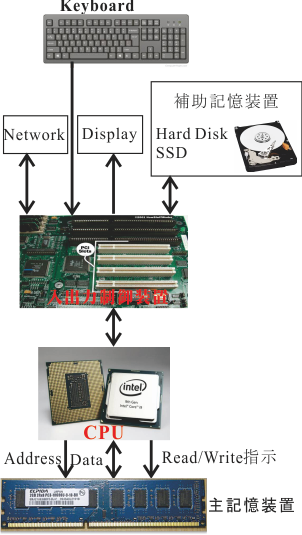
データ処理に必要なPythonの基本

医学部物理学教室

**ウエット研究が主流の従来型医学部から脱却して、生命科学と情報科学を身に着けた研究者の重要性が高まっている（金井医学部長）。**

A　コンピュータの仕組み

コンピュータは、cpu(central processing unit)、主記憶装置、補助記憶装置、入出力装置(input/output, i/o装置)から構成されている。プログラムやデータなど重要な情報は、補助記憶装置に記憶され、必要に応じで主記憶装置に転送してcpuが処理する。cpuは、主記憶装置に記憶されている機械語プログラム(0,1の集まり)を実行し、結果の大半を主記憶装置に記憶させる。cpuサイドからコンピュータを見ると、cpuはほとんどのデータのやり取りを主記憶装置との間で行い、必要に応じて、ディスプレー・キーボード・ネット・補助記憶装置などコンピュータ外部と接続されている入出力装置とデータのやり取りを行う。

主記憶装置はデジタルデータの記憶装置であり、cpuに直結して高速でデータのやり取りを行う。デジタルデータ量には単位があり、0か1（0ffかon）の状態をとる素子を1単位として扱い、これを1 bit(binary digit,ビット)という。また、これを8個(8bit)一組として1 byte(バイト)という。これには歴史的経緯があり、アルファベットや数字その他で100通りほど記号があり、文字通信には7bit(27=128)で十分であるが、2のべき乗(8=23)のほうが何かと便利なので8bitを単位としている。最近のコンピュータは64bit単位でデータを扱うようにできており、8bit単位での扱いはむしろ手間取る。しかし、AIにおける計算は8bitで十分なことが多く、最近のAIブームから、最新のcpuは8bit計算を格段に速くできるように作られている。

いずれにせよ、コンピュータは、1byteを単位として、右図のようにメモリーセルを並べ、各1byteのデータをアドレスで指定する。したがって、特定のメモリーセルを指定するときには、アドレスの数字を指定し、さらにb0-b7を指定する。右の図は4Gbytes(232bytes)のメモリーの例である。ほとんどのパソコンはこれより多くのメモリーセルを備えている。

主記憶装置に要求される性能は、高速であること（アドレスを指定してデータが取り出されるまでの時間）、高集積化ができること（記憶容量が大きい）、低消費電力であること、安価であること等である。これを全て満たすことを最優先した結果、メモリーセルとしてコンデンサを用いるDRAM(Dynamic Random Access Memory)が主流となった。コンデンサに電荷が蓄積していると1、そうでないと0として1bitの情報を記憶する。この結果、妥協した点がある。コンデンサと言ってもすぐに電荷を失うような粗悪コンデンサなので、1秒間に500回復習して忘れないようにしている。また、電源を切ると全て忘れるため、パソコンで文書を作成しているとき、補助記憶装置に保存せずに電源を切るとデータが消失する。以下で述べるPythonはプログラミングが容易であることに重点を置いているため、メモリーセルの配置（8bitを単位にしてアドレスで指定する）を知らなくてもよいが、処理速度やメモリーの有効利用を優先している言語（ｃ,c++言語など）では常に意識する必要がある。

**B　補助記憶装置とファイル**

補助記憶装置は、人間が作成したデータを記憶しておく部分であり、コンピュータを使用する観点で最も重要である。現在はハードディスクと呼ばれる磁気ディスクを用いる。ハードディスクはwindows上では「ｃ：」,「ｄ：」,「e：」 のように命名され、ドライブ名という。通常は、c:にwindowsやPythonを入れ、d:に実験データや個人が作成したプログラムを入れることが多い。ワードで作成した文書や画像の様にひとまとまりのデータを「ファイル」と呼ぶ。したがって、Pythonのプログラムも実験データもファイルである。ファイルには名前を付けて区別するが、どのような種類のファイルか区別できると便利である。このため、ファイル名は、keio.docxのように、「.」より前が名前で、「.」の後がファイルの種類を表し拡張子と呼ばれる。ハードディスク上には何万個もファイルがあり、整理をしないと探し出すことができない。そこで、フォルダまたはディレクトリと呼ばれるファイルの入れ物を用意し、その中に収納して整理する。

ハードディスク上の特定のファイルを示す時には、ドライブ名、フォルダ名、ファイル名を指定する。windowsの規則では、

例：　c:ドライブ上のkeio.docx 　　　　 c:\keio.docx

d:ドライブ上のフォルダdata内のa.txt d:\data\a.txt

「\」もしくはバックスラッシュでドライブやフォルダ、ファイルの区切りを示す。ただし、Pythonやｃ言語のプログラム上では、「\\」のように2回繰り返す。

1. **フォルダを作成する**

レポート課題や配布物の作成や保存を行う前に授業の科目（物理学、化学、生物学）、種類（講義、実験）、課題名に応じてフォルダを作成する。以下ではMacの場合にフォルダを作成し、本日の資料を保存する。Windowsの場合はこの後のWindows でのフォルダ作成法を参照すること。

