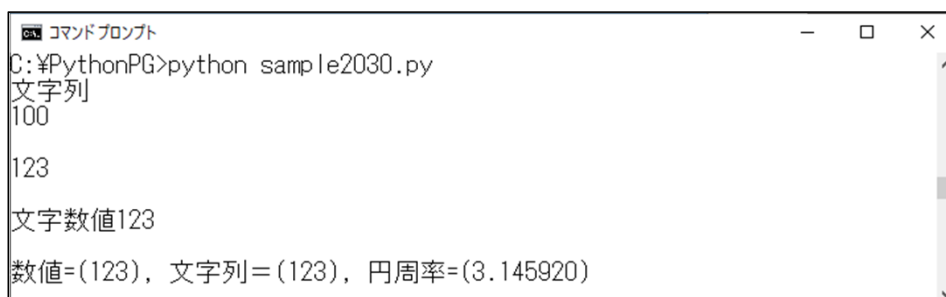
**スクリプトモード：**

print(値)関数, print(変数名)関数, print(%書式付き)関数が存在するプログラムを  
 [ C:¥pythonPG¥sample2030.py ] として作成しなさい。**(赤枠だけで上で実行した)**



```
C:¥PythonPG¥sample2030.py - sakura 2.1.0.0
ファイル(F) 編集(E) 変換(C) 検索(S) ツール(T) 設定(O) ウィンドウ(W) ヘルプ(H) UTF-8(CRLF) 19:1
1 # 例題ファイル：C:¥pythonPG¥sample2030.py
2
3 print( "文字列" )
4 print( 100 )
5
6 print()
7 var123 = 123
8 print( var123 )
9
10 print()
11 varStr = "文字数値123"
12 print( varStr )
13
14 print()
15 x = 123      # 数値
16 y = "123"    # 文字列
17 z = 3.14592 # 円周率
18 print( '数値=(%d), 文字列=(%s), 円周率=(%f)' %( x, y, z) )
```

作成したプログラムを実行すると、次の結果が得られる。



```
コマンドプロンプト
C:¥PythonPG>python sample2030.py
文字列
100

123

文字数値123

数値=(123), 文字列=(123), 円周率=(3.145920)
```

上記のプログラムと結果を見ると、5個のprint()文があり、それぞれの結果が別々の行に出力されていることが分かる。これに対して、同じ行に出力したい場合がある。この場合の書式は、上記例題の最初のprint()文で示すと、つぎのようになる。

書 式	print( varStr, end=" " )	# 改行なし
-----	--------------------------	--------

つぎに上記例題を1行にまとめて多くの出力表示するような場合を示す。

先ほどの[ C:¥pythonPG¥sample2030.py ] を参考に、

下記のように修正して、[ C:¥pythonPG¥sample2035.py ] として作成しなさい。

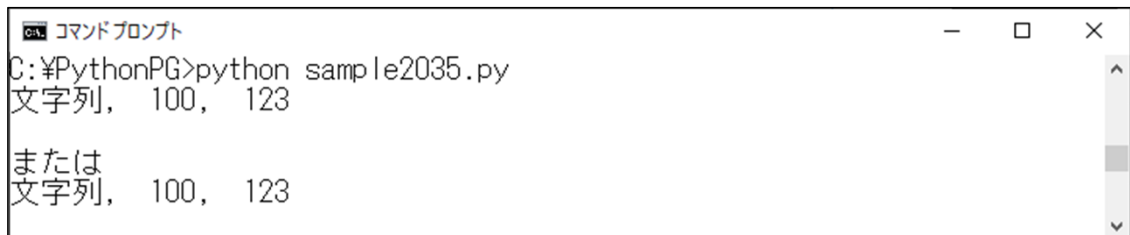
# 例題ファイル : C:¥pythonPG¥sample2035.py

```
print( "文字列", end=" " )
print( 100 , end=" " )
var123 = 123
print( var123 )

print()
print("または")
print( '%s, %d, %d' %("文字列", 100, var123))
```

このプログラムを実行しなさい。

下記の**赤矢印**のように改行されないことを確認しなさい。



```
コマンドプロンプト
C:¥PythonPG>python sample2035.py
文字列, 100, 123

または
文字列, 100, 123
```



## (B) 標準入力

ここでは、標準入力とはキーボード入力を考える。

つぎのように標準入力(キーボード)から**文字列**を変数strに取得する。

書式	str = input()	# ■ (<=カーソル位置)
	または str = input("文字列=")	# 文字列= ■ (<=カーソル位置)

## インタラクタモード：

```
>>> input()
3.14
'3.14'
>>> input("入力=")
入力=3.14
'3.14'
>>>
```

上の結果より、キーボード入力は**文字列**であることが分かる。

## スクリプトモード：

下記を[ C:\pythonPG\sample2041.py ] にプログラムしなさい。

```
1 # 例題ファイル：C:\pythonPG\sample2041.py
2
3 print("")
4 str = input('文字列入力 == ')
5 print('入力された文字列 = (%s)' %(str))
6
7 print("")
8 str = input('KeyBordから数値を入力 == ')
9 print('変換された数値 = (%d)' %(int(str)))
10
```

実行して任意の文字をキーボード入力しなさい。

下記は、キーボードから「あああ」と「123」を入力した場合の実行結果である。

```
コマンドプロンプト
C:\PythonPG>python sample2041.py

文字列入力 == あああ
入力された文字列 = (あああ)

KeyBordから数値を入力 == 123
変換された数値 = (123)
```

(C) 文字列・数値==>数値      数値==>文字列      変換      実数 = float(文字列)

変換の書式はつぎのようになる。

書式	文字列==>数値,    数値==>数値	
	str0 = "314"; str1="3.141592"	
	num0 = int( str0 )	# 314(整数)
	real0 = float(str1)	# 3.141592(小数点)
	数値==>文字列	
	x = 3.14	
	str( x )	# "3.14" (文字列)

