プログラミング入門

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どうさせるための手順という考え方に基づいて、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。

プログラミングでできること。

excelマクロを見せて処理自動化のメリットを見てもらう

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！

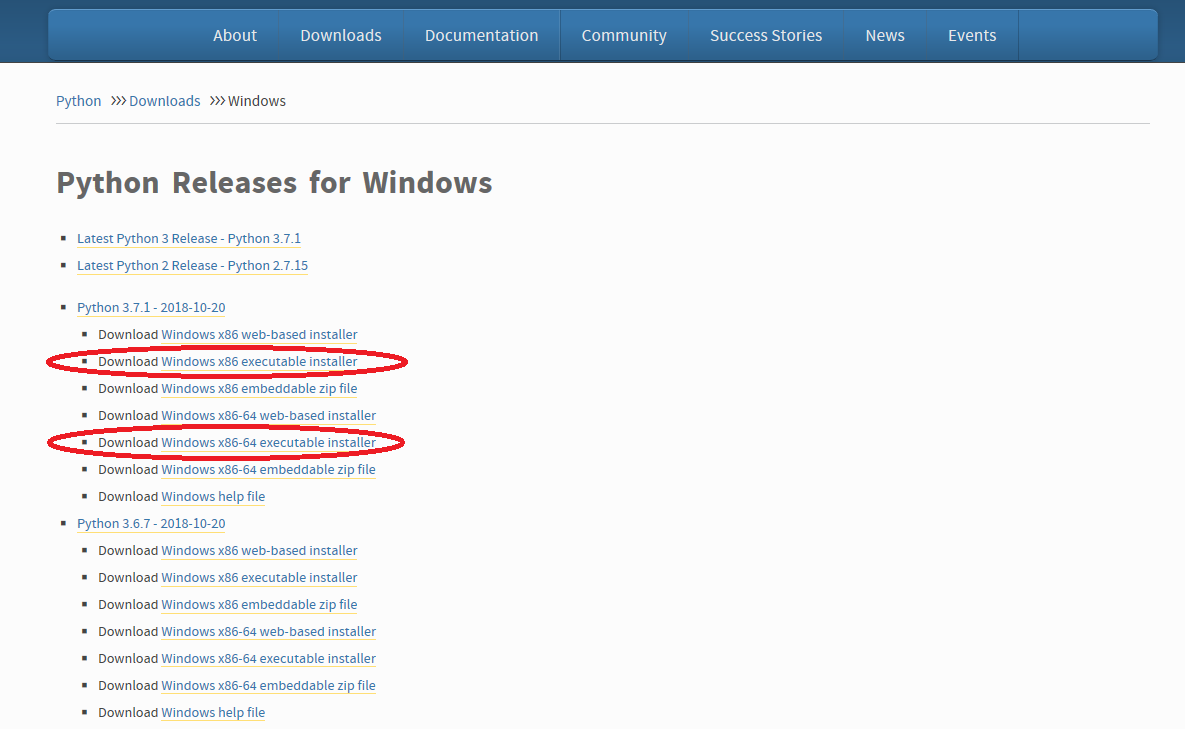
1.pythonインストール

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を参考にしてください。

１．次のURLを参照してください。

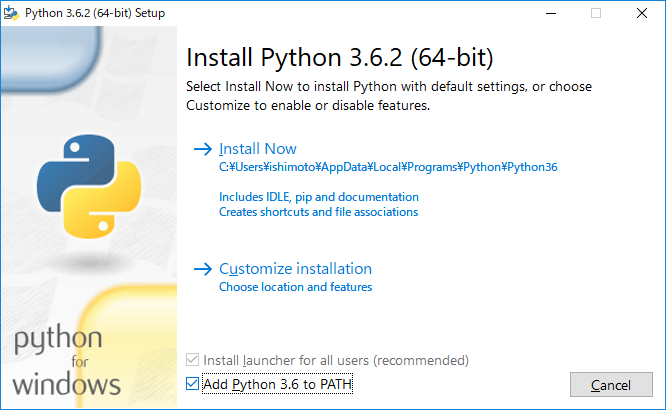
<https://www.python.jp/install/windows/install_py3.html>

[](https://www.python.jp/install/windows/install_py3.html)

・Download Windows x86 executable installer (32bit版)