1. メニューからFinderをクリックする。
2. フォルダーの並んでいる欄の空きスペースを右クッリクして新規フォルダーを選ぶ。（下の図を参照。）
3. ファイル名を入力する。プログラミングでファイルを指定する際にはローマ字の方が便利であるのでプログラミングをする課題の場合にはローマ字名にすること。
4. ここでは、homeworkと入力する。
5. homeworkを選択した状態で空欄の縦列（赤枠）を右クリックする。
6. 再び新規フォルダーを選び、ファイル名を入力する。
7. 5)と6)を繰り返す。ここではhomework→physics→exp→pythonとフォルダを繰り返し作る。（下の図を参照。）

グラフィカル ユーザー インターフェイス, テキスト, アプリケーション, Word

自動的に生成された説明

**グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明**

グラフィカル ユーザー インターフェイス, テキスト, アプリケーション

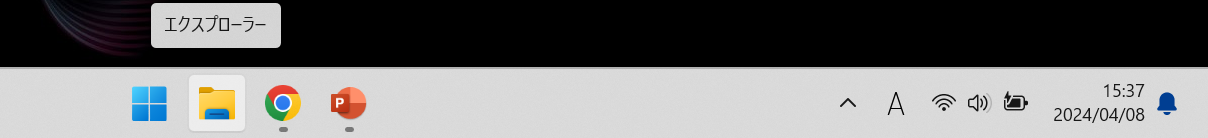
自動的に生成された説明

1. 続いて作成したフォルダに資料を保存する。Boxからダウンロードしたファイルは通常「ダウンロード」という名前のフォルダに保存される。メニュー欄のFinderを右クリックし、新規Finderウィンドウを選ぶ。フォルダ欄から「ダウンロード」を選択し、該当のファイル名を見つける。二つのFinderウィンドウが見える状態で資料をドラッグアンドドロップで作成したフォルダに移す。
2. **グラフィカル ユーザー インターフェイス, テキスト, アプリケーション

   自動的に生成された説明**同様にして次回以降の実験でもexpの中に新しく実験名のフォルダを作成して資料の保存やレポートの作成を行うこと。

Windows でのフォルダ作成法

1. エクスプローラーを立ち上げる。いろいろな方法がありますが、下記は一例。



グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

1. フォルダを作成するドライブを選ぶ。新品購入時は「C:」だけの場合が多いので、C:を選ぶ。

グラフィカル ユーザー インターフェイス, テキスト, アプリケーション

自動的に生成された説明

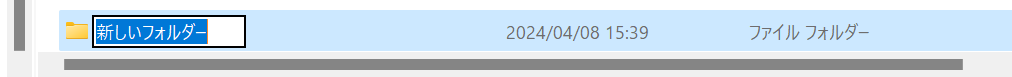
3)新規作成を選ぶ。

4)フォルダを選ぶ。

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

5) フォルダの名前を入力



**C　コンピュータ言語とPython**

コンピュータはonとoffまたは1と0を組み合わせた、いわゆる2進数のデータにより動き、これを機械語(Machine language)という。機械語を直接人間が書いてコンピュータを動作させることもできるが、難解で生産性が低い。そこで、人間に分かり易い英単語や10進数の数字でプログラムを書き（以後プログラムと呼ぶ）、これを機械語に翻訳する（コンパイルという）という方法が考案された。実際に書かれたプログラムをプログラム、コードもしくはソースコードと呼ぶ。コンパイルして機械語に変換されたものを実行型コード、実行プログラムなどと呼ぶ。

プログラムには様々な種類があり、プログラミング言語という。アセンブリ言語は機械語と1対1対応する言語で、2進数を組み合わせた機械語命令を英単語に置き換え、それなりに分かり易くした言語である。しかし、アセンブリ言語でプログラムを書くためには、コンピュータの内部構造を詳細に知る必要があるだけでなく、別な設計のコンピュータでは動かないという欠点がある。一方で、コンピュータサイドから見ると、人間がコンピュータのために最も効率よく動く実行型コードを用意してくれるのだから、短いコード長で最速処理ができる。今でも、処理速度を少しでも上げたい場合はアセンブリ言語で書く。

プログラムの開発時間や可読性(どのような処理をしているのか見てわかること)と実行したときの処理速度や実行型コードの長さや必要なメモリー量などは相反するため、プログラミング言語は無数にあり、用途に応じて適切に選ぶ必要がある。

c言語は、どのような設計のコンピュータでも大半が同じソースコードで動くという条件で、コンピュータ本体（ハードウエアという）の持つ性能を100％引き出すこと、すなわち実行速度（処理時間）(速く)と使用するメモリー量(少なく)を優先させたプログラミング言語である。言語仕様上、人間がコンピュータに合わせることを前提とし、ｃ言語の側は人間に配慮しない。したがって、分かりにくい点もあるが、コンピュータで出来ることは何でもできて処理速度が速く、安定した動作が期待できる。windowsやMacはc、c++言語で書かれている。cを拡張して使いやすくしたのがc++である。物理学や数学を専門とする研究者はc++を愛用している(私も愛用)。ｃやc++言語は、ソースコード全部を機械語に翻訳してから、コンピュータが実際に実行する。このような翻訳作業をコンパイルという。コンパイルしてから実行するため、プログラムミスを見つけるまでに時間がかかり、プログラム作成に手間取る。