インタラクタモード：

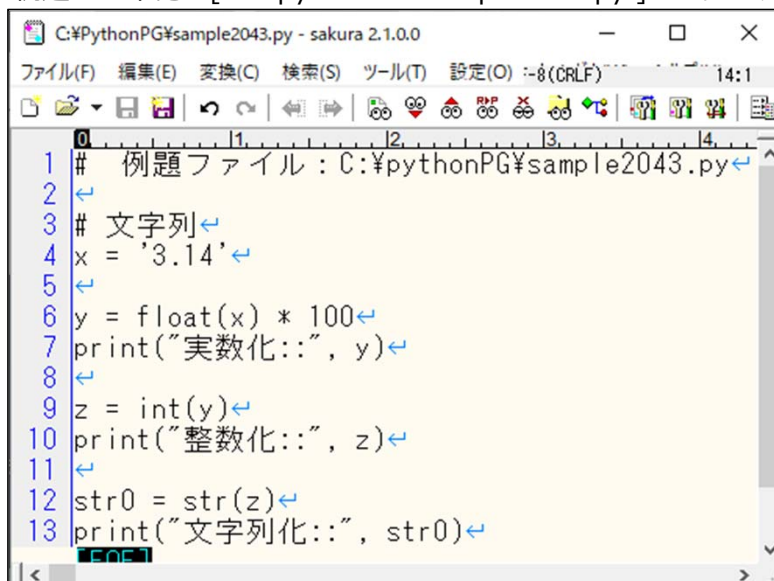
```
>>> x = '3.14'
>>> y = float(x) * 100
>>> print("実数化::", y)
>>> z = int(y)
>>> print("整数化::", z)
>>> str0 = str(z)
>>> print("文字列化::", str0)
```



```
コマンドプロンプト - python
>>> x = '3.14'
>>> y = float(x) * 100
>>> print("実数化::", y)
実数化:: 314.0
>>> z = int(y)
>>> print("整数化::", z)
整数化:: 314
>>> str0 = str(z)
>>> print("文字列化::", str0)
文字列化:: 314
>>>
```

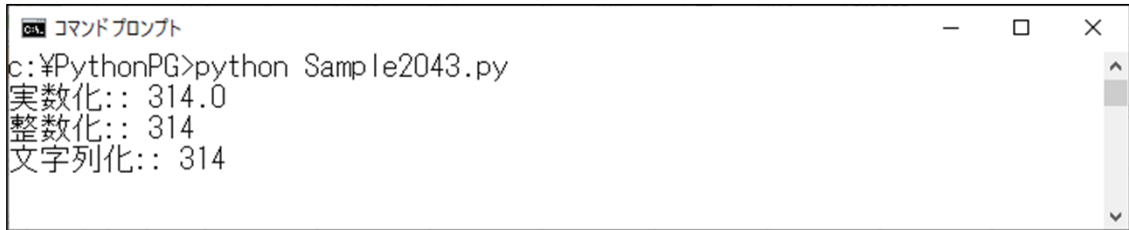
スクリプトモード：

例題： 下記を[ C:\pythonPG\sample2043.py ] にプログラムしなさい。



```
C:\pythonPG\sample2043.py - sakura 2.1.0.0
ファイル(F) 編集(E) 変換(C) 検索(S) ツール(T) 設定(O) :-8(CRLF) 14:1
1 # 例題ファイル：C:\pythonPG\sample2043.py
2
3 # 文字列
4 x = '3.14'
5
6 y = float(x) * 100
7 print("実数化::", y)
8
9 z = int(y)
10 print("整数化::", z)
11
12 str0 = str(z)
13 print("文字列化::", str0)
```

実行結果:



```

c:\PythonPG>python Sample2043.py
実数化:: 314.0
整数化:: 314
文字列化:: 314

```

#### (4) 文字列と数値

##### (A) 文字列リテラル と 数値リテラル

文字列リテラル と 数値リテラル はつぎのような例である。

[文字列リテラル]

"おはよう！"

"abcdef"

[数値リテラル]

3.1459

3124

##### (B) 2進数・8進数・16進数

10進数：0,1,2,3,4,5,6,7,8,9,10,11,...

ここで使われている文字；{0,1,2,3,4,5,6,7,8,9} 10個

2進数：0000,0001,0010,0011,0100,0101,0110,0111,1000,1001,...

ここで使われている文字；{0,1} 2個

16進数：0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,10,11,...

ここで使われている文字；{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E} 16個

2進数と10進数

$$\begin{aligned}
 (1010)_2 &= 1x(2)^3 + 0x(2)^2 + 1x(2)^1 + 0x(2)^0 &= 1x(8) + 0x(4) + 1x(2) + 0x(1) \\
 &= (8 + 0 + 2 + 0)_{10} &= (10)_{10}
 \end{aligned}$$

10進数と16進数

$$\begin{aligned}
 (15)_{10} &= (F)_{16} = 0xF \\
 (16)_{10} &= (10)_{16} = 0x10 \\
 (170)_{10} &= (AA)_{16} = 0xAA
 \end{aligned}$$

2進数・8進数・16進数の記述

書式	10進数
	10
	2進数
	0b1010
	8進数
	0o12
	16進数
	0xA

**(C) エスケープシーケンス**

文字列 ( 彼は"おはよう"と言いました。 ) を表示するときに

```
print("彼は"おはよう"と言いました。") # ==> エラー
```

とすればエラーになる。ただし、[""]が全角ならば[OK]。

このようなときに下記のエスケープシーケンスを用いてつぎのようにする。

```
print("彼は¥"おはよう¥"と言いました。") # ==> 彼は"おはよう"と言いました。
```

エスケープシーケンス表

記号	内 容
¥¥	「¥」文字そのもの
¥'	シングルクォーテーション
¥"	ダブルクォーテーション
¥b	バックスペース
¥f	改ページ
¥r	キャリッジリターン
¥n	改行
¥t	水平タブ
¥v	垂直タブ
¥0	NULL

