2021 年 7 月

プログラミング入門（Python）

初めてのプログラミング　Ver1.2

大原簿記情報専門学校 札幌校

佐々木博幸

目次

第 1 章

プログラミング入門（はじめに）

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

今回利用する python 言語の環境設定は次の通りとなります。

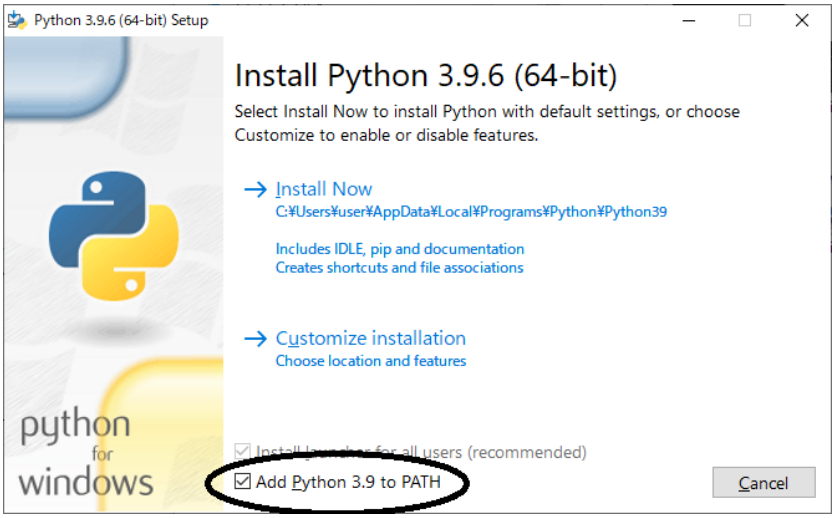
今回の教室内の設定およびインストールは済ませていますが、ご自宅でインストールする場合は次の手順が参考になります。