Pythonは、可能な限り人間にとって分かり易いこと、データ処理上の合理性（最小限の指示で実行できる）を最重点に設計された言語である。したがって、人間にはありがたいが、コンピュータ側から見ると複雑なため、実行速度はc言語の1/10かそれ以下であり、さらに実行するためにかなり多くのメモリーが必要となるので最近まで流行らなかった。しかし、最近のコンピュータは高速で大量のメモリーを持っているため、このことが重要な欠点では無くなったため、cやc++を追い越して、最も使われているコンピュータ言語になった。また、Pythonは、初めにソースコード全体を機械語に翻訳するのではなく、一行または一部分を機械語に翻訳して実行し、終わったら次の行を機械語に翻訳して実行というようにしてプログラムを実行する。これを、インタプリタ方式という。この方式は、実行速度は遅いが、直ちに実行を開始するため、プログラムミス（エラーやバグという）を見つけやすい。また、一行書いて実行し、結果を見て一行書いて実行することができるため、学びやすく開発もしやすい。Pythonは、最も多くの人が使用している言語であり、使い易くするためのプログラム集（ライブラリ）が無数にある。ライフサイエンスや医療では、統計処理や機械学習のパッケージソフトが多用されている。

**D プログラムを作成して意味のある処理を行うとはどういうことか**

赤ちゃんは生まれたばかりのとき手足すら自由に動かせないが、しばらく練習すると自由に動かせるようになる。しかし、**手足を自由に動かせることと、手足を用いて意味のあることをする---外科手術や車の運転等無数にある---ことの間には大きな隔たりがある。**同じように、自由にプログラミングができるようになっても、それだけでは何もできない。

Pythonやｃなどプログラミング言語の目的は、cpuを人間が扱いやすいようにすることである。cpuはデータのやり取りの大半を主記憶装置との間で行うので、**プログラミングは必然的にcpuと主記憶装置とのデータやり取りに関連した部分が大半を占める**。具体的には、**様々な型を持つ変数を定義しこれをif, for, while, def, sin, cosなどを用いて操作**する。しかし、残念なことに、このような命令文や関数を理解して変数を自由に操作できるようになっても、これだけでは意味のあることは何もできない。諸君らが白昼夢でenjoyしていても他人にはなんの御利益が無いのと同じである。しかし、この白昼夢も、絵にしたり、小説にしたりすれば(場合によれば)大いに価値が出る。

cpu-主記憶装置間のデータやり取りだけでは無意味なので、入出力装置がある。人間でいえば、入力装置が目、耳、皮膚などであり、出力装置が音声の口、手足である。コンピュータ言語の基礎において、入出力装置としてキーボードの文字を読む(input())、ディスプレーに表示する(print())、補助記憶装置と主記憶装置のデータやりとり(file操作)などを習うので、入出力装置はこれだけと思いがちであるが、これは間違いである。Pythonには主記憶装置上のデータ、すなわち変数の型として、整数型・実数型・文字列型・リスト型など多数用意されている。これらの変数は実際には外界と対応しており、変数の操作は外界の操作である。例えば、ある実数型変数に患者の血圧としての意味があるとすれば、この実数型変数の値を変えることは患者の血圧を操作することに相当する。あるリスト型変数が画像データを表していれば、このリスト型変数の操作は、画像処理となる。

諸君らがコンピュータ言語を通じてコンピュータの扱い方を習得するときに、心にとめてほしいことは、**コンピュータは0, 1を組み合わせたデータを扱う機器であり、コンピュータ言語上では様々な型の変数として扱いやすくなっている。この変数で私たちが生活する実世界を表すとは、どのようなことなのか、どのようにすればよいのか、既にどのような規格が用意されているのか、**である。例えば、写真など静止画は、bit mapなど様々な規格で提供されるので、プログラム内でこれを2次元リスト(配列)に変換すれば、画像認識などの処理は自由にできる。音声であれば、例えばwavフォーマットなど様々な規格で提供されるので、1次元リスト(配列)に変換すればプログラムで音声認識などができる。パソコンのスピーカーで音を出力することは、然るべき変数に値を代入することであり、マイクロフォンから音の情報を得るためには然るべき変数の値を読み込むことである。数学を習うと様々な計算ができるようになるけれど、これだけでは実際の世界を扱うことができない。これはプログラミングの技術を習っただけでは何もできないのと同じである。物理学や化学・生物学の知識があれば、数学を活用して実生活で意味のあることができるようになる。同じように、0,1で現実世界を表す方法の習得も心がける必要がある。これは決して容易なことではなく、難しいことなので、コンピュータサイエンスという学問分野があり、専門家が研究している。

**E コンピュータにPythonを入れる**

・既にPythonを使用している場合は、各自の開発環境を用いよ。

・windowsを例にしているがMacでもほぼ同じである。

1) Pythonを入れる(次ページの図を参考にせよ。)

[https://www.Python.org/](https://www.python.org/)

を開く。Downloads-Windows installer(64-bit)を選ぶ。ただし、Windowsが32bitの場合は、-64の無いものを選ぶ。あとは指示に従えばよいが、Add Python to Pathにチェックを入れる。

グラフィカル ユーザー インターフェイス, アプリケーション, Word

AI によって生成されたコンテンツは間違っている可能性があります。Macの人のみ

インストール後に上のようなFinderが立ち上がるので、赤枠で囲ったInstall Certificates.commandをクリックする。

次に<https://github.com/thayata/hands_on_python.git>から.zshrcというファイルをダウンロードしてhomeディレクトリに置く。

携帯電話の画面のスクリーンショット

AI によって生成されたコンテンツは間違っている可能性があります。グラフィカル ユーザー インターフェイス, アプリケーション, Web サイト

AI によって生成されたコンテンツは間違っている可能性があります。下図のcodeを押して, Download ZIPからファイルをダウンロードする。

グラフィカル ユーザー インターフェイス, アプリケーション, Web サイト

AI によって生成されたコンテンツは間違っている可能性があります。ダウンロードしたファイルをFinderで開く。Command+shift+.(ピリオド)を同時に押すと.zshrcというファイルが半透明で表示されるので, このファイルをhomeディレクトリ(家のマークが表示されているディレクトリ)に移動させる。

