2021 年 9 月

プログラミング（Python）

プログラミング初心者から活用まで　Ver1.3c

大原簿記情報専門学校 札幌校

佐々木博幸

目次

第 1 章

プログラミング言語 python 導入

1.1　プログラムとは？

プログラムとは日常では計画表を表す言葉としてよく知られています（運動会や行事のプログラムとして)。なにかのイベント（行事）を進めていくための段取りや手順、出演者などを記載しているものをイメージすることができます。

今回はコンピューターをなにかしらの目的を持って動作させるための手順という意味でプログラムを作ること（プログラミング）について学んでいきます。

各種業務処理における利用例について

- 電子カルテシステム
- 各種電子決済・電子マネー
- 鉄道系のシステム
- ゲーム
- etc.etc.

python 特徴の説明

- シンプル
- わかりやすい
- 誰が書いても同じようになりやすい
- 流行っている！

1.1.1　python インストール

今回は python 言語を学習する環境として Google Colaboratory を利用します。

この環境はブラウザ上で動作するためソフトウェアのインストールは不要です。

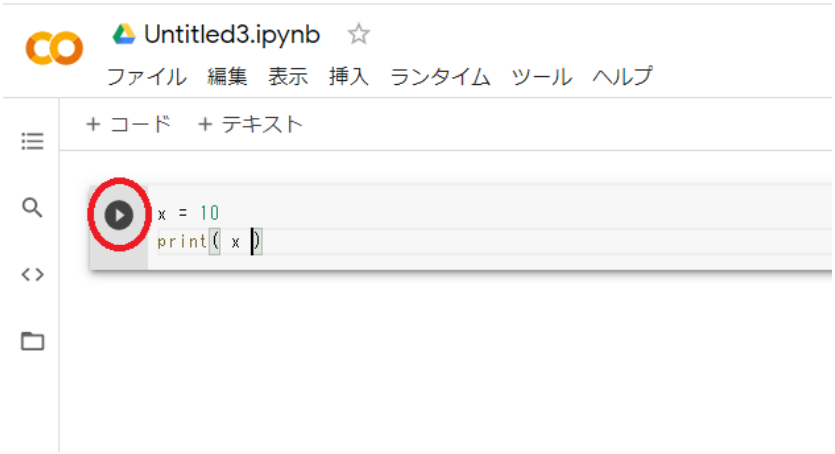
検索エンジンで”Colab”を入力して検索すると次のリンク先が表示されます。

リンク先をクリックすると次の画面が表示されます。

準備はこれにて完了です。

早速「ノートブックを新規作成」をしましょう。

ここで「コードセル」領域にプログラムを入力することができます。



実行ボタンを押して動作確認しましょう。少し待ち時間がかかりますが、無事結果が表示されます。

ここからは「コードセル」にプログラムを入力し実行結果を参照するを繰り返すことで Python のプログラムの理解を進めることができます。

第 2 章

プログラミング 基礎

2.1 コンピューターを使った命令の実行

2.1.1 命令や計算の実行方法

一つ一つ結果を確認して実行する方法と手順をたくさんまとめて実行する方法があります。
コンピューターとは電気電子の作用によって動作する汎用計算機と訳されます。
Colaboratory では一つ一つの命令を確認しながら「コードセルによる実行」ができます。

凡例：python の実行方法

Google Colaboratory ではコードセルに直接命令を入力することができます。

2.1.1: 実行

```
x = 10
print(x)
#ここで実行ボタンを押します。


10
```

2.1.2 python の考え方

python の特徴の説明でも書きましたが、python では他の言語と大きく異なる特徴があります。

- 1 行で 1 つの命令が基本
- 処理のまとまり（ブロック）がインデント（文左側空白）により決まる
- ライブラリが豊富（IoT や AI、画像解析、WebAPI 等）なので、ライブラリの使い方を覚えるだけでいろいろな処理ができる

凡例：他の言語の例

「C または Java」

```
    for(int i = 1 ,int total = 0; i <= 10; i++){total += i;}
```

「VisualBasic」

```
      if a = b then
   print "true"
        endif
```

「ShellScript」

```
    echo "facebookyahooapplegooglemicrosoftamazon" | sed 's/[a-z]\{5\}\
       (.\)....\(.\)..\(.\).\{13\}\(.\).\{5\}\(.\).*/\1\2\3\4\5/'
```

#統一感がありません

それでは python の基本文法を理解してから先に進めていきましょう。

- 空白（インデント）が重要
- 変数の宣言が不要

インデントの使い方を見てみましょう。

凡例：シンプルな python の命令

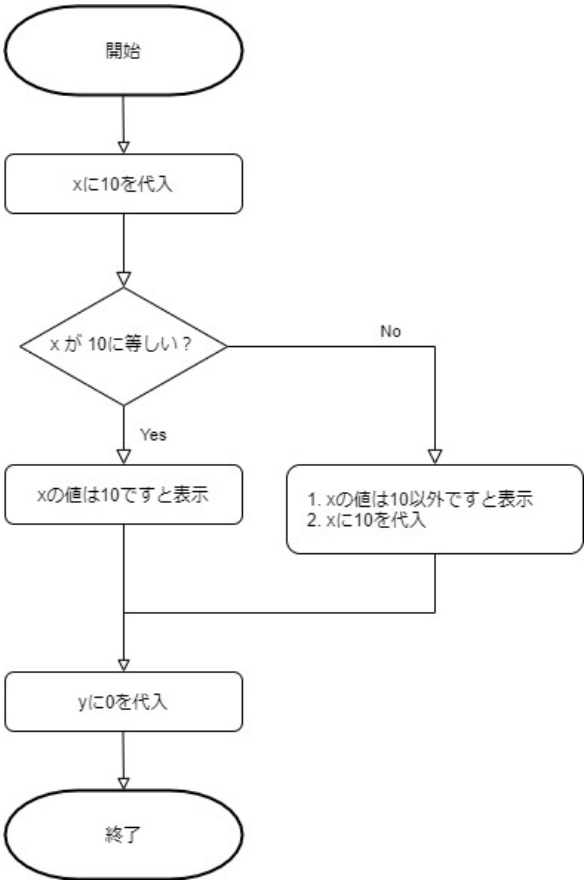
インデント（スペース・空白）には意味がありますので、必要のないスペースはエラーになります。

コード　2.1: インデントのよるエラーの有無

```
x = 10 #エラー無し
 y = 10 #左側にスペースが１文字空いているのでエラーになります
(unexpected indent)
```

python では#から右側はコメントとなり、処理には無関係でプログラムの説明等を記述します。

python では一連の処理をブロックという形で表現することになっています。ある条件が成立したときに行う処理と成立しなかったときに行う処理を分けて行う流れ図を見てブロック記述を見てみましょう。


開始
xに10を代入
x が 10に等しい？
No
Yes
xの値は10ですと表示
1. xの値は10以外ですと表示
2. xに10を代入
yに0を代入
終了

前述の流れ図を python で記述した場合、次のようになります。

凡例：python のブロック

ブロック開始「:」とインデントの組み合わせです。if 文で x の値が 10 であれば 10 と表示し、x の値が 10 以外であれば「10 以外」と表示し x の値を 10 に設定するプログラムの例

コード　2.2: インデントのよる処理の流れ

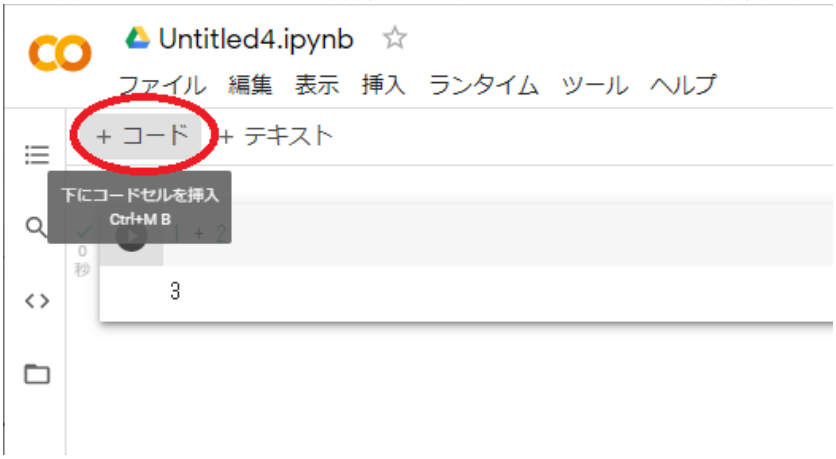
```
x = 10
if x == 10 : #x が 10 だったらの処理を書くためには「:」でブロックを開始
    print("x の値は 10 です")
    #他の処理はなにもしません。
else : #x の値が 10 以外だったらの処理 2 行が x の値が 10 以外の処理ブロック
    print("x の値は 10 以外です")
    x = 10 #x の値が 10 以外なので 10 に設定します。
y = 0 #これは if 文のどちらを通過しても必ず実行されます。
```

python では左側のインデントによってどこからどこまでが一連の処理か判断しています。

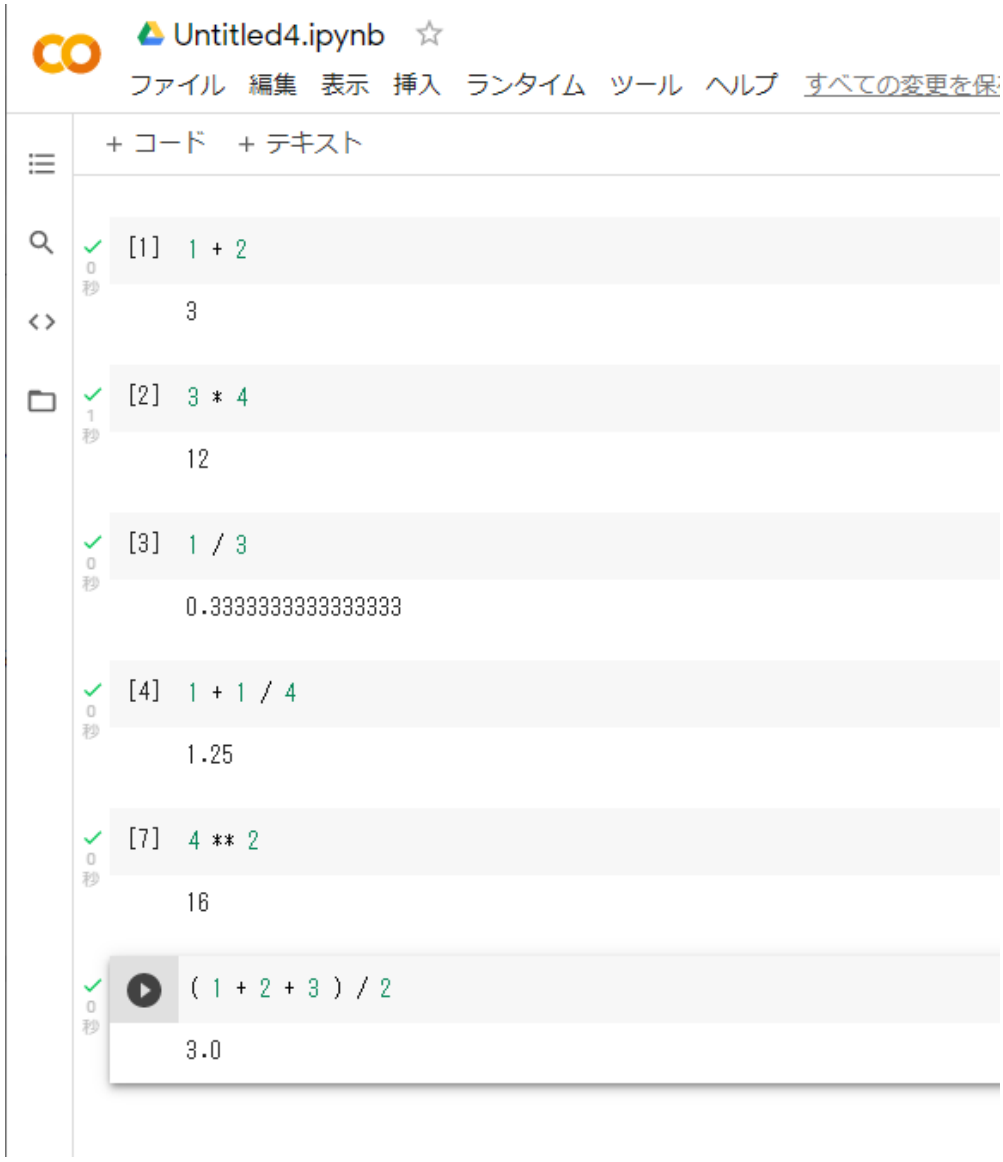
2.1.3　python を使った計算

まずは **コードセル**による計算を行ってみましょう。

次の例の通りコードセルに計算式を入力することですぐに計算結果を取得することができます。



2.1.2:　コードセルによる python 実行：例



ここまでのように算式を入力していくと正しい結果を得られることを確認することができます。

※ 計算結果が割り切れない場合に無限に続く少数ではないことを確認しておきましょう。　結果は有限桁で表現されています。

2.1.4　変数とは？

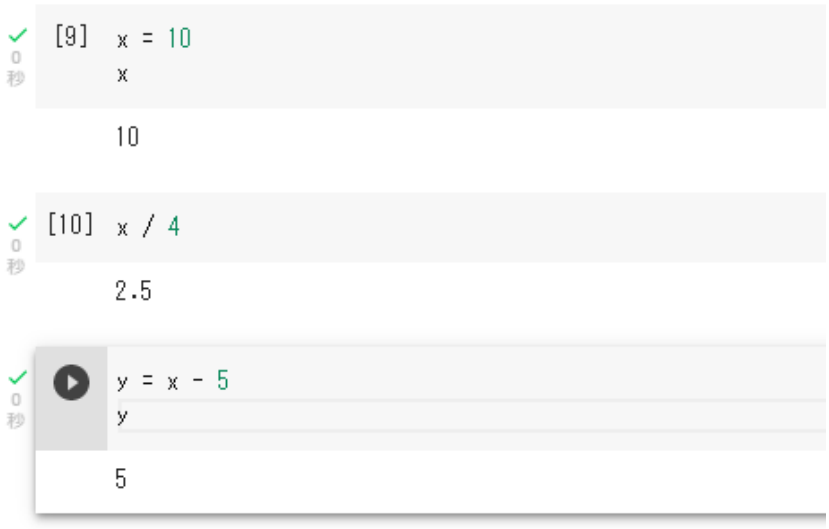
プログラムでは単純な計算以外に処理の結果を次に引き渡して連続したまとまった処理を行うことが多いです。