https：*//www.python.jp/install/windows/install_py*3*.html*

手順に従い *https*：*//www.python.org/downloads/windows/* より、パッケージをダウンロードします。

- ・Download Windows installer（32-bit）
- ・Download Windows installer（64-bit）

いずれかを選択しダウンロードします。



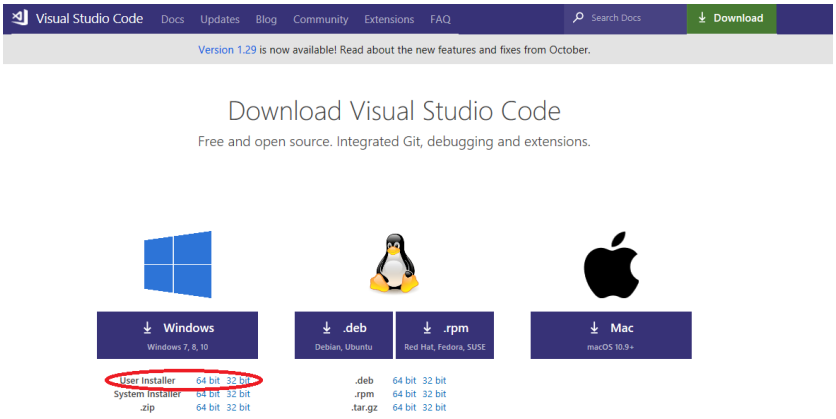
※画面は 64bit 版の 3.9.6 です。

1.「Add Python 3.x to PATH」 をチェックしてください。
2. Install now をクリックしてインストールをしてください。

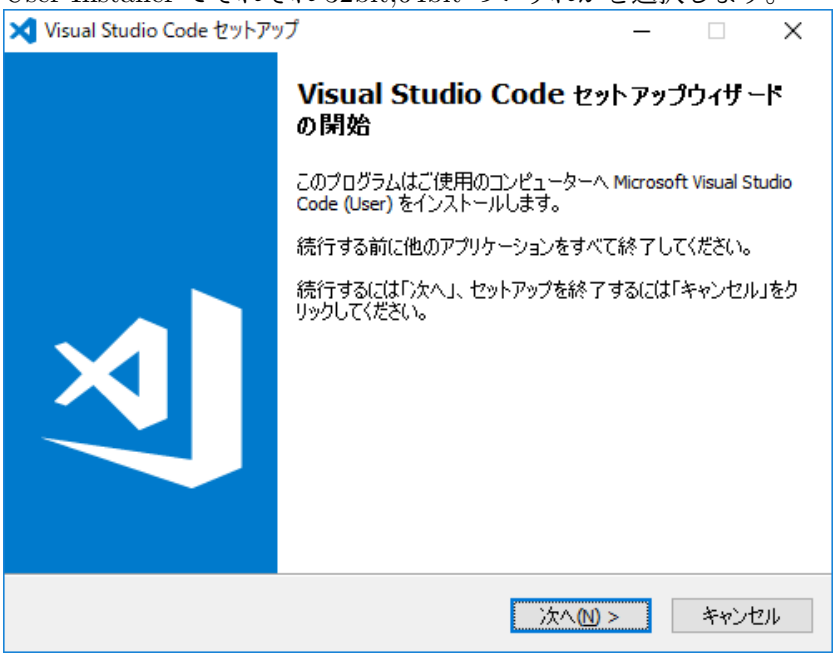
1.1.2　エディター（Visual Studio ode）準備

次の URL を参照し「Visual Studio Code（vscode)」をダウンロードを行います。

https : //code.visualstudio.com/download



User Installer でそれぞれ 32bit,64bit のいずれかを選択します。

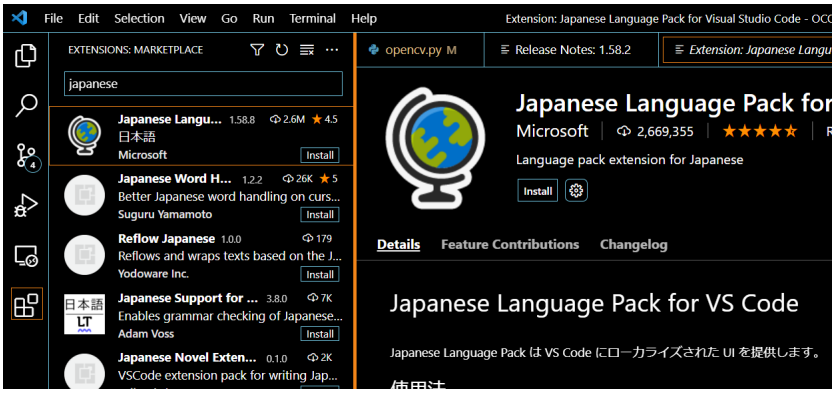


あとはインストールするだけです。

1.1.2.1　vscode の環境設定

今回のプログラミングで利用する python の拡張機能をインストールします。

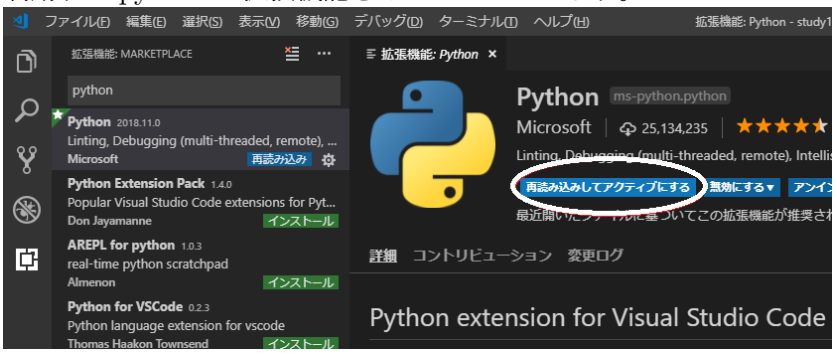
日本語表示のために日本語拡張機能のインストールを行います。



起動画面左下の拡張機能をクリックし、検索キーワードに「japanese」と入力し「japanese Language Pack for Visual.....」を選択し「install」します。

ここで vscode を再起動します。

次に日本語拡張機能と同様に検索キーワードに「python」を入力することで今回プログラミングで利用する python の拡張機能をインストールします。



インストール後に再読み込みしてアクティブにするを選択して python 拡張を有効にします。

これでインストールは完了です。

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。

python では一つ一つの命令を確認しながら実行できる「対話実行モード（インタラクティブモード)」とプログラムファイルを作成して実行する「スクリプトによる実行」を使うことができます。