2) visual studio codeを入れる

Pythonはwordのような文字入力のできるアプリ(Text Editor)でcodeを書き、実行する。visual studio code はcode入力や実行、問題点の洗い出しなどを一つにまとめた(統合開発環境)アプリであり、ほかにもanacondaやJupyter notebookなどがあり、どれでもよいが、ここでは表題のアプリで説明する。

<https://azure.microsoft.com/ja-jp/products/visual-studio-code/>をクリックしてダウンロードする。**デスクトップにアイコンを作成**すると立ち上げ易くなる。

3）visual studio code でPythonが使える設定にする

・visual studio codeを実行し、open folderから先程作成したフォルダを選ぶ。

・左端の四角形が４つの(下の図で赤く囲った)項目を選び、出てくる窓にPythonと入れる。一番上のPythonのInstallを押す。

コンピューターの画面のスクリーンショット

AI によって生成されたコンテンツは間違っている可能性があります。

・New Text Fileをクリックして、

print("Hello World")

をコピペして貼り付ける。Pythonは、左側のスペース(空白)に処理上の意味があるため、上記のコードは左側にスペースを入れないこと。CtrlとSを同時に押す (MacはcommandとSを同時に押す)。ファイル名を求められるので、a.pyをタイプして保存する。.pyにより、Pythonのプログラムであることをvisual studio codeが認識してこれに合う処理を行う。黄色や赤に色が変わるはずであるが、変わらなくても気にしない。

・プログラムの実行

右上の三角ボタンを押す。下にグラフィカル ユーザー インターフェイス, テキスト, アプリケーション

AI によって生成されたコンテンツは間違っている可能性があります。Hello Worldが表示されれば一先ず成功。





グラフィカル ユーザー インターフェイス, テキスト

自動的に生成された説明

visual studio codeを立ち上げて、使えるように設定する。

Pythonが使える状態になると必要な物をインストールするようにいろいろ勧められるので、インストールする。下のターミナル窓で、

pip install numpy　エンター

pip install matplotlib　エンター

を実行して、ライブラリーを入れる。ターミナル窓がない場合は上のTerminalからNew Terminalを選ぶ。他にはscipy, pandasなど無数にあるが、今回使用するのは上記。

**F Python入門**

簡単な統計処理や数値計算、グラフの作図に必要なPythonの基本を覚えよう。

1. **プログラムを実行してみる**

ここでは、visual studio codeを用いてプログラムの作成と実行を行う。

1. visual studio codeを立ち上げる：　デスクトップのアイコンをクリック。
2. コードを書く：

File – New Fileをクリックする。

print('Python is running. Example of the command of input() ')

a = input('type 1 and enter on the terminal window >> ') # 文字列として扱われる

b = input('type 2 and enter on the terminal window >> ')

print(a+b)                #文字列なので、両者の結合

print(a,'+',b,'=',int(a)+int(b))      #文字列を整数に変換して、足し算

をコピペして貼り付ける。

File – Save As をクリックして、自分の作成したフォルダ(セクションBで作成したフォルダ)にファイル名として例えばa.pyをタイプして保存する。Pythonは、左側のスペース(空白)に処理上の意味があるため、上記のコードは左側にスペースを入れないこと。

1. 実行する：このためには、Run – Run Without Debugging。

**input(‘文字列’)は、ターミナル窓に文字列を表示し、キーボードを読み取り**文字列型として出力するコマンドである。したがって、このプログラムを実行すると、type 1 and enter on the terminal window >>と表示されるから「1エンター」を入力する。続いて、type 2 and enter on the terminal window >>と表示されるから「2エンター」を入力する。12と1+2＝3が表示される。12は文字1と2をつなげただけ、3は1+2である。int(a)は文字列型aを整数型に変換する。

**ｂ）変数**

Pythonに限らず、コンピュータ言語には変数があり、プログラムの役割は変数の値を適切な値に設定することである。プログラミングの操作対象が変数なので最も重要であり、様々な形式の変数が用意されているだけでなく、それらを組み合わせ自分で定義することもできる(class, オブジェクト指向)。Pythonの特徴は、変数をあらかじめ宣言する必要がなく、使用するときに代入する値で決まる。

様々なデータの型があり、代入したデータにより自動的に決まるため、どのような型か分からなくなる。そのような時は、typeを使う。たとえば、type(A)の入力によりintという結果が得られるが、int=integer=整数なので、Aが整数型ということが分かる

A=100     #整数

B=3.1415  #実数型

C=[1,2,3] #リスト型 1次元

D=[[1,2],[3,4]]  #リスト型 2次元

E='Keio Univ. School of Med.' #文字列型

  # ' 'もしくは" "で囲むと文字列型変数となる

F={'hiyoshi': 0, 'shinanomachi': 377} #ディクショナリー

G = True    # ブーリアン TrueとFalseがあり、条件判断に用いる

H = False

print("A: ", type(A))

print("B: ", type(B))

print("C: ", type(C))

print("D: ", type(D))

print("E: ", type(E))

print("F: ", type(F))

print("G: ", type(G))

print("H: ", type(H))

様々な変数型が用意され、それぞれを処理するための関数が多数用意されているので、使い方を覚える必要がある。英単語より数が少ないので、英語の学習より遥かに簡単である。