・Download Windows x86-64 executable installer（64bit版）

いずれかを選択しダウンロードしますが、通常は32bit版であれば汎用性が高いです。



※画面は64bit版の3.6.2です。

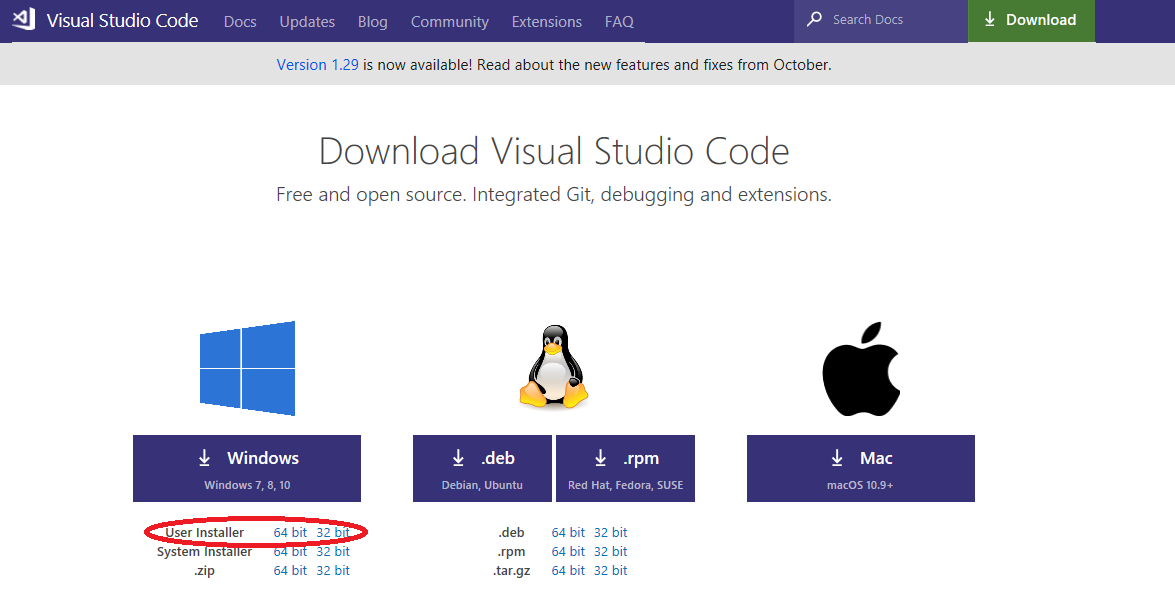
１．「Add Python 3.x to PATH」 をチェックしてください。

２．Install now をクリックしてインストールをしてください。

環境作成vscode

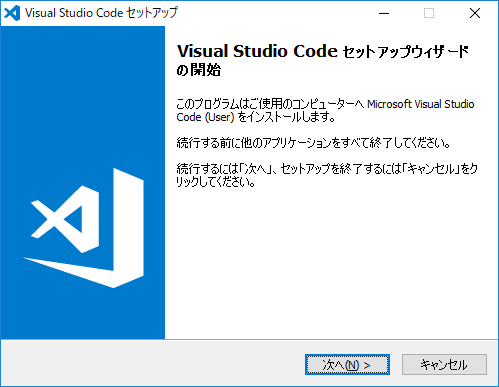
次のURLを参照してください。

<https://code.visualstudio.com/download>

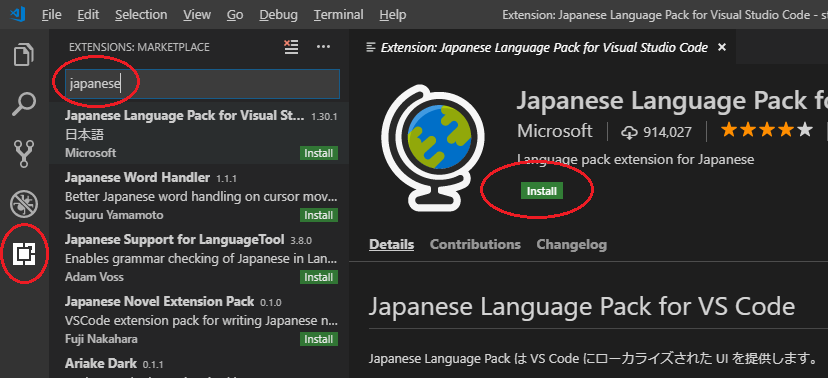


User Installerでそれぞれ32bit,64bitのいずれかを選択します。

あとはインストールするだけです。