その際に変数といわれる計算結果を一時的に保存しておくためのデータ領域が使われます。

変数とは値を格納するもので数学の方程式で出てくる x,y などによく似ています。

早速先程の続きで変数に値を代入する方法を見てみましょう。

2.1.3:　変数の確認



最初の行の「x = 10」は x と 10 が等しいという意味ではなく左辺 x に右辺の 10 という数値を入れておくという意味になります。（代入といいます。）

2.1.5　変数の名前付けルール

それではここで変数に利用できる文字にはどのようなものがあるか次の表で確認しましょう。

表 2.1　変数に使える文字

a 〜 z のアルファベット（大文字も可）
0 〜 9 の数字
_（アンダースコア）

※　最初の文字を数字にすることはできません。

※　予約語/キーワード（if や for など・・・）と同じ名前は使えません。

先程の「=」は代入という考え方なので次のように左辺 10 に右辺 x を代入という考え方になってしまう式はエラーとなります。

2.1.4: イコールの考え方等式

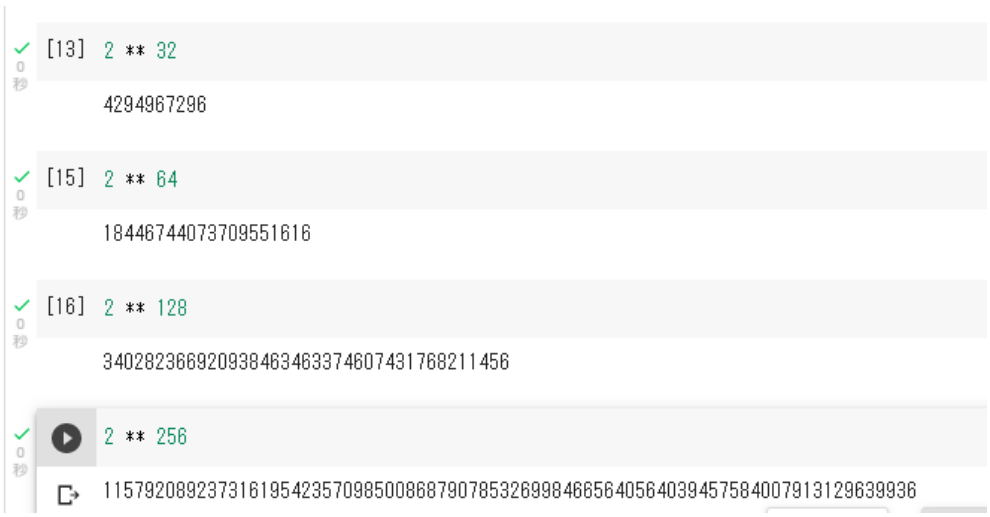
等式として等しいという意味ではないので右辺の計算結果を左辺に代入という解釈を行います。したがって右辺には左辺と同じ変数を利用して計算式を書くことができます。

※ =は右辺の式の値を左辺に代入という意味になります。

2.1.6 式と計算

電卓では簡単にできないこんな計算もできます。

2.1.5: 桁の多い計算



！？（無量大数を超えます）

ここまでであれば Excel でもできそうです。

Point

その他の計算、平方根（ルート）など一部の複雑な計算は別の仕組みを使います。したがってこのテキスト（授業）では触れません。

2.2　まとまった手順の実行

もっと手順について考えてみましょう。

それではまとまった手順を実行するスクリプト（簡易的な）プログラミングを行っていきましょう。

※本格的プログラミングは何千行・何万行・何十万行の命令を書いていきます。
１０万行のプログラムを１ページ１００行の印刷すると１０００ページです。（小規模なプログラムでもこのくらいの量になります）

2.2.1　一連の処理

1〜5 までの合計（1 + 2 + 3 + 4 + 5）を計算するプログラムを実行しましょう。

2.2.1:　同じような手順の繰り返し

```
[18] x = 1
     x = x + 2
     x = x + 3
     x = x + 4
     x = x + 5
     x
```

```
15
```

ここではプログラムで画面に出力する方法を学びます。いままでも **コードセルによる実行**で結果の出力はできていましたが、明示的に変数や文字列、計算結果を出力する方法を学びます。

2.2.2　画面に出力する方法

結果を出力する print 文を使ってみます。

2.2.2:　出力命令

```
print(x)
```

```
15
```

```
x
```

```
15
```

「print(x)」も「x」も x の値が出力されます。違いについては次のセクション「コードセルを使った出力」の中で説明します。

2.2.3　コードセルを使った出力

これらのプログラムをひとまとまりの処理としてコードセルに記述してみましょう。

2.2.3.1　コードセルで複数の出力の作成

2.2.3:　プログラム名（python1-1.py）

```
x = 1
x = x + 2
x
x = x + 2
print(x)
```

```
5
```

実行するとコードセルの最後に記述された「print(x)」の値だけが出力されることが分かります。

明示的に文字列や計算結果、変数の内容を出力するには print 関数を利用します。

まとまった手順をプログラムとして書いた場合、結果を出力する手続きを実行しなければ出力されません。プログラムで結果を出力する方法は「print 関数」を利用することを覚えておきましょう。

※ いままで実行結果として変数名だけで内容が表示された理由は式の評価した値が結果として

表示されたためです。
（式の）評価についてはもう少し先に進めてから確認します。

2.2.3.2 練習問題

練習問題 ： Practice 1

底辺 x を 3cm、高さ y を 4cm とする三角形の面積を求め出力するプログラムを作成しましょう。

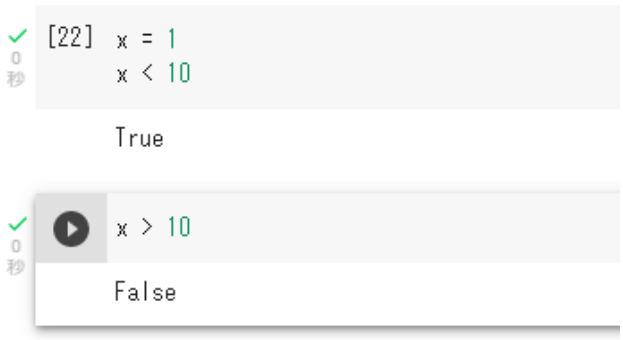
2.3 変数と式の評価

プログラムを作成すると様々な条件式によって異なる動作をすることが求められます。ここでは条件判定に必要な式とその評価について説明します。

2.3.1 式の評価

ここで式（条件式）の評価について学びます。（まずは不等号から）

2.3.1: 式の評価



※ 条件式が成立していると「True」が不成立であれば「False」になっていることがわかります。

条件式で利用される主な比較演算子

表 2.2　条件式（比較演算子）

記号	意味
==	”等しい”
<	”小なり”
>	”大なり”
<=	”小なりイコール”
>=	”大なりイコール”
!=	”等しくない”

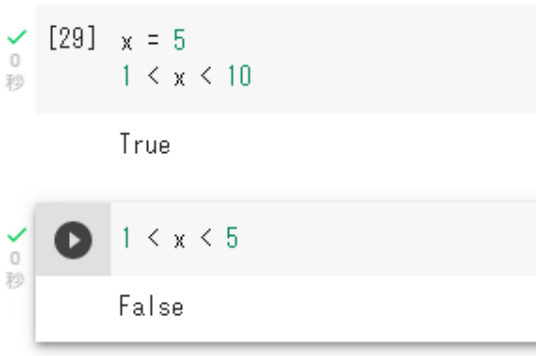
それでは条件式を使った式の評価を見ていきましょう。

2.3.2:　式の評価（条件式１）

```
[24] x <= 1
     True

[25] x >= 2
     False

[26] x == 1
     True

[27] x == 2
     False

     x != 1
     False
```

他の言語では書けないことが多い次のような条件式も可能です。

2.3.3: 式の評価（条件式３）



※ 他の言語では「1 < x and 1 < 10」など「かつ」条件で記入することが多いです。
文字や文字列も比較できます。

2.3.4: 式の評価（条件式４）

```
[31] a = "z"
     b = "z"
     a == b
```

```
True
```

```
[32] c = "y"
     a == c
```

```
False
```

```
a > c # "z" > "y" を辞書順で評価します。
```

```
True
```

```
[34] a < c
```

```
False
```

```
[35] a = "abc"
     b = "abc"
     c = "azc"
     a == b
```

```
True
```

```
[36] a == c
```

```
False
```

```
[37] a > c # "abc"と"azc"の辞書順の評価
```

```
False
```

```
a < c
```

```
True
```

今まで見てきたように **コードセルによる実行**では最後の評価結果を表示しますので、コードセルの途中で式の評価の値を出力したい場合は print 命令によって明示的に出力する必要があります。

2.3.2　if 文（場合分けの処理）

それでは評価を使った場合分け「if 文」を利用してみましょう。

条件が成立した (式の評価が真) のときに行う手続き、不成立（式の評価が偽）のときに行う手続きをまとめて記述します。

2.3.5:　if 文の書き方１

```
1   if 条件式:
2       真の手続き１
3       真の手続き２
4        ・
5        ・
6   else:
7       偽の手続き１
8       偽の手続き２
9        ・
10       ・
```

Point

「python」ではまとまった手順を「ブロック」といい、共通のインデント（左端からの字下げ）で表します。ブロックを書くときは「行末」に**「：」**を書く必要があります。

※「真の手続き」「偽の手続き」は「タブ」や「スペース」で字下げ（インデント）することで複数の手続きをひとまとまりのブロックにすることができます。

それでは if による場合分けを確認しましょう。

凡例：if-1

変数 x に学生 A さんの数学の得点を入力します（今回は 60 点とします)。このテストでは 50 点以上を合格としますので、A さんの数学の得点が合格なのか不合格なのか判定して出力するプログラムを作りましょう。

コード　2.3: if-ex.py

```
x = 60
if x >= 50:
    print ( " goukaku " ) # 左側のインデント空白は
                          #  「Tab」キー（1回）で入力します。
else:
    print ( " fugoukaku " )
#インデントなしで「Enter」キーを押すと式を評価して結果を表示します。
```

```
goukaku
```

print 命令で文字を出力するときは「"」(ダブルクォーテーション) や「'」(シングルクォーテーション）で文字を囲む必要があります。(変数と区別するためです。)

if 文の意味が理解出来たらまとまった手順としてプログラムを作ってみましょう。

2.3.2.1　練習問題

練習問題　：　Plactice 2

「凡例 if-1」を実行してみましょう

他にも if 文の書き方はいくつかありますので、確認してみましょう。

if 文では真の手続きのみ必要で偽の手続きを記載する必要の無い場合があります。その時の if 文は書き方 2 を確認してください。

2.3.6: if 文の書き方２（偽の手続きがない場合）

```
if 条件式:
    真の手続き１
    真の手続き２
    ・
```

このときのブロックは複数行書くことができます。if 文が終了後はインデント無しで次の行から命令を書いていきます。

また、if 文には１つの条件が偽の場合２つ目、３つ目と順番に条件判定を行いいずれかが真の場合に実行する処理を記述しますが、すべての条件に当てはまらない時の実行も書くことができます。

2.3.7: if 文の書き方３（条件１にあっているとき、条件２にあっているとき・・・・）

```
if 条件式１:
    条件式１が真の手続き
    ・
elif 条件式２:
    条件式１が偽で条件式２が真の手続き
    ・
elif 条件式３:
    条件式１～２が偽で条件式３が真の手続き
    ・
else:
    条件１～３が偽の時の手続き
    ・
```

それでは if を使って判定するプログラムを作ってみましょう。

凡例：if-2

変数 x に学生 A さんの数学の得点を入力します（今回は 60 点とします)。このテストでは 80 点以上を「A」とし、60 点以上を「B」とし、40 点以上を「C」、40 点未満を「D」と評価します。A さんの数学の得点の評価を判定して出力するプログラムを作りましょう。

コード　2.4: puthon2-1.py

```
1   x = 60
2   if x >= 80:
3       print ( "A" )
4   elif x >= 60:
5       print ( "B" )
6   elif x >= 40:
7       print ( "C" )
8   else:
9       print ( "D" )
```

保存し if 文の動作を確認してみましょう。
※　x に代入されている得点を A,B,C,D それぞれに該当する値にして実行してみましょう。

if 分だけで完結するのではなくその後のプログラムに値を引き渡すプログラムを作成してみましょう。

凡例：if-3

凡例「if-2」を評価メッセージ A 判定なら「”excellent”」、B 判定なら「”good”」、C 判定なら「”passing”」、D 判定なら「”failing”」と変数 mes に代入し出力するプログラムを作成しましょう。

コード　2.5: puthon2-2.py

```
1   x = 60
2   if x >= 80:
3       mes = "excellent"
4   elif x >= 60:
5       mes = "good"
6   elif x >= 40:
7       mes = "passing"
8   else:
9       mes = "failing"
10
11  print ( mes )
```

※　x に代入されている得点を A,B,C,D の判定にそれぞれに該当する値にして実行してみましょう。

2.3.2.2　練習問題

練習問題　：　Plactice 3

変数 x と変数 y にそれぞれ整数を代入して x の値が大きければ”x is large.”と表示し、y の値が大きければ”y is large.”と表示、等しければ”x and y are equal.”と表示するプログラムを作りましょう。
ここで x と y はプログラム中でそれぞれ値を入れて if 文が正しく動作することを確認しましょう。

2.3.3　if 文（複合条件）

複数の条件が関係する条件式も利用することができます。

例えばテストの点数が 0 点以上と 100 点以下であることを判定するための条件（複合条件と言います）の記載方法もあります。

2.3.8:　条件式の書き方（複合条件）

1．条件式 1 と条件式 2 がともに真の時「かつ」

条件式 1 and 条件式 2

2．条件式 1 と条件式 2 のどちらかが真の時「または」

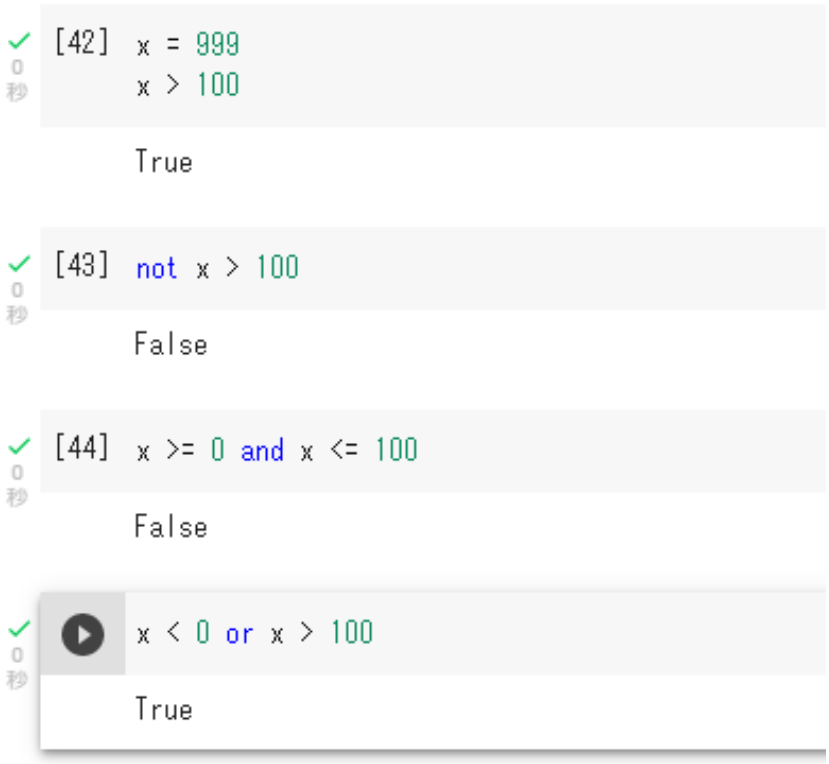
条件式 1 or 条件式 2

3．条件式の判定が偽の時「否定」（わかりにくいですが）

not 条件式

コードセルで「and or not」条件の評価を確認しましょう。

2.3.9:　and or not の式の評価



※式の評価について確認ができたと思います。

それでは if 文を使った練習問題をやってみましょう。

2.3.3.1　練習問題

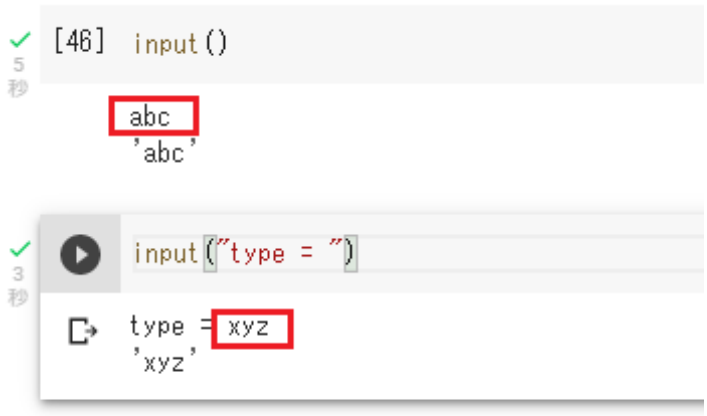
練習問題　：　Plactice 4

「凡例 if-2」を 101 点以上、0 点未満の場合「error」と表示されるプログラムに変更してください。

2.4　キーボードからの入力

プログラムを実行時に変数に値をキーボードから入力した値に実行中に設定してみましょう。このような値の代入を動的な値の代入といいます。

2.4.1:　キーボードからの入力



input 関数を使うとキーボードから入力された値を取得することができます。※　この例では入力した文字が評価されることで表示されたということになります。

入力した文字列を変数 s に入れ、変数 s の値を出力します。

2.4.2: キーボードから変数への代入

```
s = input(" type = ")
print(s)
```

```
 type = xyz
xyz
```