"　"もしくは' 'で囲まれた文字は文字列変数とみなされる。

**b1) リスト型**

コンピュータの素晴らしいところは、大量のデータを扱うことができる点である。大量のデータ一つ一つに名前を付けて割り当てることは不可能である。そこで、Pythonでは0,1,2,...の整数で番号をつけてデータの順番を識別する。複数の変数をひとまとめにして、整数で識別する変数をリスト型という。リスト型の内部には、整数型、文字列型など様々な変数が入る。

a=[0,1,2,3,4,5,6,7,8,9]  #リスト型変数の作成と初期値の代入

print(a)

print(len(a))  #リストの長さ

print(type(a)) # aの変数型を表示

print(a[0])     #最初の要素を表示

a[2]=100  #3番目の要素に100を代入

a[7]=3.14

print(a)

print(a[3:5])  # 3番目から5番目の前まで

print(a[2:])   # 2番目から最後まで

print(a[:2])   #最初から2番目の要素まで

print(a[:-2])  #最初から最後の要素の2つ前まで

print(a[-2:])  #最後から2つ前から、最後まで

print(a[::2])  #最初から、2ステップで最後まで。0,2,4,6,8

print(a[1::2]) #2番目から、2ステップで最後まで。1,3,5,7,9

a.append(100)   #要素の追加

a.append(101)

print(a)

print(len(a))

b=[ ]        #bはリスト型であるとして事前に宣言

b.append("keio")

b.append("univ.")

b.append("school")

b.append("of")

b.append("med.")

print(b)

print(len(b))

**b2) ディクショナリー型**

ディクショナリー（辞書）型はキーと値を対にしたデータ {キー:値, キー:値,..}

リスト型が整数を引数にして内部の変数を指定しているのに対して、ディクショナリー型は、keyとなる文字列で内部の変数を指定する。

fare = {'hiyoshi': 0, 'shinanomachi': 377, 'mita': 398,'sfc':617, 'tsurumi':294}

print (fare)

print(fare['mita'])

fare['sfc']=1000

print (fare)

**b3) ブーリアン型**

ブーリアン型(bool)はTrueとFalseの二つの状態があり、条件判断に用いる。

adult = True

minor = False

type(minor)

print(not minor) #True Falseの反転

# 比較演算子　 < > ==(等しい)は結果をbool で返す

print("3 < 1:  ",  3 < 1)

print("3==3: ",  3==3)

print("3 >= 3: ", 3 >=3)

# orは1つでもTrueならTrue, andはすべてTrueのときTrue

print("3 >= 3 or  1<0: ", 3 >=3  or 1< 0)

print("3 >= 3 and 1<0: ", 3 >=3  and 1< 0)

if True:

    print('True')

else:

    print('False')

if minor:

    print('True')

else:

    print('False')

if 5<2:

    print('True')

else:

    print('False')

if文は、ifと：の間に挟まれた式を評価して Trueなら次を実行Falseならelseを実行する。else文は省いてもよい。「5<2」のような文は、真偽を判断して正しければTrue間違っていればFalseを返す。「5＜2」は間違いなのでFalseが返されてif でこの結果に従った処理をする。「if True:」は直接Trueを入れた例であり、常に次の文を実行する。プログラムとして，無意味であるが理解を促進させるための例として入れた。

**ｃ）ライブラリーNumPyの導入**

ライブラリーは他人が作成し、自由に使えるように用意してくれたプログラムの集まりである。NumPyは複雑な数値計算を行うためのライブラリーであり、この中にはsin,cosをはじめとして数値計算に必要な関数やデータの型などが入っている。

関数各種

import numpy as np

a=10

b=np.log(a)

c=np.exp(a)

d=np.sin(a)

e=np.cos(a)

f=np.tan(a)

g=np.sqrt(a)    # root

h=np.power(a,5) # aの5乗

i=a\*\*5          # aの5乗

print(a,b,c,d,e,f,g,h,i)

1次元配列

import numpy as np  #NumPyを組み込み、「np.sin(x)」のようにしてNumpPyの関数を使う

a=[1,2,3]

b=[4,5,6]

x=np.array(a) #numpyの1次元配列に変換

print("x=",x," len(x)=", len(x))

# xは1次元配列

print("np.dim(x)=", np.ndim(x))

y=np.array(b)

print(x+y)   # element-wise addition

print(x-y)

print(x\*y)

print(x/y)

print(x/100)

2次元配列

import numpy as np

a=[[1,2],[3,4]]

b=[[5,6],[7,8]]

x=np.array(a)

x=np.array(a) #numpyの2次元配列に変換

print("x=",x," len(x)=", len(x))

# xは2次元配列

print("np.dim(x)=", np.ndim(x))

y=np.array(b)

print('add ', x+y)   # element-wise addition

print('sub ',x-y)

print('mul ',x\*y)

print('div ', x/y)

print(x/100)

**ｄ）繰り返し**

コンピュータの特徴は、単純作業を何回も繰り返してできること。

for と whileがある。

**d1）for**

n=0

for i in range(1,10,2):

n += i

print(n)

for は指定した範囲内の要素を順次取り出し、**4スペース右(通常はtab)にずらした領域を繰り返し実行する**。forの行の最後に「:」をつける。

これは、1+3+5+7+9の計算例である。for…rangeはまず初めにiに1を代入し、2を足しながら、4スペース右へずらした下のコードを繰り返し、iが10かそれ以上になったら繰り返しをやめるという意味である。n += i　は、ｎにiを加える命令である。print(n)は左端から始まるので、for文が終わってから1回実行される。

繰り返しの例

π^2/6 = Σ1/ｎ＾2

を用いて円周率を求めてみよう。

import numpy as np

n=0

for i in range(1,1000):

n += 1/i\*\*2

print(np.sqrt(6\*n))

ルート（√）の計算は標準では備わっていないので、numpyという計算用のプログラム（モジュールという）を読み込んで、npという名前で登録する。npの中にあるルート関数がnp.sqrtである。このプログラムでは、printも4スペース右にずらしているから、forの内部で繰り返す。

forで順番に取り出す要素は、様々な方法で指定できる：

#直に範囲指定

n=0

for i in range(1,10,2):

    print(i)

    n += i

print("range(1,10,2): ",n)