これらのモードを利用することでコンピューターへ命令するというプログラミングの基礎を効率よく学ぶことができます。

凡例：python の実行方法

対話実行の場合は入力待ちとなり「>>>」の後ろに直接命令や計算式を入力することができます。

2.1.1: 対話実行

```
>>> x = 10
>>> print(x)
10
>>> exit() #または　Ctrl+z で終了します。
```

プログラム（スクリプト）を記述して実行する場合は「python」コマンドに続いて保存したファイル名を指定します。
Visual Studio Code（vscode）で python 拡張を導入している場合は「F5」キーで実行することができます。

2.1.2: スクリプトによる実行

```
C:\Users\username> python ファイル名.py
10
C:\Users\username>
```

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

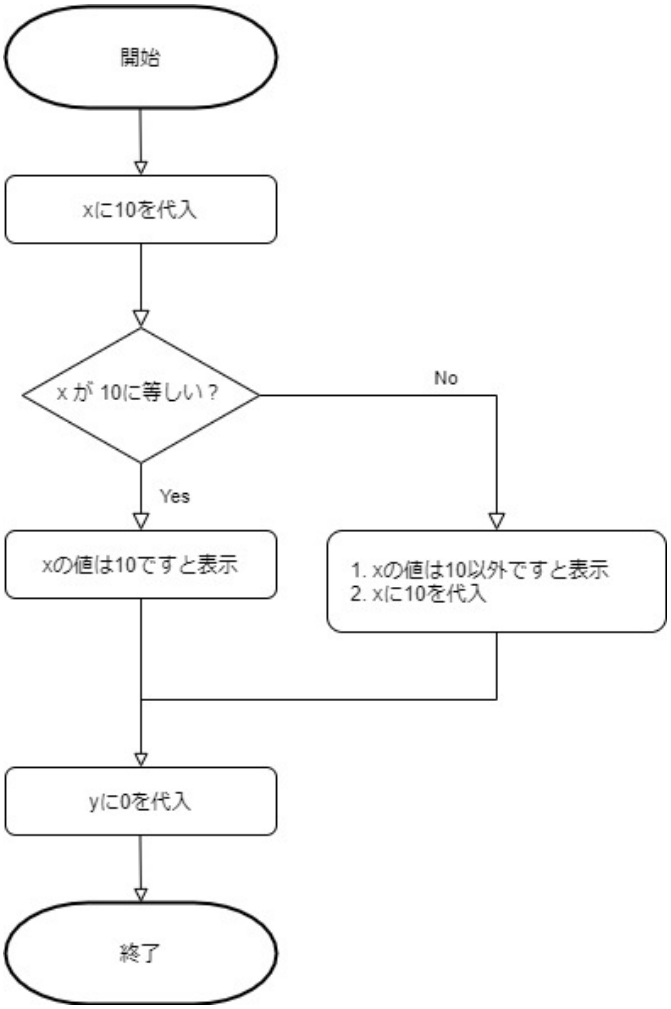
インデント（スペース・空白）には意味がありますので、必要のないスペースはエラーになります。

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整理し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

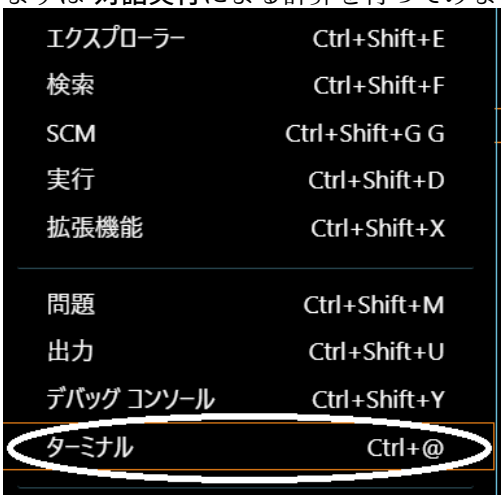
```
x = 10
if x == 10 : #x が 10 だったらの処理を書くためには「:」でブロックを開始
    print("x の値は 10 です")
    #他の処理はなにもしません。
else : #x の値が 10 以外だったらの処理 2 行が x の値が 10 以外の処理ブロック
    print("x の値は 10 以外です")
    x = 10 #x の値が 10 以外なので 10 に設定します。
y = 0 #これは if 文のどちらを通過しても必ず実行されます。
```