上記の**赤い背景色**の部分はよく使うエスケープシーケンスである。

**インタラクタモード：**

```
>>> print("エスケープ場合===", "AAA¥'abc¥'BBB")
エスケープ場合=== AAA'abc'BBB
>>> print()

>>> print("raw文字列場合===", r"AAA'abc'BBB")
raw文字列場合=== AAA'abc'BBB
>>>
```

**スクリプトモード：**

下記を[ C:¥pythonPG¥Sample1\_4.py ] にプログラムしなさい。

```
1 # 例題ファイル：C:¥pythonPG¥Sample1_4.py
2
3 print()
4 print("円記号を表示します。：¥¥")
5 print(r"円記号を表示します。：¥¥")
6
7 print()
8 print("アポストロフィを表示します。：¥'")
9 print(r"アポストロフィを表示します。：'")
10
```

注：記号 [ r ] はraw文字列であり、役割は例題の実行結果で理解できる。

実行結果：

```
コマンドプロンプト
C:¥PythonPG>python Sample1_4.py

円記号を表示します。：¥
円記号を表示します。：¥¥

アポストロフィを表示します。：'
アポストロフィを表示します。：'
```

## 2 変数と式

中学校の数学で

$x = 2$  (整数)

$y = 3.145$  (実数)

のように変数を使ったことを覚えていることと思います。

Javaなどのようなコンパイル型言語では、上記のような使い方はできないが、スクリプト言語は可能である。

変数としては、大雑把に言うと、**型宣言なしに**整数と実数と(複素数)および文字列などとして使うことができる。

書式	変数名 = 値 変数名 = 他の変数名 変数名 = 式	# 変数に値を代入する。または、値を変更する。 # 変数(他の変数)を利用する。
例題	$x = 3$ $y = 3.145$ $z = \text{"おはよう!"}$ $a = x$ $b = y + 6.0$	# 変数に値を代入する。または、値を変更する。 # 実数 # 文字列 # 変数(他の変数)を利用する。 # 式 または 変数(他の変数)を利用する。
変数名規則	<ul style="list-style-type: none"> <li>1文字目は英文字またはアンダーバー( _ )</li> <li>1文字目以降は英数文字、または、アンダーバー</li> <li>大文字と小文字は区分できる</li> <li>使えない文字列がある。(使用済みの変数) import, print, for, if, else, try, class, and, or など</li> </ul>	

## 3 演算子 (四則・論理演算子など)

演算子の説明は分かりやすいように2個の変数の場合を示すが、中にはオブジェクトの場合にも成立するものもある。

$1+2$

演算子(+: operator)、オペランド(1,2: Operand) と言う。

主な演算子は次表ようになる。

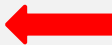

記号	名前	記号	名前
+	加算	=	代入
-	減算	>	より大きい
*	乗算	>=	以上
**	べき乗	<	未満
/	除算	<=	以下
//	切捨除算	==	等価
@	行列演算	!=	非等価
%	剰余	and	論理積
+	単項 +	not	論理否定
-	単項 -	if else	条件
~	補数 (ビット否定)	in	帰属検査
	ビット論理和	not in	非帰属検査
&	ビット論理積	is	同一性検査
^	ビット排他的論理和	is not	非同一性検査
<<	左シフト	lambda	ラムダ
>>	右シフト		

赤矢印の演算子は、Python特有である。

## (1) 四則演算子(+, -, \*, /, %, \*\*, //)

下記のように 2 個の変数に対する加算・減算などの演算子がある。

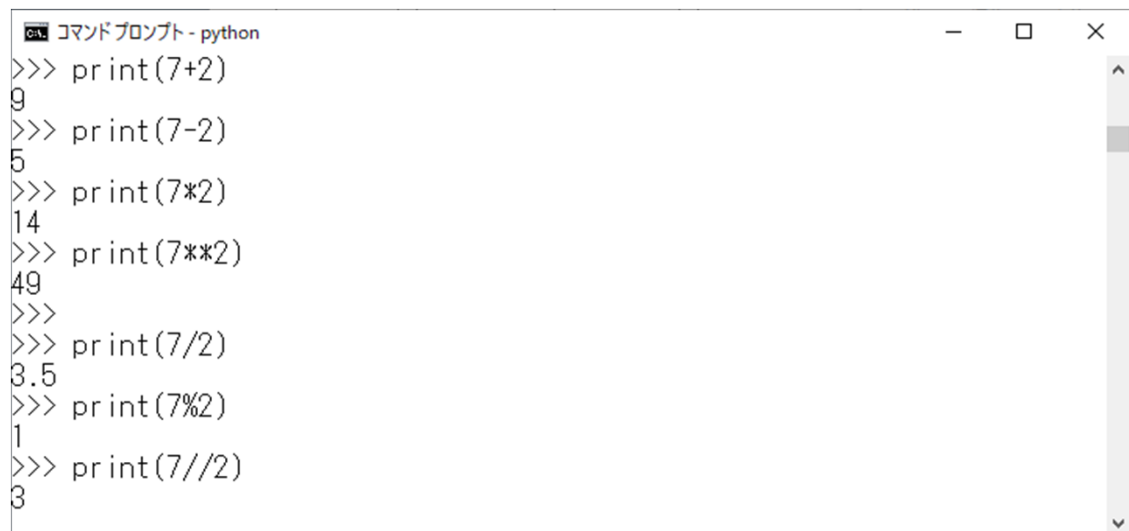
特に、プログラム特有は最下行の「商」「余り」の項である。

演算子	使い方	意味
+	$x + y$	加算
-	$x - y$	減算
*	$x * y$	乗算
**	$x ** y$ 	べき乗 $9 = 3^2 = 3 ** 2$
/	$x / y$	除算
数字の場合： $x, y$ が整数ならば $x / y = (\text{商}) \dots (\text{余り}) = (x // y) \dots (x \% y)$ $5 / 3 = (1) \dots (2)$		
//	$x // y$ 	商
%	$x \% y$	余り (整数の場合)