#　リスト内の要素を順番に指定

b=[2,3,4,5,6,7,8,9,10]

n=0

for i in b:

    print(i)

    n += i

print("b[] : ",n)

#　リスト内の要素を[2:8:2]で取り出す

n=0

for i in b[2:8:2]:

    print(i)

    n += i

print("b[2:8:2] : ",n)

#　要素が文字列のリストでもよい

b=[ ]        #bはリスト型であるとして事前に宣言

b.append("keio") # appendは要素の追加

b.append("univ.")

b.append("school")

b.append("of")

b.append("med.")

n="affiliation:"

for i in b:

    print(i)

    n +=" "

    n += i

print("Chain of the words ",n)

ディクショナリーを用いた例

fare = {'hiyoshi': 0, 'shinanomachi': 377, 'mita': 398,'sfc':617, 'tsurumi':294}

print (fare)

print(fare['mita'])

for key in fare:

    print(key)

for key,value in fare.items():

    print(key,value)

fare['sfc']=1000

print (fare)

**d2）while**

whileの右がTrueの間は、4スペース右にずらした部分を実行。「：」をつける。

# whileの右がTrueの間は、4スペース右にずらした部分を実行。「：」をつける

#　Falseは実行されない

while False:

    print("Not executed ")

print("Next commands")

# aがFalseになるまで実行

a=True

n=0

while a:

    n += 1

    if n >= 10:

        a=False

print("n=",n)

# cn <= 10 がFalseになるまで実行

n=1

cn=1

while cn <= 10:

    n  \*= cn  #  n\*cnをnに代入

    cn += 1

print("10!=",n)

#　要素が文字列のリストでもよい

b=[ ]        #bはリスト型であるとして事前に宣言

b.append("keio") # appendは要素の追加

b.append("univ.")

b.append("school")

b.append("of")

b.append("med.")

b.append("EndOfWords")

n="affiliation:"

cn=0

while (not b[cn]=="EndOfWords") and ((len(b)-1) > cn):

    n += " "

    n += b[cn]

    cn +=1

print("Chain of the words ",n)

**d3）強制終了の方法**

以下のブログラムは、無限ループを用いており、終了しない。実行には注意が必要。これは、意図的に無限ループにしたけれど、プログラムミスでなることはよくある。このような時、windowsはctrl+c, Macはcontrol+cで強制終了できる。

強制終了の方法はこのほかに、上のメニューから「Run」「Stop Debugging」でもよい。こちらの方が終了しやすいかも。

このプログラムはwindowsはctrl+c, Macはcontrol+cでとまればよいが、止まらない可能性もあるので、

#このプログラムは不用意に実行しないこと

cn=0

while True:

    print("Endless Running cn=", cn)

    cn  += 1

こちら(下)で試してから、実行する方が安全。これは、cn>10000で終了するので、10000になる前に、windowsはctrl+c, Macはcontrol+cを試すこと。「Run」「Stop Debugging」でもよい。

cn=0

while True:

    print("Endless Running cn=", cn)

    cn  += 1

    if cn > 10000:

        break # whileやforの強制終了

print("Exit infinite loop")

**e）関数を定義する**

def func(x): #関数の定義

return x\*\*2

print(func(4)) #関数の実行　4の二乗の計算

これは、funcという関数の定義で、戻り値は、引数xの2乗である。

**f）ファイルを読み込んで、表示**

以下のプログラムはpath=” ”で指定したファイルを読み込む例である。東京都のコロナ陽性者数をboxにupしたので、data20230505-20200120.txtをダウンロードせよ。このとき、Windowsでは、(デフォルト設定では)PC>ダウンロードというフォルダにダウンロードされる。各自、セクションBで作成したフォルダにコピーして、以下のpathにコピー先を記入せよ。ファイルの場所を正確に指示しないとエラーになる。Pythonはデフォルトでは日本語対応していないので、アルファベットなど半角の文字でフォルダを作成してすること。

以下のプログラムではd:ドライブのdataとう名前のフォルダにファイルがある場合の例である。d:\data\\*\*\*\*.txtを読み込み、初めの20行だけ表示する。windowsでは、フォルダやファイル名の区切りは「\」で指示するが、Pythonのプログラム内では単独の「\」には別な意味があるので、「\\」とする。

path = "./data20230505-20200120.txt"     #ファイル名を入れる.自分の環境に修正

with open(path) as fileobj:           #ファイルを開いて、fileobjという名前にする

    datalist =  fileobj.readlines()

for s in datalist[:20]:

    print(s)

**ｇ）グラフを書く**

matplotlib は図やグラフを書くためのライブラリー。

例が沢山ある：<https://matplotlib.org/tutorials/> 　これをコピペして練習したほうが良いかも。

一番簡単な方法(Pyplotインターフェース)

import numpy as np

import matplotlib.pyplot as plt

# x軸のデータ点を配列として作成

x=np.arange(0,6,0.1)  #0から6まで0.1刻みの配列

print(x)

y=np.sin(x) # yを配列として作成。

# y[0]=sin(x[0]), y[1]=sin(x[1]),...配列xの最後まで。

plt.plot(x,y) #  図としてのグラフデータ作成。まだ表示しない

#図をファイルとして保存できる。

plt.savefig("D:\\data\\Fig.pdf")

plt.savefig("D:\\data\\Fig.jpg")

plt.savefig("D:\\data\\Fig.png")

plt.show() # 図の表示

二つ描く

import numpy as np

import matplotlib.pyplot as plt

x=np.arange(0,6,0.1)  #0から6まで0.1刻みの配列作成

y=np.sin(x)

z=np.cos(x)

plt.plot(x,y,label="sin")

plt.plot(x,z,linestyle="--",label="cos")

plt.xlabel("x")   #x軸のラベル

plt.ylabel("y")

plt.title("Example of the plot")

plt.show()

すこし凝った方法(Object-Oriented API. In general, try to use the object-oriented interface over the pyplot interface.