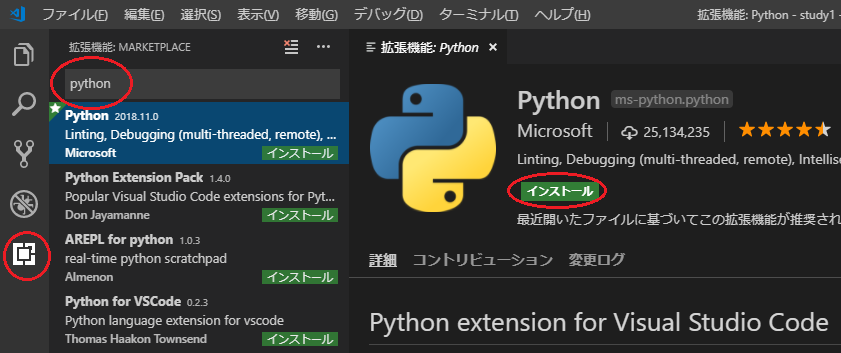


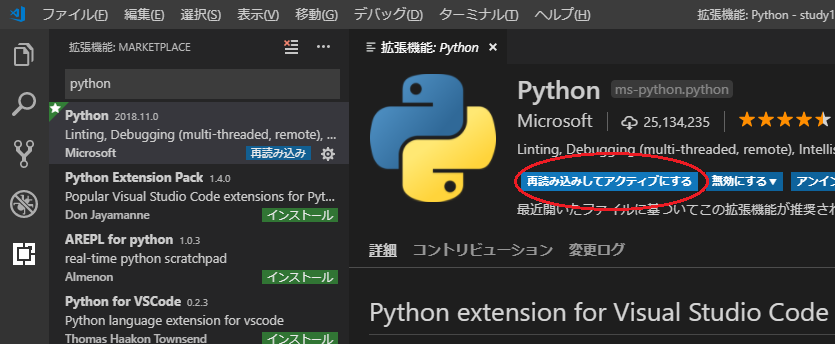
セットアップ後の日本語拡張機能のインストールを行います。



ここでvscodeを再起動します。

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。





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。

hint!

今回の講座の目的はプログラミングとはどんなものなのかを理解することを優先していますので、効率よくとかもっと速度をとか便利な機能をもっと知りたいというかたはこの講義の後に書籍やインターネットで調べてみるとよいでしょう。

１．プログラミング 最初の一歩

コンピューターへの命令の実行

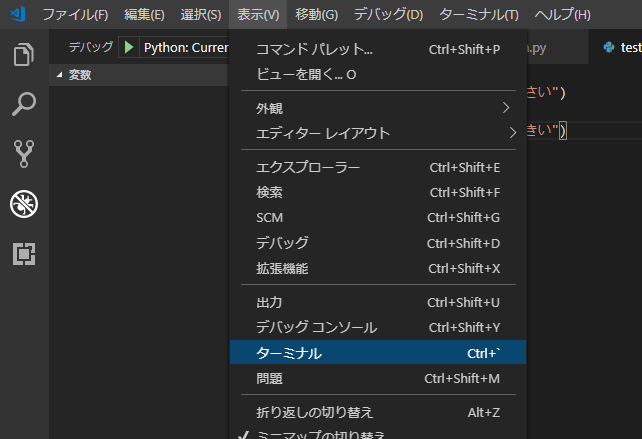
一つ一つ結果を確認して実行する方法と手順をたくさんまとめて実行する方法があります。