※変数 s に文字列が入力されていることを確認できました。

2.4.1 文字列の取り扱いについて

文字列は「"」(ダブルクォーテーション) や「'」(シングルクォーテーション) で囲まれ数値とは異なりこのままでは数値計算を行うことはできません。

先の例で変数 s に格納された文字列を利用し文字と数値の違いを確認してみましょう。

2.4.3: 文字列の取り扱い

```
[49] s * 2
```

```
'xyzxyz'
```

```
[50] s + 10
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-50-30fc4c06cb2b> in <module>()
----> 1 s + 10

TypeError: can only concatenate str (not "int") to str

SEARCH STACK OVERFLOW
```

```
s + "10"
```

```
'xyz10'
```

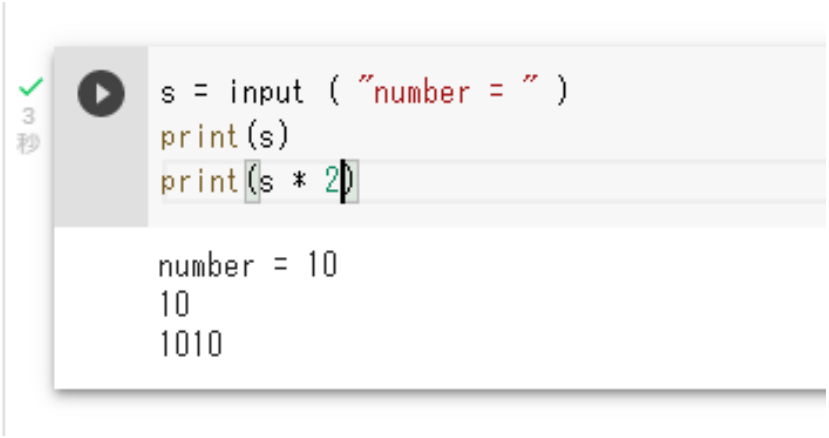
文字列に * 2 とすることで同じ文字列を 2 回繰り返す意味になります。 + 10 は文字列に対して

の算術演算はできないので「エラー」となります。＋ ’10’ は文字列の連結となり、変数 s に保存されている値と文字列’10’ を連結後の一つの文字列として評価されます。

それでは入力した数値を使って計算する方法を学んでみましょう。

キーボードから入力された数字は数値ではなく文字として認識されていることを確認しましょう。

2.4.4:　文字列の取り扱い（数字１）

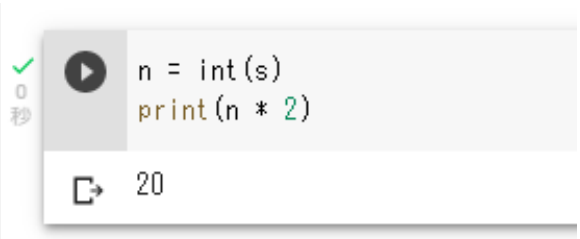


※　この例だと変数 s は文字列の’10’ が入っているため s * 2 は s を 2 回繰り返すという意味になります。

2.4.1.1　文字列から数値への変換

文字として入力された数字（文字列）を数値に変換するためには int 命令を利用します。

2.4.5:　文字列の取り扱い（数値への変換）



※　入力された文字列を数値に変換することで変数 n は数値として評価されます。

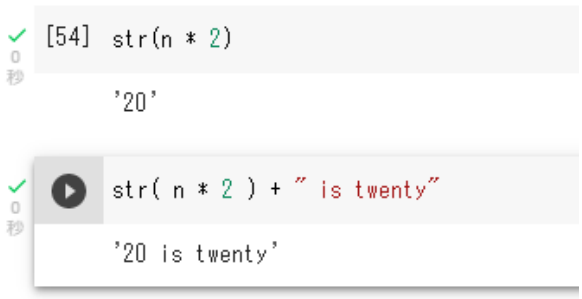
Point

数値として判定できない文字の扱いはエラー処理（例外処理）として別に考える必要があります。

2.4.1.2 数値から文字列への変換

数値を文字列にするには str 命令を利用します。

2.4.6: 数値から文字列のへの変換



str 関数によって文字列として取り扱われるようになりました。

文字列にすると「+」演算子で連結（文字列を結合すること）もできます。

凡例：input-1

学生 A さんの数学の得点を x に、英語の得点を y に入力し数値化して平均点を出力しましょう。（数値化後の変数をそれぞれ m と e を使います）

※　入力が数値として判定できない場合の処理は行いません。

コード　2.6: python4-1.py

```
1   x = input ( "math = " )
2   m = int ( x )
3   y = input ( "english = " )
4   e = int ( y )
5   print ( "The average score is " + str ((m + e) / 2) )
```

実行結果

```
1   math = 50
2   english = 80
3   The average is 65.0
```

※　数値以外のものを入力するとエラーになりますのでご注意ください。

2.4.1.3 練習問題

練習問題 ： Plactice 5

「凡例 input-1」を社会の得点を z に入力「”`social = `”」し、変数 s に数値化して学生 A さんの 1 科目当たりの平均点を出力しましょう。

Point

変数名について

変数名は変数が少ない時は一文字のアルファベット「a 〜 z」でも問題ないですが、長い手順になってくると意味を持たない文字列では分かりにくくなります。そこで少しずつ分かりやすい変数名をつけていくことを意識していきましょう。

2.5　繰り返し処理

ここからは繰り返し処理について学びます。コンピューターはどんなに同じことを繰り返しても文句ひとつ言いません。これがコンピューターを利用する大きな理由になっています。

2.5.1　for 文による繰り返し

プログラミングでは繰り返し処理をする方法が用意されています。

凡例：for-1

for 文を使って 3 回”wan” ”nyan”と表示します。

コード　2.7: python5-1.py

```
1   for x in range(3):
2       print( "wan" )
3       print( "nyan" )
```

実行結果

```
1   wan
2   nyan
3   wan
4   nyan
5   wan
6   nyan
```

※　range() の括弧中に書かれている回数繰り返していることがわかります。

for 文はブロック内に書かれている処理を range の中に書かれている回数だけ繰り返しすることが理解できたと思います。

2.5.1.1　練習問題

練習問題　：　Plactice 6

同じく 1000 回”wan” ”nyan”と表示させましょう。

2.5.1.2 range による値のリスト

for 文では in の後ろに書かれている range（）の括弧中に書かれている数値を in の前に書かれている変数に値を代入しながら値がなくなるまで繰り返す命令です。

では具体的に代入される値を確認してみましょう。

凡例：for-2

for 文の次に書いているのは変数 x です。

変数 x の値を表示してみましょう。

コード　2.8: python5-2.py

```
1   for x in range(3):
2       print( x )
```

実行結果

```
1   0
2   1
3   2
```

変数 x には range 命令で指定された範囲の値がひとつずつ入っていることが分かりました。range 命令ではどのような値が展開されるかは次の通りとなります。

range(3)・・・ 0,1,2

range(5)・・・ 0,1,2,3,4

※　「0」から括弧の中にある数値-1 までの範囲の値が作られます。

凡例：for-3

凡例 for-3

それぞれ次の range 命令で指定するとどんな値が入っているか確認してみましょう。

```
range(1, 5)
range(1, 10)
range(1, 10, 2)
range(2, 10, 3)
```

コードセルで実行してみましょう。

コード　2.9: コードセルによる実行

```
print("range(1, 5)")
for x in range(1, 5):
    print(x)
print("range(1, 10, 2")
for x in range(1, 10, 2):
    print(x)
print("range(2, 10, 3)")
for x in range(2, 10, 3):
    print(x)
```

```
range(1, 5)
1
2
3
4
range(1, 10, 2
1
3
5
7
9
range(2, 10, 3)
2
5
8
```

このように range() で作られる値は最初の値が初期値、２つ目の値が上限値（未満となるよう）、３つ目の値を加算していくため、次のようになることがわかります。

range(1, 5) → 1, 2, 3, 4
range(1, 10) → 1, 2, 3, 4, 5, 6, 7, 8, 9
range(1, 10, 2) → 1, 3, 5, 7, 9
range(2, 10, 3) → 2, 5, 8

括弧の中で指定している最初の値から次の値-1 まで最後の値が加算された値が取得されます。

凡例：for-4

1 から 10 までの数値を足した合計を求めるプログラムを作りましょう。

コード　2.10: python5-3.py

```
1   sum = 0
2   for x in range(1,11):
3       sum = sum + x
4   print ( sum )
```

2.5.1.3　練習問題

練習問題　：　Plactice 7

1．「凡例 for-4」を改良して 1 から 1000 までの数値を足した合計を求めるプログラムを作りましょう。

2．「凡例 for-4」に追加して「アキレスと亀」（ゼノンのパラドックス）をやってみましょう。（詳しくはネットで検索しましょう）

※　1/2 + 1/4 + 1/8 + 1/16 無限に計算すると値が収束することを確かめます。
人間が計算するとすぐ嫌になりますが、コンピューターは音を上げません。
ただし range で計算する数値の範囲が、2 ** 128 あたりでどのくらいになるのか想像してから実行してみてください。

※　（1 / 2 ** 53) あたりまでで精度が足りなくなるみたいです。

2.5.2　繰り返し処理の利用

　繰り返しの中にキーボードからの入力を処理する機能をいれて、プログラムを短くしてみましょう。

先ほどの「凡例 input-1」を 10 科目の平均表示に変更することを考えてみます。

※　悪い例「人がやる繰り返し」です。変数名が増えますので、ちょっと手抜きします。

2.5.1:　人がやる繰り返し

```
 1   x = input ( "subject1 = " )
 2   y = int ( x )
 3   x = input ( "subject2 = " )
 4   y = y + int ( x )
 5   x = input ( "subject3 = " )
 6   y = y + int ( x )
 7   x = input ( "subject4 = " )
 8   y = y + int ( x )
 9   ※　そろそろ飽きてきます。
10           •
11           •
12           •
13   x = input ( "subject10 = " )
14   y = y + int ( x )
15   print ( y / 10 )
16   ※　できますが、電卓入力のほうが楽です。
```

　この問題を解決するために、効率よく繰り返しコンピューターに処理を行わせる方法を学びましょう。

　よく見ると 2 科目目以降はそっくりです。

```
x = input ( "subject2 = " )
y = y + int ( x )
```

　1 科目目も 0 に入力した値を加算すると書き換えてみましょう。（初期値 0 という意味です。）

```
x = input ( "subject1 = " )
y = 0 + int ( x )
```

つまり最初だけ変数 y に 0 を入れます。

```
y = 0
ここから 10 回繰り返し
x = input ( "subjectN = " )
y = y +   int ( x )

繰り返しが終わったら平均値を出力します。
print ( y / 10 )
```

凡例：for-5

繰り返しを用いてシンプルに記述します。

python5-4.py

```
1   y = 0
2   for n in range(1, 11):
3       x = input ( "subject" + str(n) + " = " )
4       y = y + int( x )
5   print( y / 10 )
```

※　こんなに短くできました。

プログラムを書くときは短く効率的に書くことが大切なことがわかります。

繰り返しはコンピュータにさせるべきで人間が繰り返し同じことを書かない方が良いといわれています。

2.5.2.1　練習問題

練習問題　：　Plactice 8

「凡例 for-5」のプログラムの科目毎の点数入力の際「subject1 5 =」のように何番目の科目を入力しているのか分かりやすく表示してください。

2.5.3　繰り返し処理のその他の制御

繰り返しには他にも while 文があります。

while 文の書き方は次の通りですが、条件式（式の評価）が真の間繰り返すことになります。

```
while 条件式:
    手続き１
    手続き２
```

・

基本的には for 文と同様に while 文以降のブロックを条件が真の間繰り返しますので、ブロック内で条件が偽になるような処理を記述する必要があります。

凡例：loop-1

while を使って１～５までの合計を求め出力するプログラムを作ります。

python5-5.py

```
1   i = 1
2   total = 0
3   while i <= 5:
4       total = total + i
5       i = i + 1
6
7   print(total)
```

繰り返し処理の途中で繰り返しを終了する制御文に break があります。
break 文を利用すると for 文や while 文といった繰り返しを途中で終了することができます。

```
while 条件式:
    手続き１
    手続き２
        break   #これで繰り返し文を抜けることができます。
```

break 制御を使うことで繰り返し処理を終了できますので、while 文の条件式に True を設定しても制御可能になります。

凡例：loop-2

while を使って１～５までの合計を求め出力するプログラムを作ります。ここで while 文は式の評価結果を常に真とするようにしています。

python5-6.py

```
1  i = 1
2  total = 0
3  while True:
4      total = total + i
5      i = i + 1
6      if i == 6:
7          break
8
9  print(total)
```

python の繰り返し処理には繰り返しが終了したタイミングで終了時の処理をするために else:ブロックを利用することができます。

```
while 条件式:
    手続き１
    手続き２
else:
    条件式が偽になった時の手続き
```