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。

今回の授業で利用する Visual Studio Code では python で記述するインデントを直前の行と自動で合わせる機能を持っています。

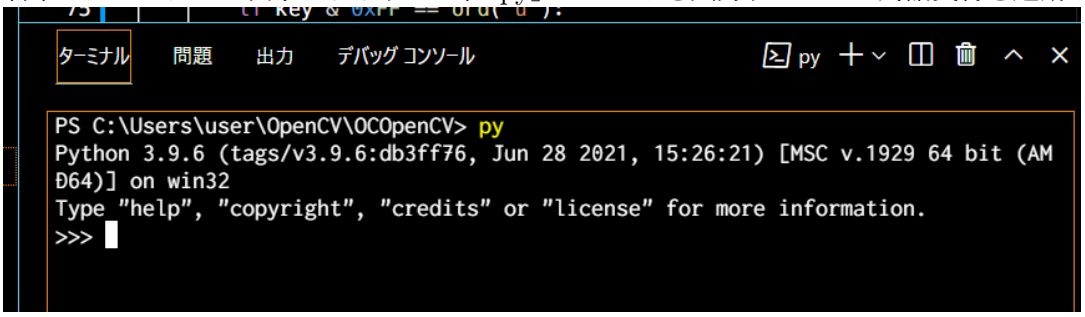
2.1.3　python を使った計算

まずは **対話実行**による計算を行ってみましょう。



vscode の「表示」メニューから「ターミナル」を選択します。

右下にターミナルが表示されますので、「py」コマンドを入力することで対話実行を起動します。



次の例の通り「>>>」に続いて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。

2.1.3:　対話実行 python の起動：例

```
1  c:\\〜> py
2  Python 3.9.6 〜省略〜 on win32
3  Type "help", 〜省略〜
4  >>> 1 + 2 #>>>の後に計算式を入力します。
5  3
6  >>> 3 * 4
7  12
8  >>> 1 / 3
9  0.3333333333333333
```

```
10  >>> 1 +  1 / 4
11  1.25
12  >>> 4 ** 2
13  16
14  >>> ( 1 + 2 + 3 ) / 2
15  3.0
```

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。

2.1.4:　変数の確認

```
16  >>> x = 10
17  >>> x
18  10
19  >>> x / 4
20  2.5
21  >>> y = x - 5
22  >>> y
23  5
```

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。

※　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。

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）

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。

2.1.5:　イコールの考え方等式

```
>>> 10 = x
  File "<stdin>", line 1
SyntaxError: can't assign to literal
 ※ SyntaxError（エラー）になります。

>>> x = x - 1
>>> x
9
```

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に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。

2.1.6:　桁の多い計算

```
>>> 2 ** 32
4294967296
>>> 2 ** 64
18446744073709551616
>>> 2 ** 128
 （実行結果を確認してください）
  340 澗 2823 溝 6692 穣 0938 じょ 4634 垓 6337 京 4607 兆 4317 億 6821 万 1456
>>> 2 ** 256
 （実行結果を確認してください）
```

41　　**！？（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

```
1   >>> x = 1
2   >>> x = x + 2
3   >>> x = x + 3
4   >>> x = x + 4
5   >>> x = x + 5
6   >>> x
7   15
```