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。

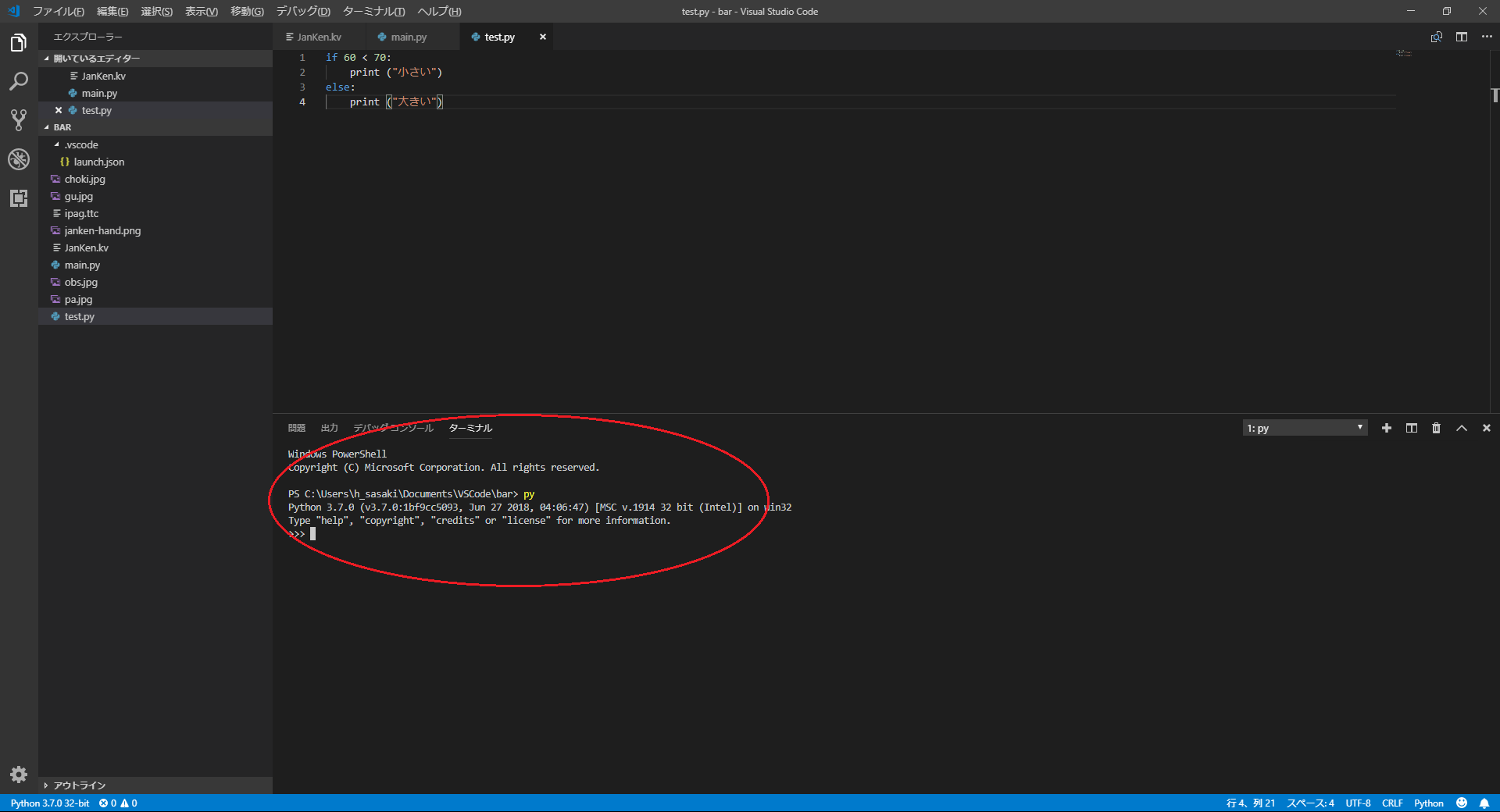
早速計算をさせてみましょう。

まずは対話型の命令実行を行ってみましょう。

pythonをインタラクティブ（対話型で）実行するためにvscodeで「ターミナル」を起動して、コマンドライン（プロンプト）上で対話型のコマンドpyを起動しましょう。



ターミナルの起動は「表示」メニューの「ターミナル」を選択します。



Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\～省略～> py

Python 3.7.～省略～ on win32

Type "help", ～省略～

>>> ←この画面で対話型のコンピューターへの命令を入力できます。

対話型では命令や計算の結果をすぐに確認することができます。

最初にコンピューターらしく計算機として利用してみましょう。

>>> 1 + 2

3

>>> 3 \* 4

12

>>> 1 / 3

0.3333333333333333

>>> 1 + 1 / 4

1.25

>>> 4 \*\* 2

16

>>> ( 1 + 2 + 3 ) / 2

3

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。

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多なうことが多いです。

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。

>>> x = 10

>>> x

10

>>> x / 4

2.5

>>> y = x - 5

>>> y

5

最初の行では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。）

変数として利用可能な名前付けルール

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。

先程の代入は数学でいうイコールと意味が違い等式ではなく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。

>>> 10 = x

File "<stdin>", line 1

SyntaxError: can't assign to literal

※ SyntaxError（エラー）になります。

ただし、等式ではないので右辺の計算結果を左辺に代入ということで右辺と左辺に同じ変数を書くことができます。

>>> x = x - 1

>>> x

9

xに10が代入されている状況であれば右辺の「x-1」の計算結果すなわち9が左辺に代入されることになりますので、結果xの値は9となります。

=は右辺の式の値を左辺に代入という意味になることが分かると思います。

コンピューターとしての能力を発揮させると電卓では簡単にできないこんな計算もできます。

「\*\*」アスタリスク２つならべるとべき乗を意味します。

>>> 2 \*\* 32

4294967296

※2の32条（2を32回乗算する）

>>> 2 \*\* 64

18446744073709551616

>>> 2 \*\* 128

（実行結果を確認してください）

※ 340澗2823溝6692穰0938𥝱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）

※ ！？（無量大数を超えます）

その他の計算

平方根（ルート）はまた後で。。。

ここまでであればExcelでできそうですね。

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）

まとまった手順とは？

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。

>>> x = 1

>>> x = x + 2

>>> x = x + 3

>>> x = x + 4

>>> x = x + 5

>>> x

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。

今後プログラム実行中に「※」のマークで始まる行は実行結果ではなく「注釈」として記載しているものです。

ここではプログラムで画面に出力する方法を学びます。

いままでも対話型の命令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。

>>> print(x)