## インタラクタモード：

```
>>> print(7+2)
>>> print(7-2)
>>> print(7*2)
>>> print(7**2)

>>> print(7/2)
>>> print(7%2)
>>> print(7//2)
```



```

C:\> python
>>> print(7+2)
9
>>> print(7-2)
5
>>> print(7*2)
14
>>> print(7**2)
49
>>> 
>>> print(7/2)
3.5
>>> print(7%2)
1
>>> print(7//2)
3
  
```

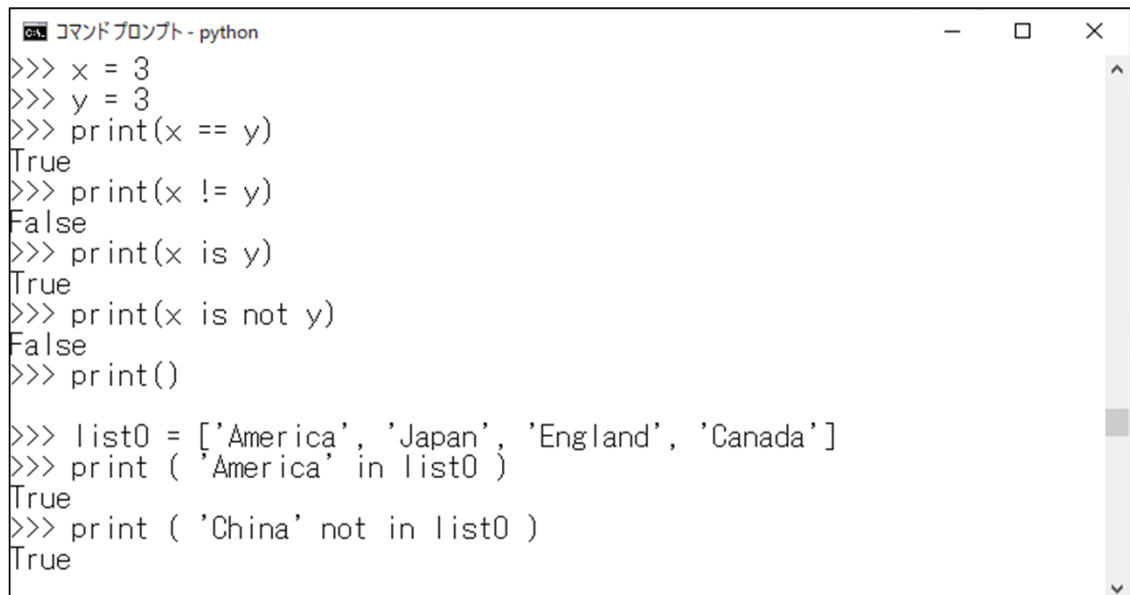
## (2) 比較演算子(==, !=, &lt;, &gt;, &lt;=, &gt;=, &lt;=&gt;, ==)

下記のように2個の変数を比較して、正しいならばtrue, 誤りならばfalseを出力する演算子がある。

演算子	使い方	意味 (出力がtrueとなる場合)
==	x == y	x が y と等しい
!=	x != y	x が y と異なる
<	x < y	x が y よりも小さい
<=	x <= y	x が y 以下である
>	x > y	x が y よりも大きい
>=	x >= y	x が y 以上である
is	x is y	x が y と等しい
is not	x is not y	x が y と異なる
in	x in y	x が y に含まれる (yは配列などを想定する)
not in	x not in y	x が y に含まれない (yは配列などを想定する)

## インタラクティブモード:

```
>>> x = 3
>>> y = 3
>>> print(x == y)
>>> print(x != y)
>>> print(x is y)
>>> print(x is not y)
>>> print()
>>> list0 = ['America', 'Japan', 'England', 'Canada']
>>> print ( 'America' in list0 )
>>> print ( 'China' not in list0 )
```



```
コマンドプロンプト - python
>>> x = 3
>>> y = 3
>>> print(x == y)
True
>>> print(x != y)
False
>>> print(x is y)
True
>>> print(x is not y)
False
>>> print()

>>> list0 = ['America', 'Japan', 'England', 'Canada']
>>> print ( 'America' in list0 )
True
>>> print ( 'China' not in list0 )
True
```

**スクリプトモード：**

下記を [ C:\pythonPG\sample2120.py ] にプログラムしなさい。  
 なお、下記1行の配列の説明は後述する。

```

1 # 例題ファイル：C:\pythonPG\sample2120.py
2
3 x = 3
4 y = 3
5
6 print ( x == y )
7 print ( x != y )
8 # print ( x < > y )
9 print ( x < y )
10 print ( x <= y )
11 print ( x > y )
12 print ( x >= y )
13 print ( x is y )
14 print ( x is not y )
15
16 print ( '*****' )
17 list0 = [ 'America', 'Japan', 'England', 'Canada' ]
18 list = 'America'
19 print ( list in list0 )
20 list = 'China'
21 print ( list not in list0 )

```

実行結果を下記に示す。すべて (true/false) での出力になっている。

**実行**

```

コマンドプロンプト
C:\PythonPG>python Sample2120.py
True
False
False
True
False
True
True
False
*****
True
True

```



## (3) 論理演算子(and, or, not)

下記のように2個のブール(true or false)に対して、  
コンピュータ概論の論理回路で示した、論理積・論理和・論理否定(出力 : true or false )の演算子がある。

変数 $a$	変数 $b$	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

(a) 論理積 (AND)

変数 $a$	変数 $b$	$a + b$
0	0	0
0	1	1
1	0	1
1	1	1

(b) 論理和 (OR)

変数 $a$	$\bar{a}$
0	1
1	0

(c) 論理否定 (NOT)

ただし、0=false, 1=true として読み替える。  
この表の結果を出す演算子はつぎのようになる。

演算子	使い方	意味	
and	x and y	論理積 : x も y も真であれば真	&& 
or	x or y	論理和 : x または y が真であれば真	
not	not x	論理否定 : x が偽であれば真	