※ ヒント「↑」キーを押すと前の行が表示され編集をすることができます。

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 **対話実行**では結果の出力はできていましたが、明示的に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
```

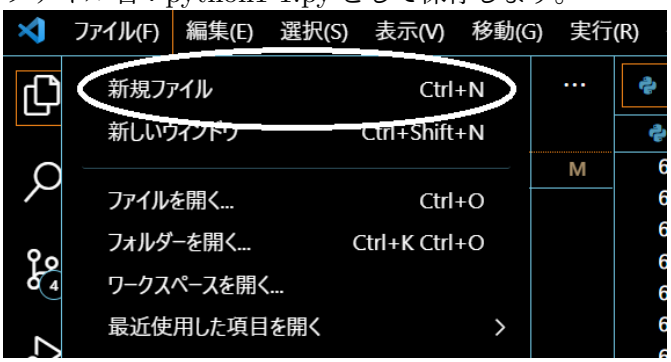
「print(x)」も「x」も x の値が出力されます。違いについては次のセクション「ファイルに保存」の中で説明します。

2.2.3 ファイルへの保存

これらのプログラムをひとまとまりの処理として書いてファイルに保存してみましょう。

2.2.3.1 新規ファイルの作成

ファイル名：python1-1.py として保存します。



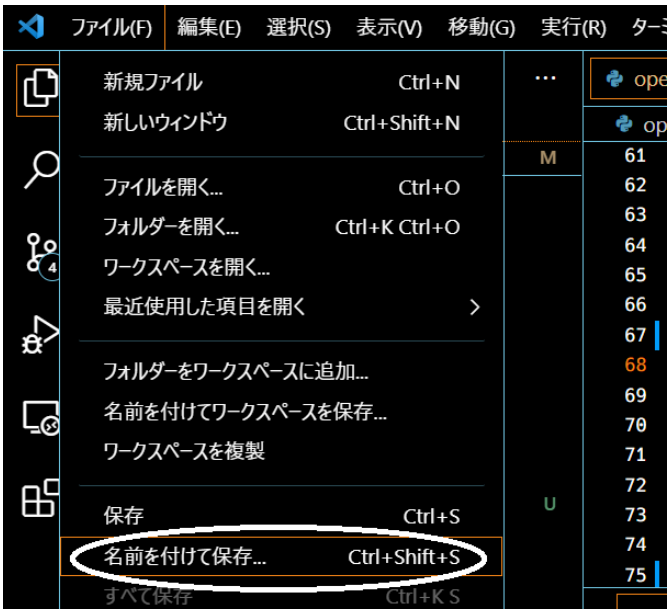
vscode の「ファイル」メニュー「新規ファイル」を選択します。

エディターで次のプログラムを入力します。行番番号は自動で附番されます（プログラムの実行に影響はありません）ので、参考にしてください。

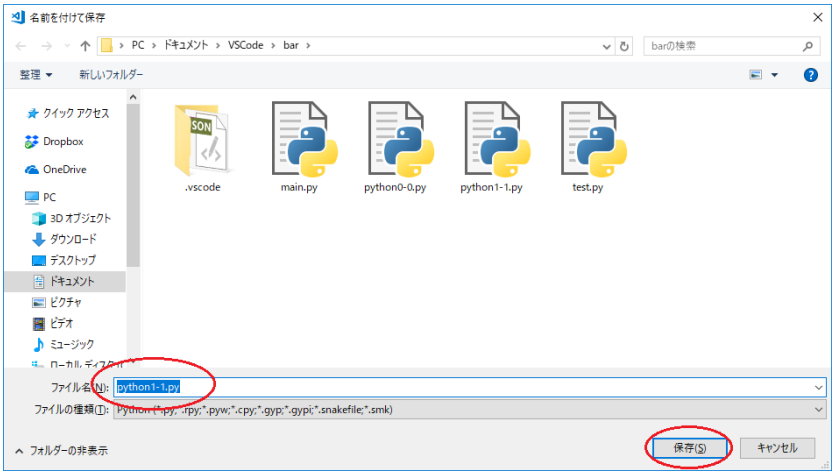
2.2.3: 保存プログラム（python1-1.py）

```
x = 1
x = x + 2
x
```

入力が完了しましたら保存の操作を行います。



vscode の「ファイル」メニュー「名前を付けて保存...」を選択します。



ファイル名「python1-1.py」と入力して保存を選択します。

保存が完了しましたら、**対話実行**で利用したターミナルに切り替えます。

Point

対話実行で動作している場合はターミナルで「>>>　」の後に、CTRL+Z を押すか「exit()」を実行することでコマンド実行状態に変更できます。

2.2.4: 出力なしの実行