ただし、この処理は条件式が偽になる必要があるため、break 文による制御で繰り返し処理を終了した場合は、else:ブロックは通過しません。

また、他には continue 制御文もありますが、ここでは説明しませんので、興味があれば検索してください。

2.6 配列について（繰り返し処理の活用）

2.6.1 配列 (list) とは？

効率的に処理するために繰り返し処理と組み合わせてよく使われるのが配列です。配列の考え方は一覧（並び）に近いので一覧を表示することをイメージしてみます。

得点入力の科目名を一覧表示するプログラムは次の通りです。

何度も同じ科目名を入力したり出力するのであれば変数に入れておきますが、繰り返し同じことを書くのは非効率的です。

2.6.1: 変数の利用 (繰り返し記述)

```
1   m = "Mathematics”
2   print ( m )
3   e = "English”
4   print ( e )
5   so = "Social”
6   print ( so )
7   j = "Japanese”
8   print ( j )
9   sc = "Scientific”
10  print ( sc )
```

そこで繰り返し利用できるように、同じ変数に値の一覧として格納していきます。

2.6.2: 配列の利用

```
subjects = ["Mathematics”,”English”,”Social”,”Japanese”,”Scientific”]
```

この変数 subjects に格納されている値のイメージは次の通りとなります。

	変数 subjects（配列）
0 番目	”Mathematics”
1 番目	”English”
2 番目	”Social”
3 番目	”Japanese”
4 番目	”Scientific”

ここで変数 subjects に格納されている値を参照するためには subjects[0] と [] 角括弧の中に何番目を表す数値や数値の入った変数を利用することができます。

つまり変数 subjects の中の”English”を参照するためには 1 番目を参照すればよいので `subjects[1]` と記述します。

Point
配列を番号で参照するときの番号を「添字」もしくは「インデックス」と呼びますので覚えておきましょう。

for 文と配列を使って効率よく出力処理してみましょう。

凡例：list-1

配列 list を使って 5 科目を一覧表示してください。

コード　2.11: python6-1.py

```
1  subjects = ["Mathematics","English","Social","Japanese",
2  "Scientific"]
3  for s in subjects:
4      print ( s )
```

実行結果

```
1  Mathematics
2  English
3  Social
4  Japanese
5  Scientific
```

for 文では in の後ろにある配列（list）の内容を最後まで繰り返し処理することができます。

for 文では配列の最初の値を変数に代入してブロック内の処理します。次に配列の次の値を変数に代入してブロック内の処理をします。配列の中身をすべて変数に代入し、次の処理すべき配列がなくなったときに処理を終了します。

つまり変数 s の中身は”Mathematics” →” English” →” Social” →” Japanese” →” Scientific” と変化しながらブロック内の処理「`print ( s )`」を実行することになります。

for 文と配列を使ってデータ入力と出力を効率よく処理する流れを見てみましょう。

凡例：list-2

配列（リスト）を使って 5 科目を一覧表示し、科目の点数を配列 scores に入力して、一覧表示してください。

ファイル python6-1.py を python6-2.py にコピーにして保存してください。

コード　2.12: python6-2.py

```
1   subjects = ["Mathematics","English","Social",
2       "Japanese","Scientific"]
3   scores = []
4   for s in subjects:
5       x = input( s + " = " )
6       y = int ( x )
7       scores.append( y )
8
9   l = len(subjects)
10  #配列の長さは len 関数を利用することで求められます（ここでは 5 となります）
11  for n in range(l):
12      print ( subjects[n] + ":" , scores[n])
13  # print 命令は「,」（カンマ）でつなぐと改行なしで値を出力できます。
```

※　プログラム中のコメント（プログラムの説明文）は「#」以降となり実行に影響を与えません。

配列リストは自由に要素を追加削除できます。

`scores = []`

この文では変数 scores が配列であることを宣言しています。

`scores.append( y )`

この文は配列に値を追加しています。（任意の値を追加していくことができます）

Point

また、何度か追加したリストの長さは `len` 関数の括弧の中に記述することで長さが取得できます。

この配列（リスト）はイメージとして次のようになります。

表 2.3　配列のイメージ

	subjects	scores
0 番目	”Mathematics”	10
1 番目	”English”	20
2 番目	”Social”	30
3 番目	”Japanese”	40
4 番目	”Scientific”	50

値を参照するためには配列名 [番号] で利用することができますので、科目名と点数の一覧は次の命令で出力させることができます。

2.6.3:　繰り返し出力

```
1   for n in range(5):
2       print ( subjects[n] + ":”  , scores[n])
```

この時の変数 n の値は 0,1,2,3,4 とれぞれ繰り返しの中で変化していきますので、出力結果は次の通りとなります。

```
1   Mathematics: 10
2   English: 20
3   Social: 30
4   Japanese: 40
5   Scientific: 50
```

for 文と配列、if 文を組みあわせて行う処理を学んでみましょう。

凡例：list-3

「凡例 list-2」の繰り返しの中で得点が 60 点以上は”合格”、60 点未満は”不合格”と表示するように変更してみましょう。
ファイル python6-2.py を python6-3.py にコピーにして保存してください。

コード　2.13: python6-3.py

```
1  subjects = ["Mathematics","English","Social",
2      "Japanese","Scientific"]
3  scores = []
4  for s in subjects:
5      x = input( s + " = " )
6      y = int ( x )
7      scores.append( y )
8
9  for n in range(0,5):
10     if scores[n] >= 60:
11         mes = "合格"
12     else:
13         mes = "不合格"
14     print ( subjects[n] + ":" , scores[n], mes)
```

2.6.1.1 練習問題

練習問題 : Plactice 9

ここまでの学習内容を復習して 5 科目の点数を入力し、0 点未満や 100 点を超える点数は「error」と表示し入力された場合は得点に-1 を設定するようにしましょう。
5 科目の点数を入力完了したら、このテストでは 80 点以上を「excelent」とし、60 点以上を「good」とし、40 点以上を「passing」、40 点未満を「failing」と評価します。5 科目の得点の評価を判定して出力するプログラムを作りましょう。
エラーと表示された科目については「error」と出力します。

実行例

```
c:\\〜> python plactice9.py
Mathematics = 50
English = 60
Social = 80
Japanese = 39
Scientific = 101
error

Mathematics: 50 passing
English: 60 good
Social: 80   excelent
Japanese: 39 failing
Scientific: -1 error
```

2.6.2　２次元配列

3 × 2 の配列（リスト）を作ってみましょう。学生 3 人の数学と英語の点数を管理する方法を考えます。

A さんの数学（60）と英語（80）
B さんの数学（70）と英語（90）
C さんの数学（80）と英語（100）

```
math = [60, 70, 80]
english = [80, 90 ,100]
a = [60, 80]
b = [70, 90]
c = [80, 100]
```

どちらの考え方も間違っていませんが、科目が増えたり人数が増えたときに効率よく扱うことができません。

2.6.2.1　２次元配列の宣言

一人分のデータをリストとしてさらにリストの中に入れることができます。

```
students_scores = [[60, 80], [70, 90], [80, 100]]
```

表 2.4　二次元配列

		0 列目	1 列目
		数学	英語
0 行目	A さん	60	80
1 行目	B さん	70	90
2 行目	C さん	80	100

A さんの数学の点数（60 点）は配列 `students_scores[0][0]` で参照できます。同様に B さんの数学の点数（70 点）を参照するには `students_scores[1][0]`、C さんの英語の点数（100 点）を参照するには `students_scores[2][1]` と表現します。

2.6.2.2　データの追加（行）

D さんのデータ（数学　50 点,　英語　70 点）が増えたときは `students_scores` に D さんの配列 `[50, 70]` を次のように追加します。

```
students_scores.append([50, 70])
```

配列の中身：`[[60, 80], [70, 90], [80, 100], [50, 70]]`

2.6.2.3 データの追加（列）

A さんの 3 科目目（社会科　70 点）を追加するときは A さんのデータ `students_scores[0]` に `append` します。

```
students_scores[0].append(70)
```

配列の中身：`[[60, 80, 70], [70, 90], [80, 100], [50, 70]]`

表 2.5　二次元配列操作後

		0 列目	1 列目	2 列目
		数学	英語	社会
0 行目	A さん	60	80	70
1 行目	B さん	70	90	
2 行目	C さん	80	100	
3 行目	D さん	50	70	

2.6.2.4 2 次元配列を利用した集計

凡例：list-4

2 次元配列の例で利用した students_scores = [[60, 80], [70, 90], [80, 100]](数学と英語の点数) を使って科目の合計点を求めるプログラムを作成しましょう。

コード 2.14: python6-4.py

```
1  students_scores = [[60, 80], [70, 90], [80, 100]]
2  m = 0 #数学の合計点
3  e = 0 #英語の合計点
4
5  for v in students_scores:
6      m = m + v[0]
7      e = e + v[1]
8
9  print ("数学の合計点数", m)
10 print ("英語の合計点数", e)
```

実行結果

```
1  数学の合計点数 210
2  英語の合計点数 270
```

この凡例の for 文では in の後ろにつく配列 students_scores は 3 人分の数学と英語の点数が入っているので一人分のデータを v に代入することになります。この一人分のデータは数学と英語の得点の配列となりますので、v[0]（数学の点数）v[1]（英語の点数）として参照することができます。

また二次元配列を参照しながら処理を行うには添字を使う方法もあります。

凡例：list-5

添え字を使って前の凡例と同「list-4」じ機能を作ってみましょう。

コード　2.15: python6-5.py

```
1   students_scores = [[60, 80], [70, 90], [80, 100]]
2   m = 0 #数学の合計点
3   e = 0 #英語の合計点
4
5   for i in range (3):
6       m = m + students_scores[i][0]
7       e = e + students_scores[i][1]
8
9   print ("数学の合計点数", m)
10  print ("英語の合計点数", e)
```

2 次元配列の表現は添字の意味が分かっていないと複雑に感じてしまいますが、配列のイメージができていればどこの値を参照しているのかは理解できると思います。

最近のプログラムでは配列に入っている要素数が未定でもプログラムの変更が少ない「list-5」のプログラムが利用されることが多いようです。

2.6.2.5 練習問題

練習問題 : Plactice 10

次のプログラムを参考にして 2 次元配列に表のような点数を 5 名分追加して数学と英語の平均点を求め出力するプログラムを作成しましょう。

	数学	英語
A さん	60	80
B さん	70	90
C さん	80	100
D さん	50	70
E さん	60	60

配列宣言とデータの追加

```
1  students_scores = []
2  #配列の準備
3  students_scores.append([60, 80]) #A さんのデータを追加
4  students_scores.append([70, 90]) #B さんのデータを追加
5          ・
```

実行結果

```
1  数学の平均点 64.0
2  英語の平均点 80.0
```

2.6.2.6 練習問題

練習問題 : Practice 11

「Practice9」のプログラムを 3 人分の得点を入力して 5 科目それぞれの平均値を出力するように変更してください。この時「error」な点数の入力はないものとします。

実行例

```
c:\\〜> python python11.py
1 人目の入力
Mathematics = 20
English = 30
Social = 40
Japanese = 50
Scientific = 60
2 人目の入力
Mathematics = 80
English = 90
Social = 100
Japanese = 20
Scientific = 50
3 人目の入力
Mathematics = 70
English = 80
Social = 90
Japanese = 100
Scientific = 70
Mathematics の平均点 56.666666666666664
English の平均点 66.66666666666667
Social の平均点 76.66666666666667
Japanese の平均点 56.666666666666664
Scientific の平均点 60.0
```

2.6.3　配列（リスト）操作について

配列で操作可能な処理として「.append()」を使いましたが、ほかにどのような操作が可能か見てみましょう。

コードセルを使って配列に対する操作をしてみましょう。

2.6.4:　配列リストの操作

```
l = [ 4 , 5 , 7 , 2 , 1 , 9 ]
print(l)
l.sort() #昇順ソート
print(l)
l.sort( reverse = True ) #降順ソート
print(l)
print(l.pop()) #最後の要素の取り出し
print(l.pop(0)) #先頭の要素の取り出し
```

```
[4, 5, 7, 2, 1, 9]
[1, 2, 4, 5, 7, 9]
[9, 7, 5, 4, 2, 1]
1
9
```

他にもリストの操作をする命令はありますが、主なものだけ紹介させていただきました。

2.7　辞書について

2.7.1　辞書（dictionary）とは？

キー値とキー値によって取得される値の組み合わせで利用されます。

キー値とは配列の添え字として利用可能な「名前」をさします。配列では 0 番目、1 番目・・・と数えることになりますが、配列の添え字として文字列を利用することで特定の値をすぐに取得できるようになります。

辞書は次のように定義します。

2.7.1:　関数の定義

```
1　辞書名 = {’キー値１’:’値１’, ’キー値２’:’値２’, "キー値３":"値３"}
2　※キー値と値をコロンで区切る
```

辞書データ dictdata に日本の空港のレターコードをキーとして空港名を設定する

凡例：dict-1

python7-1.py

```
1　dictdata = {’CTS’:’SHINCHITOSE’, ’HKD’:’HAKODATE’,
2　　’HND’:’TOKYOKOKUSAI’, ’NRT’:’NARITAKOKUSAI’ }
3　print(dictdata[’CTS’])
4
```

実行結果

```
1　SHINCHITOSE
```

キー値	値
’CTS’	’SHINCHITOSE’
’HKD’	’HAKODATE’
’HND’	’TOKYOKOKUSAI’
’NRT’	’NARITAKOKUSAI’

このような参照ができるようになります。

WebAPI では JSON を利用してデータの交換をすることが多いですが、JSON データをリスト（配列）もしくは辞書（連想配列）として取り扱うことができます。

2.7.2 JSON と python の関係

2.7.2.1 JSON 入門

JavaScript Object Notation の略であり、連想配列とリスト構造のテキストとしての表記法です。

ネットワーク上のデータを API で取得する際に利用されることが多く、python 上での利用も目立ってきています。

連想配列とはキー・バリュー形式の配列を意味していて配列の添え字の代わりにキーの値を使って値を取り出すことができる仕組みとなります。

JSON では「{}」で囲まれた中に「,」カンマで複数の要素を入れることができます。また「キー値」と格納する「値」を「：」コロンで区切ります。

凡例：json-1

JSON では「{}」で囲まれた中に「,」カンマで複数の要素を入れることができます。また「キー値」と格納する「値」を「：」コロンで区切ります。
このまま連想配列やリストと同様の記述となりますので、python の dict オブジェクトとして利用することができます。

python7-2.py

AKJ ⇒旭川空港
MMB ⇒女満別空港
CTS ⇒新千歳空港
HKD ⇒函館空港
HND ⇒東京国際空港

これらを JSON で記述すると次のようになります。
基本的に python の連想配列と同じ記述となります。

```
1  {
2   "AKJ":"旭川空港" ,
3   "MMB":"女満別空港" ,
4   "CTS":"新千歳空港" ,
5   "HKD":"函館空港" ,
6   "HND":"東京国際空港"
7  }
```

また JSON では配列（リスト構造）も「[]」角カッコで記述することができます。
[60, 70 ,50 , 80, 100]