15

>>> x

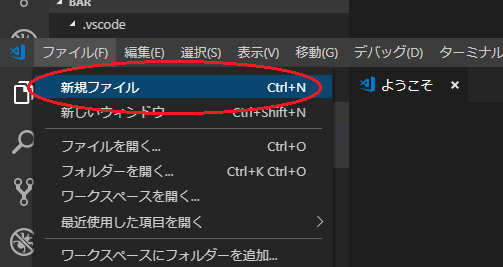
15

違いはなんでしょうか？

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みます。

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。



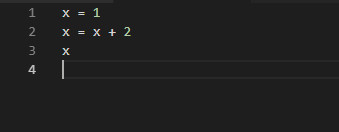
vscodeの「ファイル」メニュー「新規ファイル」を選択します。

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。

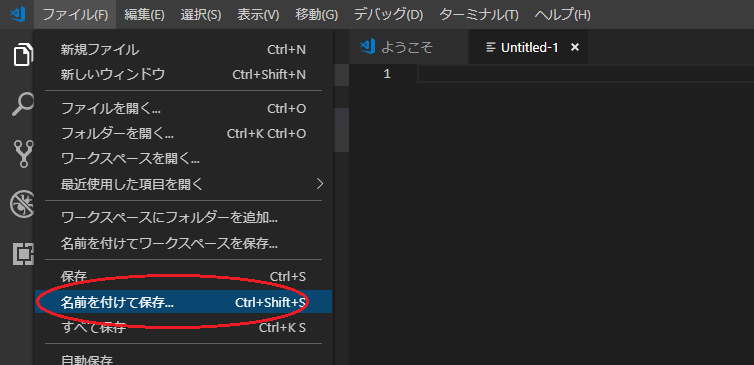
x = 1

x = x + 2

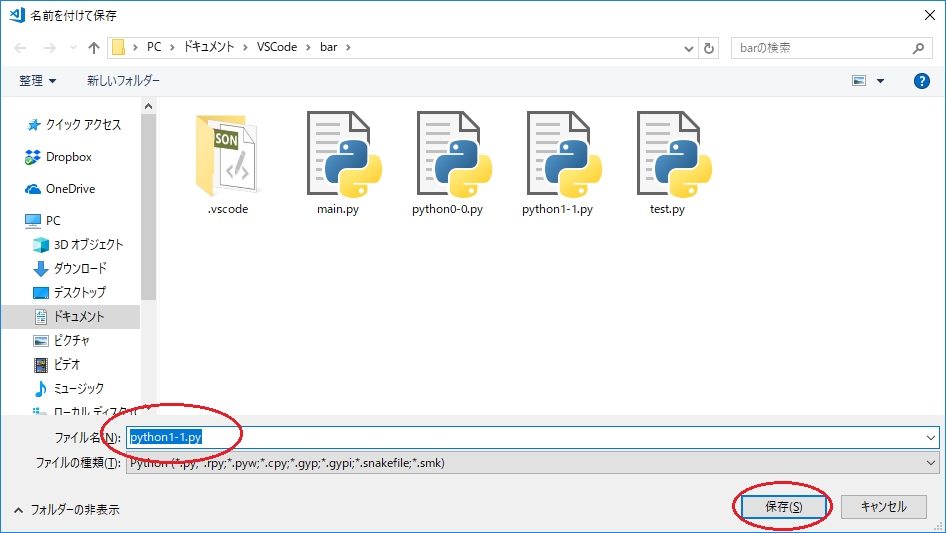
x



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。

実行方法

PS C:\～省略～> python python1-1.py

PS C:\～省略～>

※ 計算はされていますが、表示されません。

ファイル名：python1-2.py

x = 1

x = x + 2

x = x + 3

print (x)

実行方法

PS C:\～省略～> python python1-2.py

6

※ 計算結果が表示されました。

PS C:\～省略～>

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。

※ 対話型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。

式評価についてはもう少し先に進めながら確認します。

plactice0-0

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。

もう一度対話型で動作確認しています。

PS C:\～省略～> py

>>> x = 1

>>> x

1

※ ここまでは復習です。

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）

>>> x < 10

True

>>> x > 10

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。

条件式の書き方１

条件式で利用される主なもの式の書き方

== “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”

>>> 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

>>> 60 < 70

True

>>> 60 > 70

False

ほかの言語では見か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。

>>> x = 5

>>> 1 < x < 10

True

>>> 1 < x < 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。

Point

対話型の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。

>>> x = 1

※ xに1を代入するという命令が実行されました。

>>> x

1

※ xを評価した結果が表示されました。

>>> print ( x )

1

※ xを表示する命令が実行されました。

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

条件が成立したときに行う手続き、不成立の時に行う手続き

if文の書き方１

if 条件式:

真の手続き１

真の手続き２

・

・

else:

偽の手続き１

偽の手続き２

・

・

※ 「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）することでまとまった手順にすることができます。

ここでまとまった手続きを「ブロック」といいます。

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対話型で作りましょう。

※　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。）

>>> x = 60

>>> if x >= 50:

... print ( " goukaku " )

... else:

... print ( " fugoukaku " )

...