インタラクタモード :

```
>>> x = 1
>>> y = 2
>>> z = 3

>>> print( (x==2) and (x==z) )
>>> print( (x==2) or (x!=z) )
>>> print( (x==2) or not(x!=z) )
```

```

C:\> python
>>> x = 1
>>> y = 2
>>> z = 3
>>>
>>> print( (x==2) and (x==z) )
False
>>> print( (x==2) or (x!=z) )
True
>>> print( (x==2) or not(x!=z) )
False

```

## (4) ビット演算子(~, &amp;, |, ^, &lt;&lt;, &gt;&gt;)

ビット演算子には下記のようなものがある。

演算子	使い方	意味
~	~x	ビット反転
&	x & y	AND:論理積(xもyも1のビットが1)
	x   y	OR:論理和(xまたはyが1のビットが1)
^	x ^ y	XOR:排他的論理和(xまたはyが1のビットが1)
<<	x << y	y ビット左シフト x = 2; print( x << 2 ) # 8
>>	x >> y	y ビット右シフト x = 8; print( x >> 2 ) # 2

$( )_{10}$  は10進数の表現を表す。  $( )_2$  は2進数の表現を表す。


10進数 $x=(2)_{10}$ の2進数表現 $x=(10)_2 \Rightarrow x \ll 2$  は  $(1000)_2 = (8)_{10}$  である。

$x \gg 2$  は 2ビット右に移動する。

10進数 $x=(8)_{10}$ の2進数表現 $x=(1000)_2 \Rightarrow x \gg 2$  は  $(10)_2 = (2)_{10}$  である。

## インタラクタモード：

```
>>> x=8
>>> print()
>>> print(x<<2)
>>> print( x*4)
>>> print(x>>2)
>>> print( x/4)
```



```
コマンドプロンプト - python
>>> x=8
>>> print()
>>> print(x<<2)
32
>>> print( x*4)
32
>>> print(x>>2)
2
>>> print( x/4)
2.0
```

## (5) 代入演算子(=, +=, -=, \*=, /=, %=, \*\*=, //=, &amp;=, |=, ^=, &lt;&lt;=, &gt;&gt;=)

代入演算子には下記のようなものがある。

注：他の言語のような++ や -- の演算子がない。

演算子	使い方	意味
=	x = y	x に y を代入する
+=	x += y	x = x + y に同じ
-=	x -= y	x = x - y に同じ
*=	x *= y	x = x * y に同じ
/=	x /= y	x = x / y に同じ
%=	x %= y	x = x % y に同じ
**=	x **= y	x = x ** y に同じ
//	x //= y	x = x // y に同じ
&=	x &= y	x = x & y に同じ
=	x  = y	x = x   y に同じ x ^= y # x = x ^ y に同じ
<<=	x <<= y	x = x << y に同じ
>>=	x >>= y	x = x >> y に同じ

インタラクタモード：

```
>>> x = 5; y = 3; x += y; print(x)
8
>>> x = 5; y = 3; x -= y; print(x)
2
>>> x = 5; y = 3; x *= y; print(x)
15
>>> x = 5; y = 3; x /= y; print(x)
1.6666666666666667
>>> x = 5; y = 3; x %= y; print(x)
2
```

```

C:\> python
Python 3.10.0 Shell
>>> x = 5; y = 3; x += y; print(x)
8
>>>
>>> x = 5; y = 3; x -= y; print(x)
2
>>>
>>> x = 5; y = 3; x *= y; print(x)
15
>>>
>>> x = 5; y = 3; x /= y; print(x)
1.6666666666666667
>>>
>>> x = 5; y = 3; x %= y; print(x)
2

```

## (6) 条件演算(if else)(三項演算子)

条件演算(if else) の書式はつぎのようになる。

書 式	(条件がtrueの時の出力される値) if (条件) else (条件がfalseの時の出力される値)
例 題	<pre>x=2 y = 222 if (x==2) else 111 print ( y ) # 222が表示される。</pre>

インタラクタモード：

```
>>> x = 3
>>> y = 222 if (x == 2) else 111
>>> print( y )
```

```

C:\> コマンドプロンプト - python
>>> x=3
>>> y=222 if (x==2) else 111
>>> print(y)
111

```

## (7) 演算子の優先順位

演算子の優先順位は、中学・高校で習ったこととほぼ同じである。

$(1+2)*4/2 - 2$

の場合には、()を先に計算し、つぎに\*を計算し、/を計算し、最後に-を計算する。

## 4 特殊な演算子（関数）：range, スライス

## (1) 範囲：range()関数

range( , )関数はずぎのように連続した数を表すことができる。

書 式	意 味
range(n, m)	数学では[n, m)のことである。n, n+1, ..., m-1まで すなわち、range(3, 5) とは 3, 4 ということです。
range(n, m, d)	n, n+d, n+2d, ... < mまで
range(m, n, -d)	m, m-d, m-2d, ... > nまで
例 題	range(2,10)    # 2,3,4,5,6,7,8,9 range(0,10,2) # 0, 2, 4, 6, 8 range(10,0,-2) # 10, 8, 6, 4, 2

インタラクタモード：

```
>>> for nn in range(2,10): print(nn)
>>> for nn in range(2,10,2): print(nn)
```

```

C:\> python
Python 3.10.0 Shell
>>> for nn in range(2,10): print(nn)
...
2
3
4
5
6
7
8
9
>>> for nn in range(2,10,2): print(nn)
...
2
4
6
8
>>>
  
```

## スクリプトモード :

下記を [ C:\pythonPG\sample2191.py ] にプログラムしなさい。

range(2,10)==>数値 2 から + 1 ずつ数値10未満まで == [2,3,4,5,6,7,8,9]

range(0,10,2)==>数値 2 から + 2 ずつ数値 9 未満まで == [2,4,6,8]

実行

```

1 # 例題ファイル : C:\pythonPG\sample2191.py
2
3 a = range(2, 10) # a = [2, 3, 4, 5, 6, 7, 8, 9]
4 print()
5 for element in a:
6     print( element, end=", " )
7
8 print()
9 for element in range(2, 10, 2):
10     print( element, end=", " )
11