凡例：json-2

2 年 3 組　佐々木君の国社数理英の JSON 表記は次のようになります。

python7-3.py

```
1   json_data = {
2     "class":"2 年 3 組" ,
3     "name":"佐々木" ,
4     "score":[60, 70 ,50 , 80, 100]
5     }
6
7   print('json_data["name"] ⇒ ' +json_data['name'])
8   print('json_data["score"] ⇒ ' + str(json_data['score']))
9   print('json_data["score"][1] ⇒ ' + str(json_data['score'][1]))
10
11  total = 0
12  for score in json_data['score']:
13          total = total + score
14  print("平均点:" + str(total/5))
```

実行結果は次のようになります。

```
1   json_data["name"] ⇒ 佐々木
2   json_data["score"] ⇒ [60, 70 ,50 , 80, 100]
3   json_data["score"][1] ⇒ 70
4   平均点:72
```

この時の json_data は dict 型（辞書型）となり、json_data[”score”] は list 型となります。

2.8 関数による処理（まとまった手順）

2.8.1 関数とは？

ここまで見てきた手順はずらずらと列挙するだけでしたが、まとまった手続きは一つの処理手順としてグループ化するようにまとめて記載することができます。これを関数といいます。

関数は（）の中に記載された値を引き渡す（引数・パラメーターといいます）ことができます。この引数は「、」カンマで区切ることで複数利用することができます。

関数は次のように定義を記述します。

2.8.1: 関数の定義

```
1  def 関数名(引数1,引数2):
2      ・
3      まとまった処理
4      ・
5      return 戻り値
```

関数の名前は変数と同じように命名できます。数学の関数定義と同じように y = f(x) と関数「f(x)」の結果（関数を評価した値）を取得するために return 文を使います。

Point
関数は関数の下に呼び出すプログラムを書くことで実行することができます。関数を呼び出すプログラムを先に書くとエラーになります。

Point
関数は引数を全く持たない関数も作ることができます。いままで利用してきた input() も関数です。

関数を利用するためには関数の定義を先にする必要があります。凡例で関数の定義と関数を使ったプログラムを動作させてみましょう。

凡例：func-1

$y = x^2 + 2x + 3$ の 2 次方程式の値を求める関数 func を定義して x が 2 と 4 の時の y の値を求めるプログラムを作ります。

python8-1.py

```
1   def func(x):
2       answer = x ** 2 + 2 * x + 3
3       return answer
4   #ここでインデント（字下げ）を戻すことで関数（まとまった手順）が終わった
    という意味になります。
5   y = func(2) #ここで上に書いた関数 func を呼び出しています。
6   print( y )
7   y = func(4)
8   print( y )
```

実行結果

```
1   11
2   27
```

それぞれ「11」と「27」が関数の結果として表示されることを確認できます。

コードセルで定義の無い関数を呼び出すと次のエラーが出力されます。プログラムとして保存して実行しても同様のエラーとなります。

2.8.1.1 関数の呼び出し階層

関数は呼び出しの階層が 1 つだけではなく何階層も深く呼び出すことができます。つまり関数から関数を呼び出すように書くことができます。

凡例：func-2

$y = x^2 + 2x + 3$ の 2 次方程式の値を求める関数 func1 と x の値を 0 から n - 1 と変化させたときの y の値の合計を求める関数 func2 のプログラムを作ります。
作ったプログラムで上記 2 次方程式の 0 から 4 まで整数代入した合計を求めます。

python8-2.py

```
1   def func1(x):
2       answer = x ** 2 + 2 * x + 3
3       return answer
4
5   def func2(n):
6           t = 0
7           for i in range(n):
8                   y = func1(i)
9                   t = t + y
10          return t
11
12  a = func2(5)
13  print( a )
```

実行結果

```
1   65
```

関数では深い階層の呼び出しも可能です。いろいろと試してみてわかりやすいプログラムを組む練習をしてみましょう。

2.8.1.2 練習問題

練習問題 : Plactice 12

次の 5 名のテストの結果（平均点 65 点）を入力して、分散を求めるプログラムを作りましょう。この時の点数は 50, 80, 85, 70, 40 で平均 65 とします。

$$\text{分散} = \frac{1}{\text{人数}} \sum (\text{得点} - 65(\text{平均点}))^2$$

関数 func1 は平均点との差の 2 乗を求め戻り値とします。
関数 func2 はテストの点数を人数分繰り返し入力し、合計を求めます。
呼び出し側は関数 func2 を呼び出し得られた結果を人数 5 で割って分散を求め表示します。
実行結果

```
1    300.0
```

結果を確認してみてください。

2.8.2　変数のスコープ（有効範囲）について

いままで変数として利用してきた変数は値が代入されてから有効になっていましたが、値が代入され利用される範囲（スコープ・有効範囲）にはルールがあります。

プログラム中に特に明確に範囲を指定せず値が代入された変数は代入後どの位置でも利用できるグローバル変数と言われ、特に制約なく利用できます。制約なく利用できるグローバル変数は一見便利ですが、いつどこで誰が値を変更するかわかりませんので必要以上に多用することはお勧めしません。

また関数内で利用されている変数は関数内でしか利用することができない変数で値の利用される範囲が関数内に限定されたローカル変数と言われます。**コードセル**で確認してみましょう。

2.8.2:　変数のスコープ

```
0
秒
```

```
##関数の定義
def func(x):
    answer = x ** 2 + 2 * x + 3
    return answer

answer
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-61-c0ea2c0929a4> in <module>()
      4     return answer
      5
----> 6 answer

NameError: name 'answer' is not defined
```

SEARCH STACK OVERFLOW

2.8.2.1 グローバル変数

グローバル変数はどこでも利用できますので、関数内であっても関数外であっても変数を共有できます。グローバル変数の有効範囲を確認してみましょう。

凡例：scope-1

凡例 func-1 の変数 x をグローバルで利用し、$x^2 + 2x + 3$ の値を x にそれぞれ 2 と 4 を設定し、計算するプログラムを作ります。

python8-3.py

```
1   #引数にはなにも指定しません。
2   def func():
3       answer = x ** 2 + 2 * x + 3
4       return answer
5
6   x = 2
7   #ここで x に代入するとグローバル変数として扱われます。
8   y = func()
9   print( y )
10  x = 4
11  y = func()
12  print( y )
```

実行結果

```
1   11
2   27
```

グローバル変数は関数の外で代入された変数で、その変数は関数内外に関わらず同じ変数名で参照できることが確認できました。

2.8.2.2 ローカル変数

変数の有効範囲を確認するために、グローバル変数とローカル変数の動きについて確認しましょう。

凡例：scope-2

グローバル変数 x に 2 と 4 を代入して $y = x^2 + 2x + 3$ となる $y = f(x)$ の $f(x)$ を呼び出し結果を出力するプログラムを作成します。

python8-4.py

```
1   #スコープが異なれば同じ変数名が利用できます。
2   def func(x):
3       answer = x ** 2 + 2 * x + 3
4       x = -1
5       #func 内で利用可能な x は他の部分で利用される x とは異なる変数となります。
6       return answer
7
8   x = 2
9   #ここで x に代入するとグローバル変数として扱われます。
10  y = func( x )
11  print( y )
12  print( x )
13
14  x = 4
15  y = func( x )
16  print( y )
```

実行結果

```
1   11
2   2   #  グローバル変数 x の値が出力された
3   27
```

この凡例では関数 func の宣言（2 行目）の中で使われている変数 x と answer はローカル変数です。ローカル変数は関数内のみ有効で関数の命令を実行し、関数を終了すると利用できなくなります。

プログラムは 8 行目でグローバル変数 x に 2 が代入されます。10 行目で func 関数を呼び出し 2 行目に実行が移りますが、2 行目の関数の引数 x はローカル変数となり、このローカル変数に 2 が代入されます。4 行目で x に-1 を代入していますが、これはローカル変数となります。6 行目で結果を戻り値として関数を終了後 10 行目に実行が移ります。この後 12 行目で x の値を出力したときには関数 func を終了していますので、ローカル変数 x は利用できなくなっていて、グローバル変数 x の値が出力されることになります。

Point

グローバル変数とローカル変数が同じ名前の変数であってもそれぞれ別のモノとして扱われます。

2.8.2.3 関数のメリット

関数を利用すると一つの機能をまとめて記述することができます。こうすることで一つ一つの機能や処理を短くすることができて複雑なプログラムもシンプルな機能を呼び出すことでわかりやすいことがわかります。

凡例：func-3

テストでは 80 点以上を「excelent」とし、60 点以上を「good」とし、40 点以上を「passing」、40 点未満を「failing」と評価する関数を記載します。100 点を超えていたり 0 点未満の場合「error」という戻り値となり、点数を引数とする関数 evaluation を作成しましょう。

※　Plactice9 を関数 evaluation を利用して記載することができます。

python8-5.py

```
1   def evaluation(score):
2       if score < 0 or score > 100:
3               mes = "error"
4       elif score >= 80:
5               mes = "excelent"
6       elif score >= 60:
7               mes = "good"
8       elif score >= 40:
9               mes = "passing"
10      else:
11              mes = "failing"
12      return mes
13
14  subjects = ["Mathematics" ,"English","Social","Japanese", \
15                          "Scientific"]
16  scores = []
17  for s in subjects:
18      x = input( s + " = " )
19      y = int ( x )
20      scores.append( y )
21  for n in range(5):
22      mes = evaluation(scores[n])
23      print ( subjects[n] + ":" , scores[n], mes )
```

2.8.2.4 練習問題

練習問題 ： Plactice 13

次の実行例のプログラム、3 名の学生の名前と 5 科目の得点を入力してそれぞれの一人ずつ平均値を出力するように変更してください。この時「error」な点数の入力があれば-1 点として平均値に算入しないものとします。
ただし、名前にエラー入力は無いものとします。

	scores	0 列目	1 列目	2 列目	3 列目	4 列目	5 列目	6 列目
		名前	数学	英語	社会	国語	科学	受験科目数
0 行目	一人目							
1 行目	二人目							
2 行目	三人目							

プログラム（未完成）

```
1  #グローバル変数（配列）の準備
2  subjects = ["Mathematics" ,"English","Social","Japanese", \
3                          "Scientific"]
4  scores = []
5  #関数 input_data の記述
6  def input_data():
7      for n in range(3):
8          cnt = 0
9          scores.append([])
10         name = input("Name = ")
11         scores[n].append(name)
12         for s in subjects:
13             x = input( s + " = " )
14
15
16
17
18
19
20
21         scores[n].append(cnt)
22 #関数 get_average の記述
23 def get_average(m):
24     sum = 0
25     for n in range(1,6):
26
27
28     return sum / scores[m][6]
29
30 #プログラムはここから実行します。
31 input_data()
32 for m in range(3):
33     ave = get_average(m)
34     print(scores[m][0],"の",scores[m][6],"科目の平均点",ave)
```

実行例

```
c:\\〜> python plactice13.py
Name = Sasaki
Mathematics = 20
English = 30
Social = 40
Japanese = 50
Scientific = 120
error score
Name = Ogawa
Mathematics = 80
English = 90
Social = 100
Japanese = 20
Scientific = 50
Name = Naoi
Mathematics = 70
English = 80
Social = 90
Japanese = 100
Scientific = 70
Sasaki の 4 科目の平均点 35.0
Ogawa の 5 科目の平均点 68.0
Naoi の 5 科目の平均点 82.0
```

2.9　機能の追加（ライブラリ（モジュール）の利用）

2.9.1　ライブラリの利用

python では時間や乱数機能の追加を行うことができます。

システムの時間を取得する datetime や time と乱数を扱う random ライブラリをここで紹介します。

他にもたくさんのライブラリが利用できますので、興味のある方は調べてみましょう。

凡例：module-1

for 文で 10 回繰り返しを行い、ライブラリを使って現在時刻、カウンター i の平方根 $\sqrt{\quad i \quad}$ や 3 乗根 $\sqrt[3]{\quad i \quad}$ を表示して、乱数で生成された 1〜3 の数値の秒数を待ち時間とするプログラムを作成しましょう。

python9-1.py

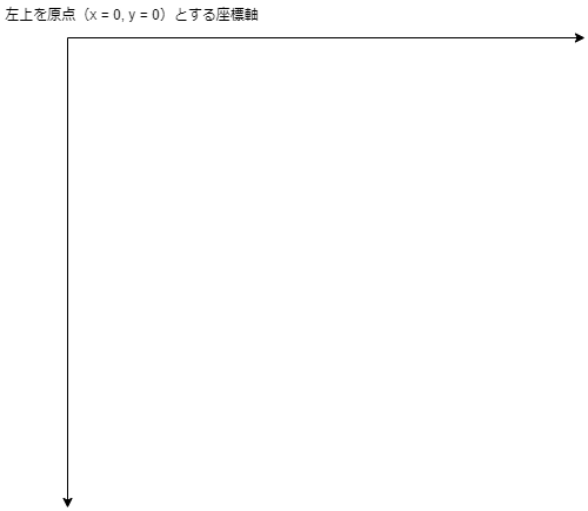
```
1   #ライブラリを利用する場合 import 文で読み込みを行う
2   import random
3   import time
4   import math
5   for i in range(10):
6       print( i )
7       #現在時刻を取得し表示する
8       print( time.ctime() )
9       #カウンターの平方根を求め表示する
10      print( math.sqrt(i) )
11      #カウンターの平方根を求め表示する
12      print( math.pow(i , 1 / 3))
13      #1~3 の乱数を生成する
14      a = random.randint(1,3)
15      print( a )
16      #sleep は引数で指定された時間
17      time.sleep( 1 )
18  else:
19      print( "end" )
```

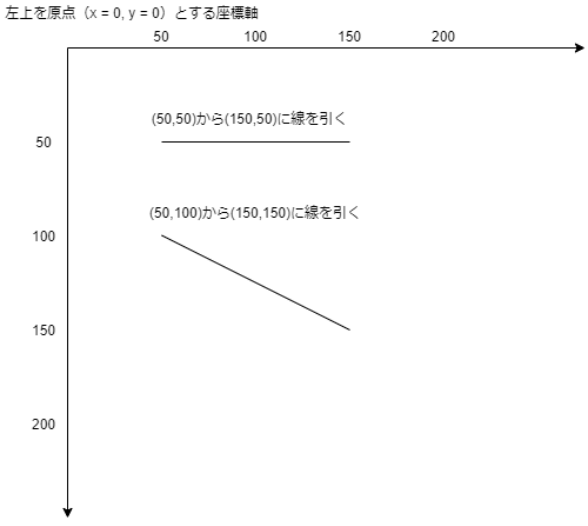
2.9.2　図形の描画ライブラリ

それではライブラリを活用してイメージを描画してみましょう。
イメージを扱うためのライブラリは「Pillow」や「OpenCV」が有名です。

- Pillow は基本的な図形描画やリサイズ・トリミング・モザイク加工等ができます。
- OpenCV は Pillow の機能に顔認識や画像変換・エッジ等の機能が追加されている高機能ライブラリです。

このテキストでは簡単でわかりやすい描画ライブラリである Pillow を使います。
コンピューターのグラフィックの座標軸は次の図形にある通り左上を原点 $(0, 0)$ として x 座標,y 座標共に増加すると右方向、下方向に座標を移動します。





Point

Pillow ライブラリを利用するには次のコマンドで python の Pillow ライブラリをインストールする必要があります。

```
pip install Pillow
```