goukaku

>>>

plactice1

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：python2-1.py

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

・

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

・

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：python2-2.pyとして保存し実行しましょう。

x = 60

if x >= 80:

print ( “A” )

elif x >= 60:

print ( “B” )

elif x >= 40:

print ( “C” )

else:

print ( “D” )

保存し実行しましょう。

hint

条件式の書き方２

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

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

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

point

対話型の実行で「and or not」条件の評価を確認しましょう。

>>> 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ると思います。

plactice2

「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：python2-2.py（ファイル名は変更しません）

キーボードからの入力

対話型で命令を実行してキーボードからの入力をしてみましょう。

>>> 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。

>>> 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しましょう。

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。

>>> 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”

xyz10

※　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。

>>> s = input ( “number = “ )

10

>>> s

‘10’

>>> s \* 2

‘1010’

※　このままだと変数sは文字列の’10’が入っています。

そこで文字列を数値にするint命令を利用します。

>>> int(s)

10

※　シングルクォーテーションがなくなりました。

>>> n = int(s)

>>> n \* 2s

20

※　数値として評価されています。

数値として判定できない文字の扱いはまた後で。。。

数値を文字列にするにはstr命令を利用します。

>>> str( n \* 2 )

‘20’

>>> str( n \* 2 ) + “ is twenty”

‘20 is twenty’

文字列にすると連結もできます。

凡例input-1

学生Aさんの数学の得点をxに、外国語の得点を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fを使います）

ファイル名：python3-1.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。

x = input ( “math = “ )

m = int ( x )

y = input ( “english = “ )

f = int ( y )

print ( (m + f) / 2 )

plactice3

「凡例input-1」を社会の得点をzに入力「”social = “」し、変数sに数値化して平均点を出力しましょう。

ファイル名：python3-1.py（ファイル名は変更しません）

凡例input-2

学生Aさんの数学の得点と、外国語の得点を入力し数値化して平均点を出力しましょう。

実行例

math = 60

english = 70

The average score is 65 ⇐　この行が出力結果です。

ファイル名：python3-2.pyとして保存し実行しましょう。

x = input ( “math = “ )

m = int ( x )

y = input ( “english = “ )

f = int ( y )

a = (m + f) / 2

print ( “The average score is “ + str( a ) )

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。

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。

先ほどの「凡例input-1」を5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。

x = input ( “subject1 = “ )

y = int ( x )

x = input ( “subject2 = “ )

y = y + int ( x )

x = input ( “subject3 = “ )

y = y + int ( x )

x = input ( “subject4 = “ )

y = y + int ( x )

※　そろそろ飽きてきます。

x = input ( “subject5 = “ )

y = y + int ( x )

print ( y / 5 )

※　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は、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：python4-1.py

for x in range(3):

print( “wan” )

print( “nyan” )

実行結果

wan

nyan

wan

nyan

wan

nyan

for文はブロックをrangeの中に書かれている回数繰り返しすることが分かったと思います。

plactice4

同じく1000回”wan”と表示させましょう。

ファイル名：python4-1.py（ファイル名は変更しません）

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：python4-2.py

for x in range(3):

print( x )

実行結果

0

1

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(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

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(1, 10, 3)

対話型で実行してみましょう。

>>> for x in range(1, 5):

... print( x )

...

1

2

3

4

・

・

>>> for x in range(1, 10, 3):

... print( x )

...

1

4

7

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(1, 10, 3) 1,4,7

括弧の中で指定している最初の値から次の値-1まで最後の値ずつ増えた値が取得されます。

凡例for-3

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：python4-2.py

sum = 0

for x in range(1,11):

sum = sum + x

print ( sum )

plactice5

１．「凡例for-3」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-3」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。

ファイル名：python4-2.py（ファイル名は変更しません）

凡例for-4

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」を効率よくできないか考えてみます。

x = input ( “subject1 = “ )

y = int ( x )

x = input ( “subject2 = “ )

y = y + int ( x )

x = input ( “subject3 = “ )

y = y + int ( x )

x = input ( “subject4 = “ )

y = y + int ( x )

※　同じ記述ばかりなのでコンピューターやらせることにしたいと思います。

x = input ( “subject5 = “ )

y = y + int ( x )

print ( y / 5 )

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科目目も書き換えてみましょう。

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

print ( y / 5 )

ファイル名：python4-3.py

y = 0

for n in range(1,6):

x = input ( “subjectN = “ )

y = y + int( x )

print( y / 5 )

※　こんなに短くできました。

plactice6

「凡例for-4」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。

ファイル名：python4-3.py（ファイル名は変更しません）

配列について

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。

m = “Mathematics”

print ( m )

e = “English”

print ( e )

so = “Social”

print ( so )

j = “Japanese”

print ( j )

sc = “Scientific”

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を格納していきます。

subjects = \

[“Mathematics”,”English”,”Social”,”Japanese”,”Scientific”]