<https://matplotlib.org/tutorials/introductory/lifecycle.html#a-note-on-the-object-oriented-api-vs-pyplot>)

import numpy as np

import matplotlib.pyplot as plt

# Make some fake data.

a = b = np.arange(0, 3, .02) #定義域を0-3まで0.02刻み

c = np.exp(a)

d = c[::-1]             #cの順番を逆にする

print('c=',c,'d=',d)

# Create plots with pre-defined labels.

fig, ax = plt.subplots()  #ここからがObject-Oriented API

ax.plot(a, c, 'k--', label='Model length')

ax.plot(a, d, 'k:', label='Data length')

ax.plot(a, c + d, 'k', label='Total message length')

legend = ax.legend(loc='upper center', shadow=True, fontsize='x-large')

# Put a nicer background color on the legend.

legend.get\_frame().set\_facecolor('C0')

plt.show()

**東京都コロナウイルス陽性者数のグラフを描く**

データ入手先　<https://stopcovid19.metro.tokyo.lg.jp/>

(2023年5月8日で公開終了)

(pyplotインターフェース)

import numpy as np  #NumPyを組み込み、npという名前でNumpPyの関数を使う

import matplotlib.pyplot as plt

path =  "d:\\data\\data20230505-20200120.txt"    #ファイル名を入れる(要修正)

pathgraph ="d:\\data\\coronapos.pdf"   #pdf形式でグラフを保存する場所(要修正)

with open(path) as fileobj:           #ファイルを開いて、fileobjという名前にする

    dlist =  fileobj.readlines()

datalist = dlist[::-1]          #古い日付から新しい日付の順にに入れ替え

data = [ int(s) for s in  datalist] #文字データを数字データに変換

print(data)

plt.plot(data)

plt.xlabel("from 20200120 to 20230505")

plt.title("Coronavirus positives in Tokyo ")

plt.savefig(pathgraph)   #pdf形式でグラフを保存

plt.show()

**1次回帰と指数関数回帰**

実行するとグラフが表示されるので、虫眼鏡マーク(Zoom to rectangle)を選び、回帰する領域を選択して拡大し、×をクリックして次へ進む。

スクリーンショット, 抽象, 地図 が含まれている画像

自動的に生成された説明

import numpy as np  #NumPyを組み込み、npという名前でNumpPyの関数を使う

import matplotlib.pyplot as plt

pathdir ="d:\\data\\" #データファイルの保存先、要修正

path = pathdir + "data20230505-20200120.txt"  #データファイル名

with open(path) as fileobj:           #ファイルを開いて、fileobjという名前にする

    dlist =  fileobj.readlines()

datalist = dlist[::-1]          #古い日付から新しい日付の順にに入れ替え

datanp = [ int(s) for s in  datalist]  #文字データを数値に変換

datalen=len(datanp)

print("Data length=",datalen)

plt.style.use('default')             #描画スタイルは通常の形

fig, ax = plt.subplots()

fig.suptitle("COVID-19 positive in Tokyo", fontsize = 16)

ax.plot(datanp, label = "Reported")     #描画用データ作成(表示しないがここで図を作成)

ax.legend()

ax.set\_xlabel("days")

ax.set\_ylabel("COVID-19 positive / day")

plt.savefig(pathdir+"Fig1.pdf")   #図の保存

plt.savefig(pathdir+"Fig1.png")   #図の保存

plt.show()                       #図を表示する

#　×をクリックして図を閉じた後の処理

xmin, xmax = ax.get\_xlim()       #選んだ領域の最小、最大

xmin=int(xmin)

xmax=min(int(xmax),datalen)

print("xmin=",xmin, "xmax=",xmax)

#統計処理関数

#ほとんどの場合、自分で関数を書かなくても用意されているが、今回は練習のため作成

# data（リスト）のstartからendまでに対する統計

def getstat(data, start, end):

    sumx=0

    sumy=0

    sumxy=0

    sumxx=0

    for i in range(start, end):

        sumx  +=  i

        sumy  +=  data[i]

        sumxy +=  i\*data[i]

        sumxx +=  i\*i

    n= end-start

    a=(n\*sumxy-sumx\*sumy)/(n\*sumxx-sumx\*sumx)

    b=(sumxx\*sumy-sumx\*sumxy)/(n\*sumxx-sumx\*sumx)

    return a,b  #  y=ax+b

a, b = getstat(datanp, xmin, xmax)

print("Linear a b ", a, b)

logdata = [ np.log(s) for s in  datanp[xmin:xmax] ]  # 対数変換

k, logg = getstat(logdata, 0, xmax-xmin)

print("Exponential  k log(g) ", k, logg)

#関数の定義

def LinearReg(x, a,b):     # 1次回帰

    return  a\*x+b

def ExpReg(x,xmin, k,logg):  #指数関数回帰

    return  np.exp(k\*(x-xmin)) \* np.exp(logg)

fig2, ax2 = plt.subplots()

fig2.suptitle("COVID-19 positive in Tokyo", fontsize = 20)