凡例：pillow-1

それでは Pillow ライブラリを使って先の図形のように 2 点間の線を引いてみましょう。
最初に描画用の領域（サイズが 500px,300px）を作成して表示してみましょう。

python9-2.py

```
1  #Pillow ライブラリを利用する場合は名称 PIL を使う宣言が必要で、
2  #import 文で描画に必要なライブラリを指定する。
3  from IPython.display import display #Colaboratory で表示するには
   display を利用します。
4  from PIL import Image
5
6  im = Image.new('RGB', (500, 300)) #イメージを作成する。'RGB' は赤・
   緑・青の色指定を意味する
7
8  #im.show() #PC 上でイメージを表示する
9  display(im)
```

描画用のエリアに「$(50, 50) - (150, 50)$」と「$(50, 100) - (150, 150)$」の座標で白い線を引いてみましょう。

python9-3.py

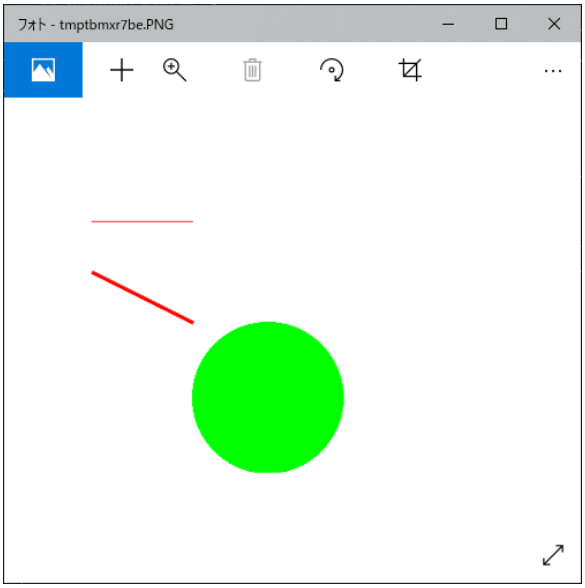
```
1  from PIL import Image,ImageDraw #ImageDraw のインポートを追加する
2
3  im = Image.new('RGB', (500, 300))
4  draw = ImageDraw.Draw(im) #描画用のオブジェクトを生成する
5
6  draw.line((50,50,150,50)) #x,y 座標を指定して線を引く
7  draw.line((50,100,150,150))
8
9  im.show()
```

それでは赤い線に変更してみましょう。

python9-4.py

```
1   from PIL import Image,ImageDraw
2
3   im = Image.new('RGB', (500, 300))
4   draw = ImageDraw.Draw(im)
5   draw.line((50,50,150,50),fill=(255,0,0)) #引数に色（赤）指定を追加
    する
6   draw.line((50,100,150,150),fill=(255,0,0), width = 4)
7     #引数に線の幅を指定する
8   draw.ellipse((150,150,300,300),fill=(0,255,0))
9     #円を緑色で描画する。
10  im.show()
```

プログラムの実行結果



ここで ImageDraw オブジェクトを利用した図形の描画関数の一部を紹介します。凡例「pillow-1」で利用している ImageDraw オブジェクトの line 以外のいくつかとなりますが、他詳しい内容は「pillow ImageDraw」で検索をしてください。

2.9.1: 図形描画関数（メソッド）の利用

ImageDraw オブジェクトで呼び出す描画関数の一部
point,line,poligon では複数の座標をしていして多地点を結ぶように座標指定することができます。

点を描画

point(座標 , fill=)

座標: (座標 x1, 座標 y1, 座標 x2, 座標 y3........)

fill: color の指定

線を描画

line(座標 , fill= , outline= , width= ,joint=)

座標: (座標 x1, 座標 y1, 座標 x2, 座標 y3........)

fill: color の指定

width: 線の幅

joint: "curve"を指定すると折り曲げた際の線の角が丸くなる

矩形（長方形）を描画

rectangle(座標 , fill= , outline= , width=)

座標:「(始点座標 x, 始点座標 y, 終点座標 x, 終点座標 y)」

fill: color の指定

outline: color の指定

width: 線の幅

楕円を描画

指定した矩形に内接する楕円を描画する

ellipse(座標 , fill= , outline= , width=)

座標:「(始点座標 x, 始点座標 y, 終点座標 x, 終点座標 y)」

※ここの座標は円図形を囲うのに必要な箱の始点と終点となります。

fill: color の指定

outline: color の指定

width: 線の幅

円弧を描画

始点終点を指定して円弧を描画

arc(座標 , start, end, fill= , width=)

座標:「(始点座標 x, 始点座標 y, 終点座標 x, 終点座標 y)」

※ここの座標は円図形を囲うのに必要な箱の始点と終点となります。

start: 時計の 3 時を原点とした円弧の開始位置を度（° ）数で指定します

end: 時計の 3 時を原点とした円弧の終了位置を度（° ）数で指定します

fill: color の指定

width: 線の幅

弦（弓）を描画

始点終点を指定して弦（弓）を描画

chord(座標 , start, end, fill= , outline=)

座標:「(始点座標 x, 始点座標 y, 終点座標 x, 終点座標 y)」

※ここの座標は円図形を囲うのに必要な箱の始点と終点となります。

start: 時計の 3 時を原点とした円弧の開始位置を度（° ）数で指定します

end: 時計の 3 時を原点とした円弧の終了位置を度（° ）数で指定します

fill: color の指定

outline: color の指定

多角形を描画

polygon(座標 , fill= , outline=)

座標: (座標 x1, 座標 y1, 座標 x2, 座標 y3.....)

fill: color の指定

outline: color の指定

それではここまでの知識を使って簡単に図形を描画してみます。

凡例：pillow-2

図形を描画する凡例

python9-5.py

```
1   from PIL import Image,ImageDraw
2
3   im = Image.new('RGB', (500, 500),(255,240,230))
4   #背景をクリーム色で描画用のエリアを作成します。
5
6   draw = ImageDraw.Draw(im)
7   draw.ellipse((100,100,400,400),fill=(230,150,110))
8   draw.ellipse((190,155,220,210),fill=(0,0,0))
9   draw.ellipse((280,155,310,210),fill=(0,0,0))
10  draw.ellipse((200,210,300,290),fill=(230,50,12))
11  draw.rectangle((250,230,270,250),fill=(255,255,255))
12
13  draw.arc((150,140,260,250),-120,-60,fill=(0,0,0),width=5)
14  draw.arc((240,140,350,250),-120,-60,fill=(0,0,0),width=5)
15
16  draw.chord((190,240,310,350),start=-10,end=190,fill=(210,80,82))
17
18  im.show()
19
```

凡例：pillow-3

繰り返しを利用して図形を描画する凡例です。色や位置を変化させながら図形描画します。

python9-6.py

```
1  from PIL import Image,ImageDraw
2  import time
3
4  im = Image.new('RGB', (500, 500),(255,255,255))
5
6  draw = ImageDraw.Draw(im)
7
8  for x in range(0,255):
9      draw.ellipse((100 + x,100,400,400),fill=(x,0,255-x))
10
11 im.show()
```

2.9.3 イメージの読み込み

ここでは写真を読み込んで表示します。

凡例：pillow-2

プログラムで画像ファイルを指定して読み込むには画像ファイルがある場所を指し示すためにパスを指定します。現在プログラムの存在する場所からの相対で指定しますので、今回は「imagesample」フォルダー（ディレクトリ）内にある「sample01.jpg」を使ってみましょう。

python9-7.py

```
1  from PIL import Image
2
3  img = Image.open("imagesample/sample01.jpg")
4  # 画像の読み込み
5
6  img.show()  # 画像表示
7
```

さらに画像を 30° 回転させてみましょう。

python9-8.py

```
1  from PIL import Image
2
3  img = Image.open("imagesample/sample01.jpg")
4  # 画像の読み込み
5
6  #img = img.rotate(30)
7  #回転画像をもとの変数（オブジェクト）に入れています
8
9  img.show()  # 画像表示
10
```

プログラムでイメージ加工する場合に簡単にできそうなことをまとめてみました。

python9-8.py

```
from PIL import Image,ImageDraw,ImageFilter

img1 = Image.open("imagesample/sample01.jpg")
img2 = Image.open("imagesample/sample02.jpg")
img3 = Image.open("imagesample/sample02.jpg").resize((180,150))
# 画像の読み込み
img1.show()
img2.show()

img4 = Image.new("RGB", img1.size)
img4.paste(img3)
img4.show()

mask = Image.new("L", img1.size, 128)
im = Image.composite(img1, img4, mask)
im = Image.blend(img1, img4, 0.5)
im.show()

mask = Image.new("L", img1.size, 0)
draw = ImageDraw.Draw(mask)
draw.ellipse((0, 0, 179, 149), fill=255)
im = Image.composite(img4, img1, mask)
im.show()

mask = Image.new("L", img1.size, 0)
draw = ImageDraw.Draw(mask)
draw.ellipse((0, 0, 179, 149), fill=255)
mask_blur = mask.filter(ImageFilter.GaussianBlur(10))
im = Image.composite(img4, img1, mask_blur)
im.show()
# 画像表示
```

第 3 章

統計グラフ入門

3.1 matplotlib 入門

python でグラフを書く時の最も有名なライブラリは matplotlib といわれるものを利用します。matplotlib の基本的な利用方法を確認しましょう。

3.1.1 matplotlib の使い方

matplotlib ライブラリの中の pyplot(データプロットモジュール) を利用します。この時に import は「matplotlib.pyplot」となるので利用する際のライブラリ名と機能名の名称が長くなるため as によって別名をつけることが一般的です。

```
import matplotlib.pyplot as plt
```

Point
matplotlib ライブラリを利用するには次のコマンドで python の matplotlib ライブラリをインストールする必要があります。

```
pip install matplotlib
```

3.1.2 グラフの描画と表示

グラフに描画するためのデータを用意して表示してみましょう。

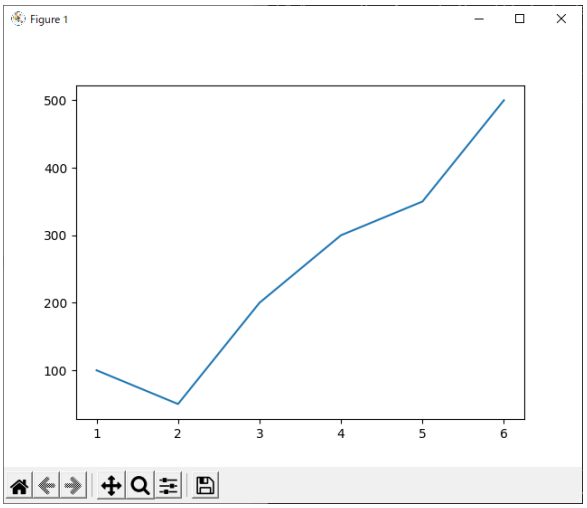
凡例：graph-1（折れ線グラフ）

6 要素のデータ [100,50,200,300,350,500] の折れ線グラフを描画してみましょう。基本的なグラフは x 軸の値と y 軸の値を指定することで簡単に表示することができます。

コード　3.1: class1-1.py

```
1  import matplotlib.pyplot as plt
2
3  x = [ 1, 2, 3, 4, 5, 6 ]
4  data = [ 100, 50, 200, 300, 350, 500 ]
5  plt.plot( x, data )
6  #plot を使ってデータを指定します。
7  plt.show()
```

実行結果



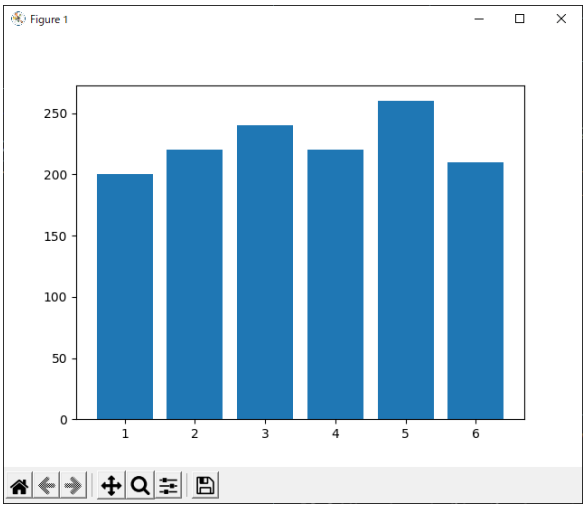
凡例：graph-2（棒グラフ）

6 要素のデータ [200,220,240,220,260,210] の棒グラフを描画してみましょう。基本的なグラフは x 軸の値と y 軸の値を指定することで簡単に表示することができます。

コード　3.2: class1-2.py

```
1  import matplotlib.pyplot as plt
2
3  x = [ 1, 2, 3, 4, 5, 6 ]
4  data = [ 200, 220, 240, 220, 260, 210 ]
5
6  plt.bar( x, data )
7  #bar を使ってデータを指定します。
8  plt.show()
```

実行結果



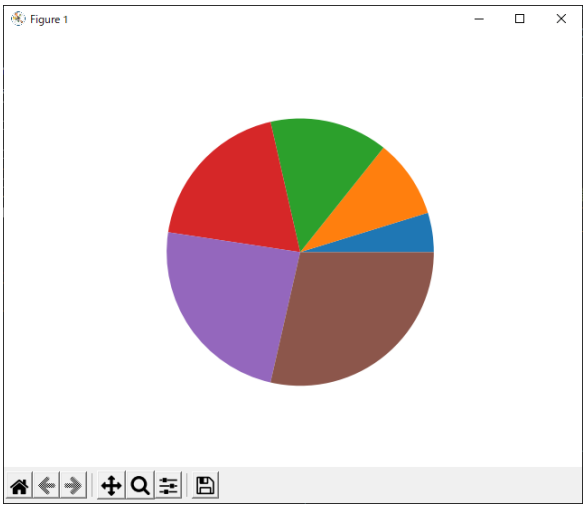
凡例：graph-3（円グラフ）

6 要素のデータ [100,200,300,400,500,600] の円グラフを描画してみましょう。

コード　3.3: graph1-3.py

```
1  import matplotlib.pyplot as plt
2
3  data = [ 100, 200, 300, 400, 500, 600 ]
4  plt.pie( data )
5  #pie を使ってデータを指定します。
6  plt.show()
```

実行結果



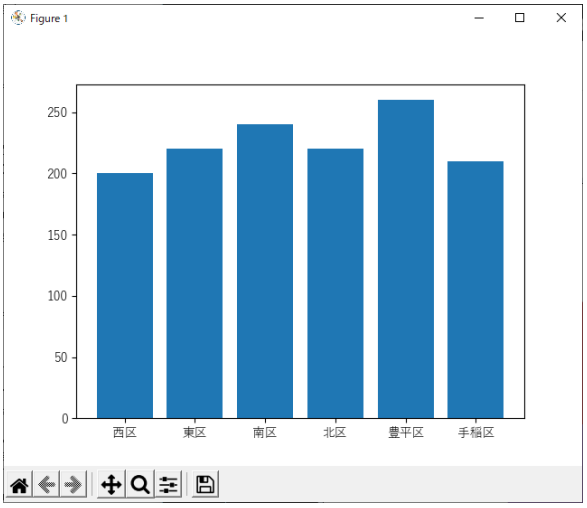
凡例：graph-4（データラベル）

graph-2 の棒グラフのデータラベルに日本語でラベルを設定してみましょう。

コード　3.4: graph1-4.py

```
1   import matplotlib.pyplot as plt
2   import japanize_matplotlib
3   #日本語 matplotlib を設定
4
5   x = [ 1, 2, 3, 4, 5, 6 ]
6   data = [ 200, 220, 240, 220, 260, 210 ]
7   label = [ '西区','東区','南区','北区','豊平区','手稲区']
8   #データラベルを指定することもできる
9   plt.bar( x, data ,tick_label = label)
10
11  plt.show()
```