```
1   c:\\~> python python1-1.py
2   c:\\~>
```

※ 計算はされていますが、表示されません。

それでは出力を行うためにもう一度ファイルを作成し print 文を追加してみましょう。

※　ファイル作成の手順は新規ファイルの作成を参照してください。

ファイル名：python1-2.py

2.2.5:　保存プログラム（python1-2.py）

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

実行方法

2.2.6:　出力ありの実行

```
1   c:\\〜> python python1-2.py
2   3
```

※ 計算結果が表示されました。

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。

※ **対話実行**で変数の値を出力するために変数名だけでよかった理由は式の評価した値が表示されたためです。

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　：　Plactice 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。

ファイル名「plactice1.py」として保存しましょう。

2.3　変数と式の評価

もう一度**対話実行**で動作確認しています。

2.3.1:　式

```
1   c:\\〜> py
2   >>> x = 1
```

```
3   >>> x
4   1
```

※ ここまでは復習です。

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）

2.3.2: 式の評価

```
5   >>> x < 10
6   True
7   >>> x > 10
8   False
```

※ 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もの式の書き方

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。

2.3.3:　式の評価（条件式１）

```
>>> x <= 1
True
>>> x >= 2
False
>>> x == 1
True
>>> x == 2
False
>>>  x != 1
False
```

当たり前ですが、こんな式の評価もできます

2.3.4:　式の評価（条件式２）

```
>>> 60 < 70
True
>>> 60 > 70
False
```

ほかの言語では書けないかもしれませんが次のような式の条件式も可能です。

2.3.5: 式の評価（条件式３）

```
1   >>> x = 5
2   >>> 1 < x < 10
3   True
4   >>> 1 < x < 5
5   False
```

※ ほかの言語では「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。

2.3.6: 式の評価（条件式４）

```
1   >>> a = "z"
2   >>> b = "z"
3   >>> a == b
4   True
5   >>> c = "y"
6   >>> a == c
7   False
8   >>> a > c # "z" > "y" を辞書順で評価します。
9   True
10  >>> a < c
11  False
12  >>> a = "abc"
13  >>> b = "abc"
14  >>> c = "azc"
15  >>> a == b
16  True
17  >>> a == c
18  False
19  >>> a > c # "abc"と"azc"の辞書順の評価
20  False
21  >>> a < c
22  True
```

Point
対話実行で表示されている「>>>」の後に入力する命令や式は実行され「評価」されます。

2.3.7: 変数の評価

```
1  >>> x = 1
2  ※ x に 1 を代入するという命令が実行された。
3  >>> x
4  1
5  ※ x を評価した結果が表示された。
6  >>> print ( x )
7  1
8  ※ x を表示する命令が実行された。
```

このように **対話実行**では条件式や計算式の評価結果を表示しますので、print 命令による出力と違いがわかりません。プログラムをファイルに保存して実行する場合は条件式や計算式の評価結果は出力されませんので、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）

評価について理解できてきたと思います。評価は場合分けで利用するために行われます。

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。

2.3.8: 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「プロック」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。

※　ここでは **対話実行**で場合分けの処理を実行してみます。

コード　2.3: if-ex.py

```
1   >>> x = 60
2   >>> if x >= 50:
3   ...       print ( " goukaku " ) # 左側のインデント空白は
4                                     「Tab」キー（１回）で入力します。
5   ... else:
6   ...       print ( " fugoukaku " )
7   ...   #インデントなしで「Enter」キーを押すと式を評価して結果を
8         表示します。
9   goukaku
10  >>>
```

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

新規ファイルを作成して「凡例 if-1」を保存して実行してみましょう
ファイル名「plactice2.py」として保存しましょう。

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。

2.3.9: 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つ目の条件が真の場合に実行あるいはすべての条件に当てはまらない時の実行を書くことができます。

2.3.10: 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。ファイル名：python2-1.py として保存し実行しましょう。

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。
ファイル名：python2-2.py として保存し実行しましょう。

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。ファイル名「plactice3.py」として保存しましょう。

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。

2.3.11:　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

対話実行で「and or not」条件の評価を確認しましょう。

2.3.12: and or not の式の評価