```

実行結果 :

```

コマンドプロンプト
C:\PythonPG>python sample2191.py

2, 3, 4, 5, 6, 7, 8, 9,
2, 4, 6, 8,

```

## (2) スライス

文字列やリストなどは

文字列abcde ==> 数学：ベクトル(a, b, c, d, e),

リスト[1,2,3,4,5] ==> 数学：ベクトル(1, 2, 3, 4, 5)

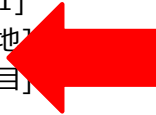
のように配列（シーケンス）型のデータである。

スライス[? : \*]はオブジェクトの要素に関する範囲（文字列/配列の(?+1)番目から\*番目）を表わすのに用いる。なお、番地では(?番地から\*-1番地)である。

参考： a=[ 12, 21, 322, 456, 531]

a=[0番地,1番地,2番地,3番地, 4番地]

a=[1番目,2番目,3番目,4番目, 5番目]



範囲の数値は、つぎのような番地のつけ方、つまり、

文字列abcde ==> CPUメモリ上のaddressで言うと、0番地a, 1番地b, ..

リスト[1,2,3,4,5] ==> CPUメモリ上のaddressで言うと、0番地1, 1番地2, ..

を用いる

書 式	# オブジェクト：リスト・文字列のオブジェクト(変数名) # 文字列・配列を作成する
オブジェクト[n:m]	# (n+1)番目から(m)番目まで取得 例： "abcde"[1,4]==>"bcd"
オブジェクト[-n:-m]	# 最後からn番目から最後から(m-1)番目まで取得
オブジェクト[n:]	# (n+1)番目から最後まで取得
オブジェクト[:m]	# 最初から(m)番目まで取得
オブジェクト[:]	# 最初から最後まで取得
オブジェクト[n]	# (n+1)番目を取得
オブジェクト[-m]	# 最後から(m)番目を取得
オブジェクト[:n]	# 最初からn文字ごと
オブジェクト[:-n]	# 最後からnステップごと後ろから取得
オブジェクト[::-1]	# 後ろから順に取得
説 明	str = "0123456789" 開始番地= 0の場合, "0"から      開始番地= 3の場合, "3"から 終了番地=-3の場合, "6"まで      終了番地=-2の場合, "7"まで

注： A = [0,1,2,3,4,5,6]

print( A[5,10] ) ==> [5, 6] エラーにならない

インタラクタモード：

```
>>> list0 = [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
>>> print( list0[1:5] )
>>> print( list0[:5] )
>>> print( list0[5:] )
>>> print()
>>> print()
>>> str0 = 'ABCDEFGFG'
>>> print( str0[1:5] )
>>> print( str0[:5] )
>>> print( str0[5:] )
```

```

cmd コマンドプロンプト - python
>>> list0 = [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
>>> print( list0[1:5] )
[1, 2, 3, 4]
>>> print( list0[ :5] )
[0, 1, 2, 3, 4]
>>> print( list0[5: ] )
[5, 6, 7, 8, 9]
>>> print()

>>> print()

>>> str0 = 'ABCDEFGH'
>>> print( str0[1:5] )
BCDE
>>> print( str0[ :5] )
ABCDE
>>> print( str0[5: ] )
FG

```

### スクリプトモード：

下記を [ C:\pythonPG\sample2192.py ] にプログラムしなさい。

オブジェクトは文字列でもリストでも大丈夫であるが、ここでは、文字列で例題を扱う。

"abcdefgh"[2:6] ==> "cdef" 3番目c から d, e, f である。

"abcdefgh"[2::2] ==> "ceg" 3番目c から e, g である。

"abcdefgh"[::-1] ==> "hgfedcba" 逆順の文字列である。

"abcdefgh"[2::-1] ==> "cba"

実行

```

1 # 例題ファイル：C:\pythonPG\sample2192.py
2
3 print( "abcdefgh"[2:6] ) # "cdef"
4 print( "abcdefgh"[2::2] ) # "ceg"
5 print( "abcdefgh"[::-1] ) # "hgfedcba"
6 print( "abcdefgh"[2::-1] ) # "cba"
7

```

実行結果：

```

cmd コマンドプロンプト
C:\PythonPG>python sample2192.py
cdef
ceg
hgfedcba
cba

```



## 5 分岐制御

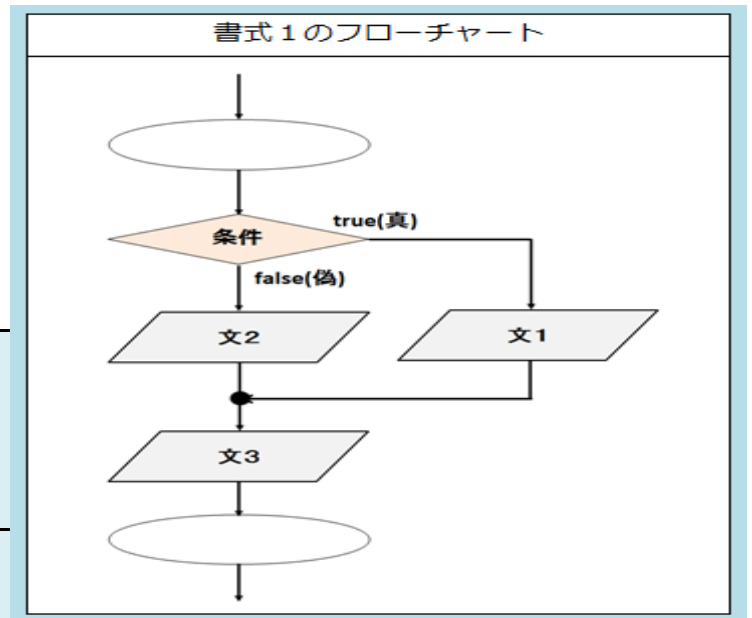
分岐処理として、if文、while文、case文を扱う。

## (1) if-else文

if文の一般的なフローチャートは右図のようになる。

また、書式は下記である。

書式 1	if 条件: 文 1 else: 文 2 文 3
書式 1 (特殊ケース)	if 条件: 文 1 文 3
書式 1 の例題	x=3 if x>=5: print("%s は 5以上" %(str(x))) else: print("%s は 5未満" %(str(x))) print("end if")