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：python5-1.py

subjects = \

[“Mathematics”,”English”,”Social”,”Japanese”,”Scientific”]

for s in subjects:

print s

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。

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。

ファイル名：python5-1.py（ファイル名は変更しないでください）

subjects = \

[“Mathematics”,”English”,”Social”,”Japanese”,”Scientific”]

scores = []

for s in subjects:

x = input( s + “ = “ )

y = int ( x )

scores.append( y )

for n in range(0,5):

print ( subjects[n] + “:” , scores[n])

# 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）

この配列（リスト）はイメージとして次のようになります。

| 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。

for n in range(0,5):

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。

Mathematics: 10

English: 20

Social: 30

Japanese: 40

Scientific: 50

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。

ファイル名：python5-1.py（ファイル名は変更しないでください。）

subjects = \

[“Mathematics”,”English”,”Social”,”Japanese”,”Scientific”]

scores = []

for s in subjects:

x = input( s + “ = “ )

y = int ( x )

scores.append( y )

for n in range(0,5):

if scores[n] >= 60:

mes = “合格”

else:

mes = “不合格”

print ( subjects[n] + “:” , scores[n], mes)

plactice7

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「エラー」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「優」とし、60点以上を「良」とし、40点以上を「可」、40点未満を「不可」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「判定不能」と出力します。

ファイル名：python5-2.py

[実行例]

PS C:\～省略～> phthon python5-2.py

Mathematics = 50

English = 60

Social = 80

Japanese = 39

Scientific = 101

エラー

Mathematics: 50 可

English: 60 良

Social: 80　優

Japanese: 39 不可

Scientific: -1 判定不能

二次元配列の考え方

凡例list-4

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君の数学（60）と英語（80）

B君の数学（70）と英語（90）

C君の数学（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。

students\_scores = [[60, 80], [70, 90], [80, 100]]

D君のデータ（数学　50点,　英語　70点）が増えたときは次のようにします。

students\_scores.append([50, 70])

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

A君の３科目目（社会科　70点）を追加するときは次のようにします。

students\_scores[0].append(70)

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

この時C君の英語の点数（100点）は次のように参照します。

students\_scores[2][1]

plactice8

「plactice7」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「判定不能」な点数の入力はないものとします。

ファイル名：python5-3.py

[実行例]

PS C:\～省略～> python python5-3.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

関数による処理の細分化

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。

def 関数名():

・

まとまった処理

・

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（関数を評価する）を取得するためにreturn文を使います。

凡例func-1

y = x^2 + 2 \* x + 3の２次方程式の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。

def func(x):

answer = x \*\* 2 + 2 \* 2 + 3

return answer

y = func(2)

print( y )

y = func(4)

print( y )

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。

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。

>>> #関数の定義

>>> def func(x):

... answer = x \*\* 2 + 2 \* x + 3

... return answer

...

>>> answer

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'answer' is not defined

# 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。

>>> x

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'x' is not defined

# xも同様にスコープ外であることがわかります。

凡例scope-1

凡例func-1の変数xをグローバルで利用し、x^2 + 2 \* x + 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。

#引数にはなにも指定しません。

def func():

answer = x \*\* 2 + 2 \* x + 3

return answer

x = 2

#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。

y = func()

print( y )

x = 4

y = func()

print( y )

凡例scope-2

#スコープが異なれば同じ変数名が利用できます。

def func(x):

answer = x \*\* 2 + 2 \* x + 3

x = -1

#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。

return answer

x = 2

#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。

y = func( x )

print( y )

print( x )

x = 4

y = func( x )

print( y )

凡例func-3

テストでは80点以上を「優」とし、60点以上を「良」とし、40点以上を「可」、40点未満を「不可」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「判定不能」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。

※　plactice7を関数evaluationを利用して記載することができます。

def evaluation(score)

if score < 0 or score > 100:

mes = "判定不能"

elif score >= 80:

mes = "優"

elif score >= 60:

mes = "良"

elif score >= 40:

mes = "可"

else:

mes = "不可"

return mes

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

scores = []

for s in subjects:

x = input( s + " = " )

y = int ( x )

scores.append( y )

for n in range(0,5):

mes = evaluation(scores[n])

print ( subjects[n] + ":" , scores[n], mes )

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。

配列要素に対する操作機能の一部

.sort()

配列を並び替え（昇順）を行います。もしも逆順にしたいのであれば（）の中に「reverse = True」を記載します。

del リスト名[対象範囲]

.remove()

（）の中に削除したい要素を指定します。

.clear()

リストを空にします。

.pop()