```
>>> x = 999
>>> x > 100
True
>>> not x > 100
False
>>> x >= 0 and x <= 100
False
>>> x < 0 or x > 100
True

```

※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。
ファイル名「plactice4.py」として保存しましょう。

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後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。
対話実行してキーボードからの入力をしてみましょう。

2.4.1: キーボードからの入力

```
>>> input()
abc #  キーボードから入力
 'abc'
>>>
>>> input( "type = ”  )
type = xyz
 'xyz'
>>>
```

※　入力した文字が評価されて表示されました。
入力した文字を変数 s にいれてみましょう。

2.4.2: キーボードから変数への代入

```
>>> s = input( "type = " )
type = xyz
>>> s
 'xyz'
>>>
```

※変数 s に文字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る扱いになります。ここでは文字列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

```
>>> s * 2
 'xyzxyz'
※　同じ文字列が 2 倍になりました
>>> s + 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
※　文字列と数値では計算できませんのでエラーになります。
>>> s + "10”
```

```
23  xyz10
24  ※  文字列と文字列はプラスで連結されることがわかります。
```

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）

```
25  >>> s = input ( "number = " )
26  10
27  >>> s
28   '10'
29  >>> s * 2
30   '1010'
31  ※  このままだと変数 s は文字列の’10’が入っています。
```

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）

```
32  >>> int(s)
33  10
34  ※  シングルクォーテーションがなくなりました。
35  >>> n = int(s)
36  >>> n * 2
37  20
38  ※  数値として評価されています。
```

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( n * 2 )
'20'
>>> str( n * 2 ) + " is twenty"
'20 is twenty'
```

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)
ファイル名：python4-1.py として保存し実行しましょう。
※　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

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

実行結果

```
math = 50
english = 80
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。
python4-1.py をファイル名「plactice5.py」として別名で保存しましょう。

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。
ファイル名を python5-1.py として保存しましょう。

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。
python5-1.py をファイル名「plactice6.py」として別名で保存しましょう。

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しする命令です。

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。
ファイル名：python5-2.py

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)
```

対話実行してみましょう。

コード　2.9: 対話実行

```
>>> for x in range(1, 5):
...     print( x )
...
1
2
3
4
          ・
          (中略)
          ・
>>> for x in range(2, 10, 3):
...     print( x )
...
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。
ファイル名を python5-3.py として保存してください。

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　：　Practice 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。
python5-3.py をファイル名「practice7.py」として別名で保存しましょう。

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。

ファイル名：python5-4.py

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を意識して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　：　Practice 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。

python5-4.py をファイル名「practice8.py」として別名で保存しましょう。

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。
ファイル名：python5-5.py

while を利用したやり方

```
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。
ファイル名：python5-6.py

while を利用したやり方

```
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修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。
ファイル名：python6-1.py

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(5):
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。
python6-3.py をファイル名「plactice9.py」として別名で保存しましょう。

実行例

```
c:\\〜> python practice9.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。ファイル名を python6-4.py として保存してください。

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
```

この判例の 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。
ファイル名を python6-5.py として保存してください。

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　：　Practice 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。
ファイル名「practice10.py」として保存しましょう。

	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 : Plactice 11

「Plactice9」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。
practice9.py をファイル名「practice11.py」として別名で保存しましょう。

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。

対話実行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]
#並べ替え（昇順）
>>> l.sort()
>>> l
[1, 2, 4, 5, 7, 9]
#並べ替え（降順）は（）の中に「reverse = True」を記述する。
>>> l.sort(reverse=True)
>>> l
[9, 7, 5, 4, 2, 1]
>>>
#リストの最後の要素を取り出す
>>> l.pop()
1
>>> l
[9, 7, 5, 4, 2]
#リストの最初の要素を取り出す。
>>> l.pop(0)
9
>>> l
[7, 6, 5, 2]
#指定の値を削除する
>>> l.remove(4)
>>> l
[7, 5, 2]
#リストを空にする
>>> l.clear()
>>> l
[]
```

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

```
辞書名 = {’キー値１’:’値１’, ’キー値２’:’値２’, "キー値３":"値３"}
※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

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

実行結果