条件の部分には、演算子で示した比較演算子を使う。

比較演算子：

演算子	使い方	意味 (出力がtrueとなる場合)
==	x == y	x が y と等しい
!=	x != y	x が y と異なる
<	x < y	x が y より小さい
<=	x <= y	x が y 以下である
>	x > y	x が y より大きい
>=	x >= y	x が y 以上である
is	x is y	x が y と等しい
is not	x is not y	x が y と異なる
in	x in y	x が y に含まれる (yは配列などを想定する)
not in	x not in y	x が y に含まれない (yは配列などを想定する)

**インタラクタモード：**

入力の仕方の注意；

```
>>> x = 2
>>> y = 3
>>> if x == y :
...     print("真")
... else:
...     print("偽")
```

```
C:\> コマンドプロンプト - python
>>> x = 2
>>> y = 3
>>> if x == y :
...     print("真")
... else:
...     print("偽")
...
偽
```

**スクリプトモード：**

下記を [ C:\pythonPG\sample2211.py ] にプログラムしなさい。

もし、数学と英語も80点以上なら、  
     "総合評価：A"と表示し、  
 それ以外なら  
     "総合評価：B"と表示する。

仮定： math = 82  
       english=90

```
1 # 例題ファイル：C:\pythonPG\sample2211.py
2
3 math = 82
4 english=90
5
6 if math >= 80 and english >= 80:
7     print("総合評価：A")
8 else:
9     print("総合評価：B")
10
```

実行結果：

```
C:\> コマンドプロンプト
C:\PythonPG>python sample2211.py
総合評価：A
```

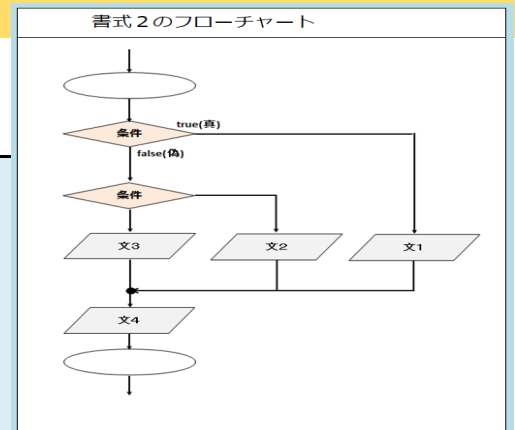
## (2) if-elif-else文

右図のようなif文の書式は下記である。

elif

に  
注意:

書式 2	if 条件: 文 1 elif 条件: ← 文 2 else: 文 3 文 4
書式 2 の例題	<pre> x=3 if x&gt;=5:     print("%sは 5 以上" %(str(x))) elif x&gt;=3:     print("%sは 3 以上 5 未満" %(str(x))) else:     print("%sは3未満" %(str(x))) print("end if") </pre>



インタラクタモード:

```

>>> x = 2
>>> y = 3
>>> if x == y :
...     print("if_true")
elif x == (y-1) :
...     print("else_if_true")
else:
...     print("else_false")

```

```

コマンドプロンプト - python
>>> x = 2
>>> y = 3
>>> if x == y :
...     print("if_true")
... elif x == (y-1) :
...     print("else_if_true")
... else:
...     print("else_false")
...
else_if_true

```

**スクリプトモード：**

下記を [ C:\pythonPG\sample2212.py ] にプログラムしなさい。

もし、数学と英語も80点以上なら、  
    "総合評価：A"と表示し、  
もし、数学が80点以上かつ英語が70点以上  
または、数学が70点以上かつ英語が80点以上  
    "総合評価：B"と表示し、  
それ以外なら  
    "総合評価：C"と表示する。

仮定： math = 82  
      english=70

```
1 # 例題ファイル：C:\pythonPG\sample2212.py
2
3 math = 82
4 english=70
5 flg1 = (math >= 80 and english >= 70)
6 flg2 = (math >= 70 and english >= 80)
7
8 if math >= 80 and english >= 80:
9     print("総合評価：A")
10 elif flg1 or flg2:
11     print("総合評価：B")
12 else:
13     print("総合評価：C")
14
```

実行結果：

```
コマンドプロンプト
C:\PythonPG>python sample2212.py
総合評価：B
```

**(4) switch(case)文**

pythonにはJava言語などに存在するswitch文は存在しない。  
pythonでは、if-elif-else文で作成する。

## 6 繰り返し制御

## (1) for文

for文はある処理を繰り返し(イテレーション)て実行する場合に適している。  
つまり、その中に記述した処理ブロックを決められた回数だけ繰り返して処理する。  
他の言語のように

```
for(int j=0; j<100; j++){ /* 処理 */ }
```

のような構文はエラーになります。

書式 1	for 変数名 in オブジェクト(List, String) 処理ブロック (複数行可)	要素の個数のみ繰り返す 繰り返し中の処理
書式 2	for 変数名 in オブジェクト(List, String, など) 処理ブロック (複数行可) else: 処理ブロック (複数行可)	要素の個数のみ繰り返す 繰り返し中の処理  for文を終了するときに実行される処理

(A) 例題 1 : List型 + 終了時の処理なし

```
for element in [ "みかん", "なし", "りんご" ]:  
    print ( element + ", ", end="" )
```

(B) 例題 3 : String型 + 終了時の処理なし

```
for element in "ELEMENT":  
    print ( element + ", ", end="" )
```