実行結果



3.2　numpy 入門

行列要素等の複数要素に対して効率よく処理をするためのライブラリが numpy となります。

3.2.1 numpy の利用

numpy ライブラリの利用時の import は as を使って「np」という別名を使うのが慣例となっています。

```
import numpy as np
```

凡例：numpy-1

numpy を使ったデータ処理の一例を見てみましょう。ここではリストとの違いを確認してみます。

コード 3.5: numpy1-1.py

```
1  import numpy as np
2
3  #list の生成
4  a = [ 1, 2, 3, 4, 5]
5  print(a)
6  #配列要素の生成
7  b = np.array([ 1, 2, 3, 4, 5])
8  print(b)
9  #list の長さが 2 倍
10 print( a * 2 )
11 #配列要素の中身がそれぞれ 2 倍
12 print( b * 2 )
13 #配列要素の要素同士が乗算される
14 print( b * b )
```

実行結果

```
1  [1, 2, 3, 4, 5]
2  [1 2 3 4 5]
3  [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
4  [ 2  4  6  8 10]
5  [ 1  4  9 16 25]
```

凡例：numpy-2

numpy を使ったデータ生成の方法を見てみましょう。ここでは関数（メソッド）を使った配列要素の生成方法を確認します。

コード　3.6: numpy1-2.py

```
1  import numpy as np
2
3  #range とよく似ていますが、整数以外も生成可能
4  print(np.arange(5))
5  print(np.arange(0, 2, 0.4))
6  #linspace を使うと値を等分したリストを生成可能
7  print(np.linspace(0,100,5))
8  #10 個の配列要素としての乱数を同時生成
9  print(np.random.rand(10))
```

実行結果（例）

※乱数はそれぞれ出力される内容が異なります。

```
1  [0 1 2 3 4]
2  [0.  0.4 0.8 1.2 1.6]
3  [  0.  25.  50.  75. 100.]
4  [0.61920954 0.29684569 0.94695251 0.79888711 0.77976444 0.10481961
5   0.83043277 0.03716237 0.26022265 0.22519305]
```

凡例：numpy-3

numpy を使った条件判定を見てみましょう。配列要素として乱数を生成し特定の値がいくつ含まれているか数えてみましょう。

コード　3.7: numpy1-3.py

```
1   import numpy as np
2
3   ransu = np.random.randint(1, 4, 10)
4   print(ransu)
5   #3 のところを確認する
6   print(ransu == 3)
7   #3 の数を数える
8   print(np.count_nonzero(ransu == 3))
```

実行結果（例）
※乱数はそれぞれ出力される内容が異なります。

```
1   [2 3 1 1 1 2 2 2 3 3]
2   [False  True False False False False False False  True  True]
3   3
```

【高等学校情報科「情報 1 教員研修用教材 (本編)】第 3 章で用いられている確定モデルと確率モデルに出てくる numpy と matplotlib について今まで学習した内容と合わせて確認しましょう。

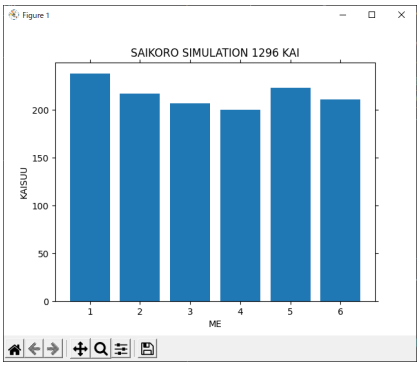
凡例：numpy-4

参考資料「【高等学校情報科「情報 1」教員研修用教材 (本編)】第 3 章」１３９ページのサンプルプログラムを修正し、描画を見ながら確率モデルを確認しましょう。

コード　3.8: numpy1-4.py

```
1   import numpy as np # 整数をカウントするための関数呼び出し
2   import matplotlib.pyplot as plt # グラフプロットの呼び出し
3
4   for times in range(1,11):
5       saikoro = np.random.randint(1, 6+1, 6 ** times) # サイコロを振る
6       deme = [ ] # 出目の数を数える配列
7       for i in range(6):
8           deme.append(np.count_nonzero(saikoro==i+1))
9       # 数を数えて配列に追加
10      left = [1, 2, 3, 4, 5, 6] # グラフの左方向の値指定用
11      plt.cla()
12      plt.title("SAIKORO SIMULATION " + str(6 ** times) + " KAI")
13                          # グラフのタイトル
14      plt.xlabel("ME") #X 軸のラベル
15      plt.ylabel("KAISUU") #Y 軸のラベル
16      plt.bar(left, deme, align="center") # グラフをプロット
17      plt.draw()
18      plt.pause(2)
19  plt.show()
```

実行結果



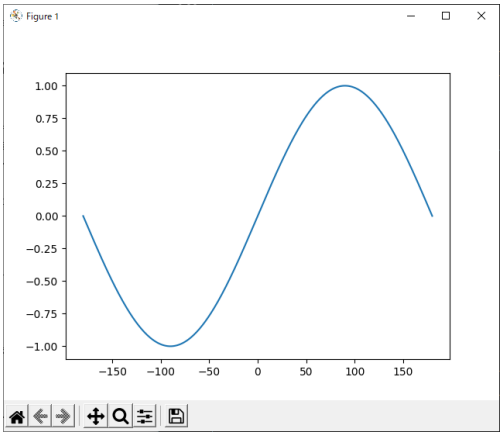
凡例：numpy-5

numpy と matplotlib を使って三角関数の sin カーブを書いてみましょう。
コンピューター内部の角度は「° 」ではなく radian 単位を使うことが多いです。0° から時計回り半周で円周率π (3.141592・・・・) となり、一周で 2 π となるように表現される角度の単位です。

コード　3.9: numpy1-5.py

```
1   import numpy as np
2   import matplotlib.pyplot as plt
3
4   #sin カーブの描画のための-180° 〜180 までの radian 設定
5   kakudo = np.linspace( -np.pi, np.pi, 361)
6   x = np.linspace(-180,180,361)
7   plt.plot(x, np.sin(kakudo))
8   plt.show()
```

実行結果



このような数値計算を効率よく行うためのライブラリが numpy ライブラリとなります。

3.3 WebAPI の利用

ここでは WebAPI による JSON データの取得を行い、グラフで視覚化してみます。

3.3.1 外部 JSON データの読み込み

まずはファイルに保存した外部 JSON データの読み込みを行い、データにアクセスしてみましょう。

凡例：json1-1（JSON データの読み込み）

graph-2 の棒グラフのデータラベルに日本語でラベルを設定してみましょう。

コード 3.10: jsondatasimple.json

json ファイルはコメントを記載できないので連想配列要素とリスト要素を確認しておいてください。

```
{
    "class": "2 年 3 組",
    "name": "佐々木",
    "score": [
        60,
        70,
        50,
        80,
        100
    ]
}
```

json ライブラリを import し、json ファイルをオープン後 json.load で読み込むことで dict 型（連想配列）の値が取得されるので、左辺の変数に代入します。

コード　3.11: json1-1.py

「jsondatasimple.json」を読み込んで各要素を取り出し、リストの合計を求めます。

```
1   import json
2
3   jsonfile = open('jsondatasimple.json' , 'r', encoding='utf-8')
4   #JSON データ（dict 型）として取り出す
5   loaded_data = json.load(jsonfile)
6
7   #JSON データの class 要素と name 要素を取り出す
8   print(loaded_data['class'])
9   print(loaded_data['name'])
10
11  #JSON データの score 要素のリストを取り出し合計を求める
12  total = 0
13  for x in loaded_data['score']:
14      total = total + x
15
16  print("goukei = " + str(total))
```

実行結果

```
1   2 年 3 組
2   佐々木
3   goukei = 360
```

3.3.2　WebAPI の利用

WebAPI で公開されている政府や行政のデータは次のようなものがあります。

$https://www.e-stat.go.jp/$
$https://data.pf-sapporo.jp/$

これらのデータを取得し利用する方法は url を利用して json データとしてダウンロードし dict 型（連想配列）のデータとして読み込むことで可能となります。

今回利用するのは $https://ckan.pf-sapporo.jp/dataset/suikeijinkou$ で取得できる札幌市の「将来推計人口の推移（平成 17〜47 年）」です。

Excel で csv ファイルを読み込むと次の通りとなります。

_id	項目	17年（2005）	22年（2010）	27年（2015）	32年（2020）	37年（2025）	42年（2030）	47年（2035）
1		1)	1)	2)	2)	2)	2)	2)
2		千人（%）	千人（%）	千人（%）	千人（%）	千人（%）	千人（%）	千人（%）
3	総人口	1,881（100.0）	1,914（100.0）	1,937（100.0）	1,933（100.0）	1,911（100.0）	1,871（100.0）	1,818（100.0）
4	年少人口（0～14歳）	234（12.4）	224（11.7）	217（11.2）	205（10.6）	189（9.9）	171（9.1）	156（8.6）
5	生産年齢人口（15～64歳）	1,318（70.1）	1,292（67.5）	1,234（63.7）	1,179（61.0）	1,139（59.6）	1,089（58.2）	1,024（56.3）
6	老年人口（65歳以上）	325（17.3）	392（20.5）	486（25.1）	549（28.4）	583（30.5）	611（32.7）	638（35.1）
7								
8	注：1）国勢調査人口による。総人口には年齢不詳も含む。2）推計値である。							
9	＜資料＞総務省統計局「国勢調査」、市長政策室政策企画部企画課							

このデータを WebAPI による読み込みをしてみましょう。

凡例：webapi1-1（WebAPI による JSON データの読み込み）

将来推計人口の推移（平成 17～47 年）の WebAPI 用の URL からデータを読み込むプログラムです。変数 content に dict 型のデータとして取り込まれます。

コード　3.12: webapi1-1.py

```
1   import urllib
2   import urllib.request
3   import json
4
5   url = 'https://ckan.pf-sapporo.jp/api/3/action/datastore_search?
6          resource_id=710b0236-6b28-441e-8500-f5c9be2e4c4a'
7   response = urllib.request.urlopen(url)
8   content = json.loads(response.read().decode('utf8'))
9
10  print(content)
```

実行して結果を確認してください。

WebAPI で用いられる JSON による文書構造の例は次の通りです。

3.3.1: WebAPI で利用される JSON の構造例

```
1   #辞書構造
2   {
3       "help": --
4       "success": --
5       "result": {
6           辞書構造
7           "records":[
```

```
8                リスト構造の中に辞書構造
9            ],
10           "fields":[
11               リスト構造の中に辞書構造
12           ],
13           "_links":
14               辞書構造
15           "total": 9,
16           "total_was_estimated": false
17   }
```

それでは具体的にＷｅｂＡＰＩのデータ構造を見てみましょう。

3.3.2: JSON 札幌市の将来推計人口の推移（平成 17〜47 年）の抜粋

```
1   {
2       "help": "https://ckan.pf-sapporo.jp/api/3/action/help_show?
3                name=datastore_search",
4       "success": true,
5       "result": {
6           "include_total": true,
7           "limit": 100,
8           "records_format": "objects",
9           "resource_id": "710b0236-6b28-441e-8500-f5c9be2e4c4a",
10          "total_estimation_threshold": null,
11          "records": [
12              {
13                  "_id": 1,
14                  "項目": "",
15              ----省略----
16              },
17              {
18                  "_id": 2,
19                  "項目": "",
20              ----省略----
21              },
22              {
```

```
                "_id": 3,
                "項目": "総人口",
                "17 年 (2005) ": "1,881 (100.0) ",
                "22 年 (2010) ": "1,914 (100.0) ",
                "27 年 (2015) ": "1,937 (100.0) ",
                "32 年 (2020) ": "1,933 (100.0) ",
                "37 年 (2025) ": "1,911 (100.0) ",
                "42 年 (2030) ": "1,871 (100.0) ",
                "47 年 (2035) ": "1,818 (100.0) "
            },
            {
                "_id": 4,
                "項目": "年少人口 (0〜14 歳) ",
            ----省略----
            },
            ----省略----
        ],
        "fields": [
            {
                "id": "_id",
                "type": "int"
            },
            {
                "id": "項目",
                "type": "text"
            },
            {
                "id": "17 年 (2005) ",
                "type": "text"
            },
            {
                "id": "22 年 (20010) ",
                "type": "text"
            },
            ----省略----
        ],
```

```
59            "_links": {
60                ----省略----
61            },
62            "total": 9,
63            "total_was_estimated": false
64        }
65    }
```

このデータ構造からデータを取得する場合は

content 辞書データから content[”result”] で値を取り出すと”result”で参照される dict 型のデータが取得できます。

content[”result”] から content[”result”][”records”] で参照される辞書データ 9 個のリストデータが取得できます。（角カッコ「[]」で囲まれた要素はリストとして扱われます。）

これらのデータから総人口の推計を取り出すには content[”result”][”records”][2] を指定することで総人口の連想配列を取得することができます。

凡例：webapi1-2（WebAPI による JSON データの読み込み）

将来推計人口の推移（平成 17〜47 年）の WebAPI 用の URL からデータを読み込むプログラムです。変数 content に dict 型のデータとして取り込まれます。

コード　3.13: webapi1-2.py

```
1  import urllib
2  import urllib.request
3  import json
4
5  url = 'https://ckan.pf-sapporo.jp/api/3/action/datastore_search?
6         resource_id=710b0236-6b28-441e-8500-f5c9be2e4c4a'
7  response = urllib.request.urlopen(url)
8  content = json.loads(response.read().decode('utf8'))
9
10
11 #値を取り出すための変数を用意
12 recode = content["result"]["records"][2]
13 #dict データを順番に取り出す
14 for v in recode:
15     print(v + ":" + str(recode[v]))
```

実行結果

```
1  _id:3
2  項目:総人口
3  17 年 (2005) :1,881 (100.0)
4  22 年 (2010) :1,914 (100.0)
5  27 年 (2015) :1,937 (100.0)
6  32 年 (2020) :1,933 (100.0)
7  37 年 (2025) :1,911 (100.0)
8  42 年 (2030) :1,871 (100.0)
9  47 年 (2035) :1,818 (100.0)
```

値を取り出す 13 行目から 16 行目は次のように items（）メソッドを利用して key と値を取り出すこともできます。

```
1  recode = content["result"]["records"][2]
2  #record の items() メソッドで key と値を k と v に入れることができる
3  for k, v in recode.items():
4      print(k + ":" + str(v))
```