```
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

```
1    AKJ ⇒旭川空港
2    MMB ⇒女満別空港
3    CTS ⇒新千歳空港
4    HKD ⇒函館空港
5    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。今まで利用してきた 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。
ファイル名を python9-1.py として保存してください。

python8-1.py

```
1  def func(x):
2      answer = x ** 2 + 2 * 2 + 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
   という意味ことになります。
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。

対話実行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。

```
>>> y = func(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'func' is not defined
```

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。
ファイル名を python9-2.py として保存してください。

python8-2.py

```
1   def func1(x):
2       answer = x ** 2 + 2 * 2 + 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。
ファイル名「plactice12.py」として保存しましょう。

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

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

今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。**対話実行**で確認してみましょう。

2.8.2: 変数のスコープ

```
1   >>> #関数の定義
2   >>> def func(x):
3   ...     answer = x ** 2 + 2 * x + 3
4   ...     return answer
5   ...
6   >>> answer
7   Traceback (most recent call last):
8     File "<stdin>", line 1, in <module>
9   NameError: name 'answer' is not defined
10  # answer が定義されていない（スコープ外）というエラーが表示されます。
11  >>> x
12  Traceback (most recent call last):
13    File "<stdin>", line 1, in <module>
14  NameError: name 'x' is not defined
15  # x も同様にスコープ外であることがわかります。
```

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。
ファイル名 python9-3.py で保存してください。

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
11  x = 4
12  y = func()
13  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。
ファイル名 python9-4.py で保存してください。

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。ファイル名 python9-5.py として保存してください。

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。
ファイル名「plactice13.py」として保存しましょう。

	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 practice13.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（モジュールの利用）

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 モジュールと乱数を扱う random モジュールをここで紹介します。

他にもたくさんのモジュール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：func-3

10 回繰り返しを for 文で行い、モジュールを使って現在時刻、カウンター i の平方根 $\sqrt{\ i\ }$ や 3 乗根 $\sqrt[3]{\ i\ }$ 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。
ファイル名 python9-1.py として保存してください。

python9-1.py

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

第 3 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。

3.1 クラス入門

3.1.1 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。

3.1.2　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　3.1: 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。

3.1.3 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　3.2: 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
```

3.1.4 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　3.3: 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。

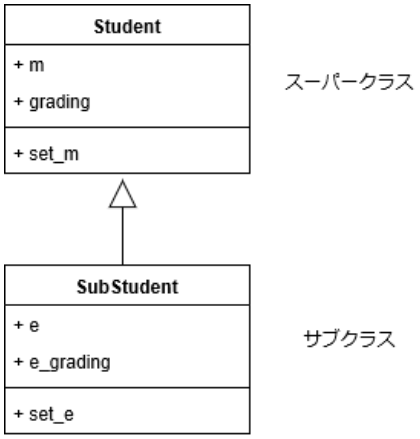
Point
多くのプログラミングではクラスの形（定義）とそのクラスを利用する方法が一緒に書かれているため、クラスとして書かれたプログラムの内容も理解しておく必要があると思われがちですが、いままで見てきた配列（リスト）のように.append や pop など使い方だけ知っていれば良いと考えておいてください。

3.1.5 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　3.4: 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 4 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

```
1  #グローバル変数（配列）の準備
2  subjects = ["Mathematics" ,"English","Social","Japanese","Scientific"]
3  scores = []
4  #関数 input_data の記述
5  def input_data():
6      for n in range(3):
7          cnt = 0
8          scores.append([])
9          name = input("Name = ")
10         scores[n].append(name)
11         for s in subjects:
12             x = input( s + " = " )
13             y = int ( x )
14             if y > 100 or y < 0 :
15                 y = -1
16                 print ( "error score" )
17             else:
18                 cnt = cnt + 1
19             scores[n].append( y )
20         scores[n].append(cnt)
21 #関数 get_average の記述
22 def get_average(m):
23     sum = 0
24     for n in range(1,6):
25         if scores[m][n] != -1:
26             sum = sum + scores[m][n]
27     return sum / scores[m][6]
28
29 #プログラムはここから開始
30 input_data()
31 for m in range(3):
32     ave = get_average(m)
33     print(scores[m][0],"の",scores[m][6],"科目の平均点",ave)
```