(C) 例題 4 : range型 + 終了時の処理なし

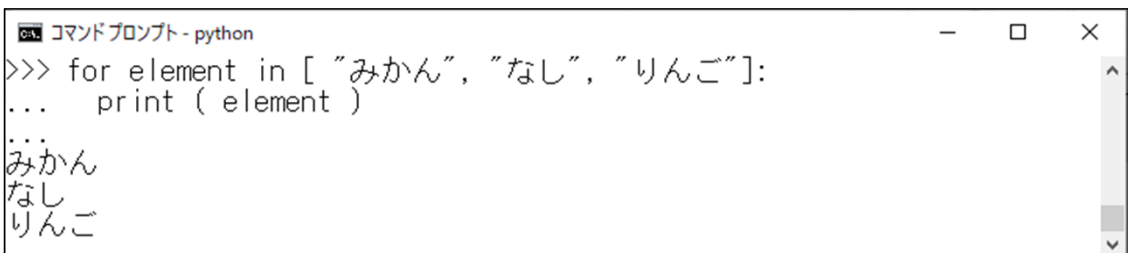
```
for element in range(0, 5):  
    print ( str(element) + ", ", end="" )
```

これが、他言語で見られる for文 である。つまり

```
for( int j1=0; j1<5; j1++){ 表示文 ; }
```

インタラクタモード :

```
>>> for element in [ "みかん", "なし", "りんご" ]:  
    print ( element )
```



```

C:\> コマンドプロンプト - python
>>> for element in [ "みかん", "なし", "りんご" ]:  
...   print ( element )  
...  
みかん  
なし  
りんご

```

## (E) 例題プログラムと実行

下記を [ C:\pythonPG\sample2310.py ] にプログラムしなさい。

```

1 # 例題ファイル：C:\pythonPG\sample2310.py
2
3 print()
4 print ( "例題1:[ 'みかん', 'なし', 'りんご']" )
5 for element in [ "みかん", "なし", "りんご"]:
6     print ( element + ", ", end="" )
7 print()
8 print()
9
10 print ( "例題2:[ 'みかん', 'なし', 'りんご'] "+ "else文" )
11 for element in [ "みかん", "なし", "りんご"]:
12     print ( element + ", ", end="" )
13 else:
14     print ( "繰り返し最後", end="" )
15 print()
16 print()
17
18 print ( "例題3:¥ELEMENT¥" )
19 for element in "ELEMENT":
20     print ( element + ", ", end="" )
21 print()
22 print()
23
24 print ( "例題4:range(0, 5)" )
25 for element in range(0, 5):
26     print ( str(element) + ", ", end="" )
27 print()
28

```

実行結果をつぎに示す。

```

コマンドプロンプト
C:\PythonPG>python sample2310.py

例題1:[ 'みかん', 'なし', 'りんご']
みかん, なし, りんご,

例題2:[ 'みかん', 'なし', 'りんご'] + else文
みかん, なし, りんご, 繰り返し最後

例題3:"ELEMENT"
E, L, E, M, E, N, T,

例題4:range(0, 5)
0, 1, 2, 3, 4,

```

## (2) while文

while文は、for文と異なり条件を満足する間、繰り返して処理する場合に使用する。  
条件がTrueである間は処理を繰り返し実行し続ける。

書式 1	while 条件式: 処理ブロック (複数行可)      # 条件が真の場合に処理
書式 2	while 条件式: 処理ブロック (複数行可)      # 条件が真の場合に処理 else: 処理ブロック (複数行可)      # 条件が偽の場合に処理
例題	# 例題ファイル : C:\pythonPG\sample2320.py  print() j = 0 while j<5: print ( str(j) + ", ", end="" ) j = j+1 else: print("write In else")  print("After ForEnd")

実行結果 :

```

C:\PythonPG>python sample2320.py

0, 1, 2, 3, 4, write In else
After ForEnd

```

下記を [ C:\pythonPG\sample2214.py ] にプログラムしなさい。  
sum = 1+2+3+...+100 の計算をする

```

1 # 例題ファイル : C:\pythonPG\sample2214.py
2
3 sum = 0
4 flg = True
5 no = 0
6
7 while flg:
8     sum += no
9     no += 1
10    if no > 100:
11        flg = False
12 print( "sum=", sum )
13

```

実行結果 :

```

C:\PythonPG>python sample2214.py
sum= 5050

```

```

flg = True
nn = 0
while flg:
    nn += 1
    if nn == 100:
        flg = False
    print(nn)

```

出力  
0,1,2,3,

出力と  
0,1,2,3,

## (3) break, continue文

for文 や while文 の繰り返し中に、

処理を中止したり、その後のループ処理のみ中止したりする命令語がある。

書 式	意 味
break	for文やwhile文中で使う。 Break文を実行するとそれ以降の処理は中止され、 for文・while文を終了する。
continue	for文やwhile文中で使う。 Break文を実行するとそれ以降のブロック処理はスキップされ、 次のfor文・while文に移る。

break, continue文に関する、つぎのプログラムを[C:\pythonPG\sample2330.py]に作成しなさい。

```

1 # 例題 ファイル : C:\pythonPG\sample2330.py
2
3 print()
4 j = 0
5 while j<5:
6     if j==3:
7         break
8     print ( str(j) + ", ", end="" )
9     j = j+1
10 else:
11     print("write In else")
12
13 print("After ForEnd with break at j==3")
14
15
16 print()
17 j = 0
18 while j<5:
19     if j==3:
20         j = j+1
21         continue
22     print ( str(j) + ", ", end="" )
23     j = j+1
24 else:
25     print("In else")
26
27 print("After ForEnd with continue at j==3")
28

```

実行結果：

```

C:\PythonPG>python sample2330.py
0, 1, 2, After ForEnd with break at j==3
0, 1, 2, 4, In else
After ForEnd with continue at j==3

```



```

x = 0
y = x^3 + 2x^2 + x + 5
x > -5
y=0を満足する最初の値 x、 y を小数点以下1位の精度で求めよ。
while文
x = x + 0.1

```

n		x
	x = 0	
1	x = x + 1	1
2	x = x + 2	3
3	x = x + 3	6
n	x = x + n	

```

x = 0
for n in range(0, 101):
    x = x + n

```

```

sum = 0
for n in range(0, 101):
    sum = sum + n

```

問題：

5の倍数は加算しない。  
sum > 700 を満足する最小のn, その sum を求めよ。

修正：

sum < 700 を満足する最大のn, その sum を求めよ。