規定値で最後の要素を取り出し、リストから削除します。

指定の位置のデータを削除するためには（）の中に場所をしていします。

それぞれの捜査結果を対話型で実行して確認してみましょう。

exarcise1

解答編

plactice1（python2-1.py）

省略

plactice2（python2-2.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” )

plactice3（python3-1.py）

x = input ( “math = “ )

m = int ( x )

y = input ( “english = “ )

f = int ( y )

z = input ( “social = “ )

s = int ( z )

print ( (m + f + z) / 3 )

plactice4（python4-1.py）

for x in range(1000):

print( “wan” )

print( “nyan” )

plactice5（python4-2.py）

sum = 0

for x in range(1,1000):

sum = sum + x

print ( sum )

sum = 0

for x in range(1, 11):

sum = sum + 1 / 2 \*\* x

print ( sum )

plactice6（python4-3.py）

y = 0

for n in range(1,6):

x = input ( “subject” + str(n) + “ = “ )

y = y + int( x )

print( y / 5 )

plactice7（python5-2.py）

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

scores = []

for s in subjects:

x = input( s + " = " )

y = int ( x )

if y > 100 or y < 0 :

y = -1

print ( "エラー" )

scores.append( y )

for n in range(0,5):

if scores[n] == -1:

mes = "判定不能"

elif scores[n] >= 80:

mes = "優"

elif scores[n] >= 60:

mes = "良"

elif scores[n] >= 40:

mes = "可"

else:

mes = "不可"

print ( subjects[n] + ":" , scores[n], mes )

plactice8（python5-3.py）

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

scores = []

for n in range(3):

print(n + 1, "人目の入力")

scores.append([])

for s in subjects:

x = input( s + " = " )

y = int ( x )

if y > 100 or y < 0 :

y = -1

print ( "エラー" )

scores[n].append( y )

for m in range(5):

sum = 0

for n in range(3):

sum = sum + scores[n][m]

print(subjects[m] , "の平均点", sum / 3)

Exarcise1

python便利な機能

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まとめながら記述する方法で記載されます。

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。

これがオブジェクト指向プログラミングといいます。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優先して考えていくことを優先しています。

この中で重要なキーワードがクラスといわれるプログラムで扱うデータと処理（メソッド）を一体化した仕組みです。

凡例class-1

ここで学生個人のデータを管理することを考えましょう。

学生は学生番号と履修科目と得点を管理しています。履修科目には必須科目（数学、英語、国語）とコース別のAcourse（絵画、音楽）、Bcourse（会計、情報）の選択必須科目があり、どちらかを選択しなければならないことを考えます。

また、学生の管理するデータを入力することと、学生ごとの合計点を計算する機能が必要だとします。

まずは必須科目を管理する機能と必須科目だけを５人分入力し、それぞれの合計点を出力するプログラムを作成します。

これをクラスとしてまとめるため、共通科目だけを扱う機能を次のように記載します。

class Student: #共通科目を取り扱うための機能をひとまとまりにしたクラス

std\_subjects = ["Mathematics" ,"English","Japanese"]

def input\_number(self):

self.number = input( "number = " )

def input\_scores(self):

self.scores = []

input\_data(self.std\_subjects,self.scores)

#データを入力するための手続きを関数としてまとめておきました。

#エラー処理およびテキスト入力された得点を数値に変換する手続き

def sum\_of\_scores(self):

self.summary = 0

for score in self.scores:

self.summary = self.summary + score

return self.summary

def input\_data(subjects,scores):

for s in subjects:

x = input( s + " = " )

y = int ( x )

if y > 100 or y < 0 :

y = -1

print ( "エラー" )

scores.append( y )

student\_list = []

#学生の追加

for n in range(5):

student\_list.append( Student() )

#学生の番号と得点の入力と合計点の出力

for std in student\_list:

std.input\_number()

std.input\_scores()

print(std.sum\_of\_scores())

凡例class-2

凡例class-1の共通機能を基にして、Acourse、Bcourseの選択必須科目の機能をを追加します。

Acourseの学生３名とBcourseの学生２名の得点を入力して（各コースの順番はばらばらでも可能）それぞれの学生の合計点とAcourseを選択している学生の平均点、Bcourseを選択している学生の平均点を計算するプログラムです。

class AcourseStudent(Student):

acourse\_subjects = ["picture","music"]

def input\_scores(self):

super().input\_scores()

input\_data(self.acourse\_subjects,self.scores)

class BcourseStudent(Student):

bcourse\_subjects = ["accounting","information"]

def input\_scores(self):

super().input\_scores()

input\_data(self.bcourse\_subjects,self.scores)

#凡例class-1の学生の追加部分を次のように書き換え

student\_list.append( AcourseStudent() )

student\_list.append( AcourseStudent() )

student\_list.append( BcourseStudent() )

student\_list.append( BcourseStudent() )

student\_list.append( AcourseStudent() )

#学生の番号と得点の入力と合計点の出力

for std in student\_list:

std.input\_number()

std.input\_scores()

for std in student\_list:

print("number = " , std.number , " summary = ",

std.sum\_of\_scores())

おまけpythonによるguiプログラミング（冬休みの宿題）