ここまでに取得したデータを使って推移および将来推計をグラフ化しましょう。

前述の「webapi1-3.py」の 15,16 行目の人口推計データを基にして数値に変換し matplotlib に設定してみましょう。

凡例：webapi1-3（WebAPI による JSON データの読み込み）

将来推計人口の推移（平成 17〜47 年）の WebAPI 用の URL からデータを読み込むプログラムです。変数 content に dict 型のデータとして取り込まれます。

コード　3.14: webapi1-3.py

```
1  import urllib
2  import urllib.request
3  import json
4  import re
5  import matplotlib.pyplot as plt
6  import japanize_matplotlib
7
8  url = 'https://ckan.pf-sapporo.jp/api/3/action/datastore_search?
9        resource_id=710b0236-6b28-441e-8500-f5c9be2e4c4a'
10 response = urllib.request.urlopen(url)
11 content = json.loads(response.read().decode('utf8'))
12
13 #値を取り出すための変数を用意
14 recode = content["result"]["records"][2]
15 #データラベルを初期化
16 label = []
17 #グラフデータを初期化
18 data = []
19
20 listdata = list(recode.items())
21 listdata.pop(0)
22 listdata.pop(0)
23 for k, v in listdata:
24     l = re.search('[0-9]+年', k).group()
25     label.append(l)
26     v = v.replace(',','')
27     n = int(re.search('[0-9]+', v).group())
28     data.append(n*1000)
29 print(label, data)
30 x = [ 1, 2, 3, 4, 5, 6, 7 ]
31 plt.bar( x, data ,tick_label = label)
32
33 plt.show()
```

もう一つ気象庁のデータを読み取り札幌エリアの天気概況を表示してみましょう。

凡例：webapi1-4（WebAPI による JSON データの読み込み）

気象庁の札幌エリアのデータを読み取り表示するプログラムを書いてみましょう。

コード　3.15: webapi1-4.py

```
1  import urllib
2  import urllib.request
3  import json
4  #詳細データ
5  url = 'https://www.jma.go.jp/bosai/forecast/data/forecast/
6        016000.json'
7  #概況
8  url = 'https://www.jma.go.jp/bosai/forecast/data/
9        overview_forecast/016000.json'
10 response = urllib.request.urlopen(url)
11 content = json.loads(response.read().decode('utf8'))
12
13 print(content["headlineText"])
14
```

実行結果 (ある日の結果)

**空知地方では、２９日は大雨による低い土地の浸水や河川の増水に、石狩・空知・後志地方では、２９日は落雷や突風、ひょう、急な強い雨に
、３０日にかけて濃い霧による交通障害に注意してください。**

第 4 章

オブジェクト指向入門

プログラミングは今まで見てきた手続きを関数としてまとめながら記述する方法で作成されています。

プログラムの規模が大きくなるともっと効率よく書く方法が考えられるようになりました。

この方法の一つがオブジェクト指向プログラミングと言われています。オブジェクト指向プログラミングではプログラムコードの再利用を効率よく行うことできることがメリットです。

この中で重要なキーワードがクラスといわれる考え方で扱うデータと処理（メソッド）を一体化した仕組みを指します。

4.1 クラス入門

4.1.1 クラスとは？

クラスとはデータとデータを処理すべき手続きをひとまとめにして、型として定義したものです。型は中身の無いものなので、たい焼きの型を思い出していただければと思います。たい焼きの型に小麦粉やクリームなどの材料を入れ加熱調理することでたい焼きになりますが、このことをインスタンス化（実体化）と言います。

ここでは概略のみの説明となりますので、厳密な定義は各解説書や Web を確認してください。

まずは簡単に学生の名前と１科目の点数と成績を管理するためのクラスを作りたいと思います。

そのためにはクラスの書き方から見ていきましょう。

一番簡単なクラスの定義は次の通りです。一般的にクラス名は英大文字から書くことが多いです。

```
class クラス名:
    pass
```

本来ブロックには１行以上プログラムを記述する必要がありますが、pass という何もしないというプログラムコードを書くことでなにもしないブロックを記述することができます。

4.1.2 実体（インスタンス）

クラスをプログラムから利用するためには変数にクラス定義された型から実体を作る必要がありますので、そのやり方を見ていきましょう。

凡例：class-1

なにも機能を持たないクラスの定義を利用したデータの管理を確認します。
変数 a も b もそれぞれ Person クラスの実体ですが、別のデータを保管管理することになることを確認しましょう。
それぞれの実体に持たせることができる変数は任意のものを定義することができます。
クラス定義は class の後にクラス名を書きますが、クラス名の頭文字は大文字にする慣例があります。

コード　4.1: class1-1.py

```
1   #クラス定義
2   class Person:
3       pass   #なんの命令もない空クラス（型）
4
5   #実体の作成　１つ目　変数 a、二つ目　変数 b
6   a = Person()
7   b = Person()
8   #１つ目の実体の x という変数に 10 を代入
9   a.x = 10
10  #２つ目の実体の x という変数に 20 を代入
11  b.x = 20
12  #１つ目 a と二つ目 b の実体の x という変数を出力
13  print("a.x = ", a.x)
14  print("b.x = ", b.x)
```

実行結果

```
1   a.x =  10
2   b.x =  20
```

Point
何度か実体という言葉がでてきますが、凡例 class-1 では 6 行目と 7 行目でそれぞれ別のものが作られていると考えると自然かと思います。なにかしら得体のしれないデータを格納できる変数 a と b をオブジェクト指向ではインスタンス（実体）ということを覚えておきましょう。

4.1.3 クラス変数

凡例 class-1 では実体１と実体２でそれぞれ別のデータを扱うことがわかりましたが、実体間で共通のデータを扱う方法を考えます。（厳密には異なりますが、グローバル変数のようなもの）

凡例：class-2

クラス変数と言って一つのクラスでは値として一つだけの変数を定義することができます。変数 a も b もそれぞれ Person クラスの実体ですが、クラス変数では共通のデータを管理することになることを確認しましょう。
それぞれの実体に持たせることができる変数は任意のものを定義することができます。

コード　4.2: class1-2.py

```
1   class Person:
2       pass
3
4   a = Person()
5   b = Person()
6   #クラス名の後に変数を記述するとクラス変数となる。
7   Person.x = 10
8   #また実体の__class__を利用するとクラス変数となる。
9   b.__class__.x = 20
10  #凡例 class-1 と同じ書き方ですが、それぞれクラス変数と解釈されると
11  #実体１と実体２で共通の値を参照していることがわかります。
12  print("a.x = ", a.x)
13  print("b.x = ", b.x)
```

実行結果

```
1   a.x =  20
2   b.x =  20
```

4.1.4 メソッド（オブジェクト内の処理）

ここまでで説明したデータを持たせることは見ることができましたが、処理（手続き）を持たせたるとはどういうことか見てみたいと思います。

クラス内の処理はメソッド（クラス内の関数の呼び方）といい実体ごとのデータを処理することができます。

凡例：class-3

ここでは数学（Mathematics）の点数のデータを処理し合格”good”か不合格”bad”を判定する手続きをクラスに記述します。
ここでメソッド set_m は Student クラスのブロック中に書かれます。set_m の第一引数は自分自身の実体となりますので、実体ごとに管理するデータは self に「.」をつけることで設定・参照することができます。

コード　4.3: class1-3.py

```
1   class Student:
2       #数学の点数を管理する処理
3       def set_m(self,m):
4           #第二引数は 17 行目以降の set_m の引数となり
5           #実体のデータ m として管理・操作できるように保存
6           self.m = m
7           if m >= 50:
8               #合否を実体のデータ grading として設定
9               self.grading = "good"
10          else:
11              self.grading = "bad"
12
13  a = Student()
14  b = Student()
15  c = Student()
16  #Student の中身を知らなくても set_m が何をするのか知っていれば OK
17  a.set_m(75)
18  b.set_m(48)
19  c.set_m(95)
20  print("a ", a.m ,a.grading)
21  print("b ", b.m ,b.grading)
22  print("c ", c.m ,c.grading)
```

実行結果

```
1   a  75 good
2   b  48 bad
3   c  95 good
```

クラスを利用するとそのクラスの中身を知らなくてもどのような結果を得られるのかということを知っているだけで利用することが可能になります。

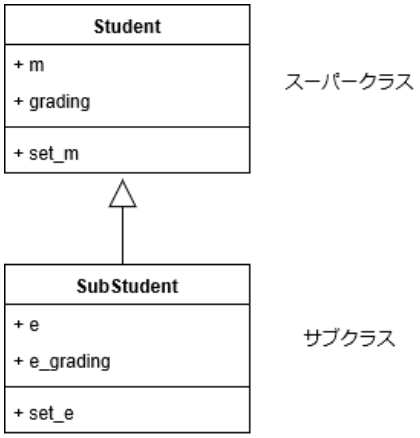
Point
多くのプログラミングではクラスの形（定義）とそのクラスを利用する方法が一緒に書かれているため、クラスとして書かれたプログラムの内容も理解しておく必要があると思われがちですが、いままで見てきた配列（リスト）のように.append や pop など使い方だけ知っていれば良いと考えておいてください。

4.1.5 クラスの継承

オブジェクト指向ではクラスの継承というコードの再利用のための仕組みがあります。

ここでは簡単に継承の形式を確認してみましょう。

オブジェクト指向ではベースとなる基盤機能をもっているクラスをスーパークラス、基盤の機能を利用するクラスをサブクラスといいます。



サブクラスではスーパークラスの機能を利用することができますので、Student クラスで定義されている set_m メソッドや grading といったデータは SubStudent クラスで定義していなくても利用できます。

凡例：class-4

凡例 class-3 の Student は（Mathematics）クラスはそのままに（English）の点数のデータを処理し合格”good”か不合格”bad”を判定する処理 set_e をサブクラス SubStudent に記述します。

コード　4.4: class1-4.py

```
1   class Student:
2       def set_m(self,m):
3           self.m = m
4           if m >= 50:
5               self.grading = "good"
6           else:
7               self.grading = "bad"
8   #クラス宣言の括弧の中に利用したい機能のスーパークラスを記載します。
9   class SubStudent(Student):
10      def set_e(self,e):
11          self.e = e
12          if e >= 50:
13              #合否を実体のデータ e_grading として設定
14              self.e_grading = "good"
15          else:
16              self.e_grading = "bad"
17  a = SubStudent()
18  a.set_m(80)
19  a.set_e(30)
20  print("a Mathematics", a.m ,a.grading)
21  print("a English", a.e ,a.e_grading)
```

実行結果

```
1   a  Mathematics 80 good
2   a  English 30 bad
```

第 5 章

練習問題解答

practice1.py

```
1  x = 3
2  y = 4
3  print ( (x * y) /2)
```

practice2.py

```
1  x = 60
2  if x >= 50:
3      print ( " goukaku " )
4  else:
5      print ( " fugoukaku " )
```

practice3.py

```
1  x = 1
2  y = 2
3  #それぞれ大小等しいとなる値は何でも構いません
4  if x > y:
5          print("x is large.")
6  elif x < y:
7          print("y is large.")
8  else:
9          print("x and y are equal.")
```

practice4.py

```
x = 60
if 100 >= x >= 80:
    print ( "A" )
elif x >= 60:
    print ( "B" )
elif x >= 40:
    print ( "C" )
elif x >= 0:
    print ( "D" )
else:
    print ( "error" )
```

practice5.py

```
x = input ( "math = " )
m = int ( x )
x = input ( "english = " )
e = int ( x )
z = input ( "social = " )
s = int ( z )
print ( (m + e + s) / 3 )
```

practice6.py

```
for x in range(1000):
    print( "wan" )
    print( "nyan" )
```

practice7.py

```
1  sum = 0
2  for x in range(1,1001):
3      sum = sum + x
4  print ( sum )
5
6  sum = 0
7  for x in range(1, 11):
8      sum = sum + 1 / 2 ** x
9  print ( sum )
```

practice8.py

```
1  y = 0
2  for n in range(1,6):
3      x = input ( "subject”  + str(n) + " = " )
4      y = y + int( x )
5  print( y / 5 )
```

practice9.py

```
1  subjects = ["Mathematics" ,"English","Social","Japanese","Scientific"]
2  scores = []
3  for s in subjects:
4      x = input( s + " = " )
5      y = int ( x )
6      if y > 100 or y < 0 :
7          y = -1
8          print ( "error" )
9      scores.append( y )
10
11 for n in range(0,5):
12     if scores[n] == -1:
13         mes = "error"
14     elif scores[n] >= 80:
15         mes = "excelent"
16     elif scores[n] >= 60:
17         mes = "good"
18     elif scores[n] >= 40:
19         mes = "passing"
20     else:
21         mes = "failing"
22     print ( subjects[n] + ":" , scores[n], mes )
```

practice10.py

```
1  students_scores = []
2  #配列の準備
3  students_scores.append([60, 80]) #A さんのデータを追加
4  students_scores.append([70, 90]) #B さんのデータを追加
5  students_scores.append([80, 100]) #C さんのデータを追加
6  students_scores.append([50, 70]) #D さんのデータを追加
7  students_scores.append([60, 60]) #E さんのデータを追加
8
9  m = 0
10 e = 0
11
12 for v in students_scores:
13     m = m + v[0]
14     e = e + v[1]
15
16 print("数学の平均点", m / len(students_scores))
17 print("英語の平均点", e / len(students_scores))
```

practice11.py

```
1  subjects = ["Mathematics" ,"English","Social","Japanese","Scientific"]
2  scores = []
3  for n in range(3):
4      print(n + 1, "人目の入力")
5      scores.append([])
6      for s in subjects:
7          x = input( s + " = " )
8          y = int ( x )
9          if y > 100 or y < 0 :
10             y = -1
11             print ( "error" )
12         scores[n].append( y )
13 s_l = len(subjects) #科目数の取得
14
15 for m in range(s_l):
16     sum = 0
17     m_l = len(scores) #人数の取得
18     for n in range(m_l):
19         sum = sum + scores[n][m]
20     print(subjects[m] , "の平均点", sum / 3)
```

practice12.py

```
1  def func1(n):
2      a = (n - 65)**2
3      return a
4
5  def func2():
6      t = 0
7      for i in range(5):
8          x = input()
9          y = int(x)
10         t = t + func1(y)
11     return t
12
13 z = func2()
14 print( z / 5 )
```

practice13.py

```
#グローバル変数（配列）の準備
subjects = ["Mathematics" ,"English","Social","Japanese","Scientific"]
scores = []
#関数 input_data の記述
def input_data():
    for n in range(3):
        cnt = 0
        scores.append([])
        name = input("Name = ")
        scores[n].append(name)
        for s in subjects:
            x = input( s + " = " )
            y = int ( x )
            if y > 100 or y < 0 :
                y = -1
                print ( "error score" )
            else:
                cnt = cnt + 1
            scores[n].append( y )
        scores[n].append(cnt)
#関数 get_average の記述
def get_average(m):
    sum = 0
    for n in range(1,6):
        if scores[m][n] != -1:
            sum = sum + scores[m][n]
    return sum / scores[m][6]

#プログラムはここから開始
input_data()
for m in range(3):
    ave = get_average(m)
    print(scores[m][0],"の",scores[m][6],"科目の平均点",ave)
```