#描画領域を2倍にしてグラフを書く

x2min = xmin-(xmax-xmin)

x2max = xmax+(xmax-xmin)

y2min=0

y2max = np.max(datanp[x2min:x2max])

x = np.linspace(x2min, x2max, x2max-x2min)

ylin = LinearReg(x,a, b)

ax2.plot(x,ylin, label="Linear  Regression",linestyle='dashed')

yexp = ExpReg(x,xmin, k,logg)

ax2.plot(x,yexp, label="Exponential Regression",linestyle='dashed')

ax2.plot(datanp, label="Reported")

ax2.plot(np.arange(xmin, xmax),datanp[xmin:xmax])  #選択領域データを色を変えて表示

ax2.legend()

ax2.set\_xlabel("days")

ax2.set\_ylabel("COVID-19 positive / day ")

ax2.set\_xlim(x2min, x2max)

ax2.set\_ylim(y2min, y2max\*2)  #y方向の上の部分にlabelを書くスペースを作る。

plt.savefig(pathdir+"Fig2.pdf")

plt.savefig(pathdir+"Fig2.png")

plt.show()

kやlog(g)の意味は「物理学実験 第 1 回課題１ 統計処理 (最小 2 乗法)」

を参照のこと。指数関数で最小二乗法を行う方法の記述に記載してあります。

**応用例：　モンテカルロ法により円周率を求める**

モンテカルロ法とは業界用語であり、乱数を用いてシミュレーションする場合に用いる。ここでは、-1から1までの一様乱数を2回発生させ、x-y座標に点を打つ。この点が半径1の円の内部にあればカウントを1つ増やし、外部ならそのまま。これを例えば10000回繰り返えし、円の内部に点が来る確率が求める。ここから、円の面積を求めて円周率を求める。

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.patches as patches

def InCircle(x,y,r):  #円の内部ならTrue, 外部ならFalse

    return  x\*x + y\*y < r\*r

def ransu():     # -1 to 1 の一様乱数

    return 2\*np.random.rand()-1

N=1000

cn=0

xin=[]

yin=[]

xout=[]

yout=[]

for n in range(1,N):

    x=ransu()

    y=ransu()

    if InCircle(x,y,1): #円の内部か否か判断　Trueが内部

        cn += 1

        xin.append(x)

        yin.append(y)

    else:

        xout.append(x)

        yout.append(y)

    print("pi = ", 4\*cn/n)

fig = plt.figure()

ax = plt.axes()

ax.scatter(xin,yin,c='red')  #点を打つ

ax.scatter(xout,yout,c='black')

#円を描く

c = patches.Circle(xy=(0, 0), radius=1.,  ec='red', fill=False)

#正方形を描く

r = patches.Rectangle(xy=(-1, -1), width=2., height=2., ec='black', fill=False)

ax.add\_patch(c)

ax.add\_patch(r)

fig.suptitle("Monte Carlo method  pi="+str(4\*cn/n), fontsize = 20)

plt.axis('scaled')

ax.set\_aspect('equal')

plt.show()

**課題**

1. 掲載したプログラムを実行せよ。物理学実験用のフォルダを作り、プログラムは作成したフォルダに保存すること。
2. コロナウイルス陽性者数の解析をできる範囲で行う。

今日のPythonコードは<https://github.com/thayata/hands_on_python.git>からもダウンロードできる。

**(参考)オブジェクト指向プログラミング**

今まで説明してきたような規模の小さいプログラムの場合には、上記で説明してきた書き方がベストである。**Pythonは短いコードでかなりのことができるのが利点**なのであり、諸君がこれから定量解析を行う上での力強い味方となる。一方で、このような方法----**上から下に一筆書きで実行していく**----でプログラムを書くと、規模の大きいプログラムを作成する場合には混乱が生じプログラムミスの除去が難しくなる。**オブジェクト指向プログラムは、大規模化を安全に行うための最も(現段階で)優れたな方法**である。マニアックな説明であり、興味が無い場合はここで終了しても課題をこなすために問題はない。以下に、オブジェクト指向プログラミングにとって重要な5項目について説明する。

**1）変数と関数の一体化:**

コンピュータプログラムの役割は、変数の値を操作することであり、変数は実世界の状態を表す様々な数値や文字・絵などである。変数にはその変数を処理するための関数が必ずあり、**変数と関数は一体**の関係である。例えば、成績処理では、実数型の配列phys[n]があり、添え字(インデックス)nが学籍番号、値がテストの点数としよう。phys[n]にはnを指定して成績を代入する関数set(n,score)、成績を取り出す関数getscore(n)、平均点を求める関数getmean()、SABCDを決める関数getgrade()などが付随する。**変数と関数は必ず対になるのだから、一つにまとめて名前をつけ、1つの変数として管理する。このことをプログラミングの最優先事項として、プログラム開発をしようというのがオブジェクト指向プログラミングである**。

**2） 単体で機能する完成品「オブジェクト」:**

**変数(データ)と関数を対にして一体化**にはもう一つの別な捉え方がある。例えば、テレビは画面・スピーカーというデータ表示器(変数)があり、これを操作するためにチャンネルを切り替えるボタン(関数)、音量を調節するボタン(関数)、電源ボタン（関数）、アンテナ端子(代入関数)などがある。それだけでなく、この全てが1つにまとまり、テレビという単体で動作する完成した製品(オブジェクト)になっている。**変数と関数を対にして一体化すると、これだけで機能する1つの完成した製品（オブジェクト）とすることができる**。Pythonには整数型、リスト型など事前に用意された様々なデータの型があるが、オブジェクトはユーザーが定義できるデータの型であり、他のデータ型変数と同じように扱うことができる。

**3）量産化:**

テレビと言えばどこのメーカーのどの機種でも基本的な機能はほとんど同じであるが、画面の大きさやボタンの位置、細かい機能の有無など微妙な違いがある。特定のテレビを1台直接作るのでは無く、**テレビ製作のための鋳型(classという)を作成し、鋳型で作成(作成したものをinstance(実体化)という)したのち、ボタンの配置など細かいことは個々のinstanceで個別に調整**していけば(テレビなど実世界の物では難しそう)、大量生産が容易にできる。Pythonではclass文によりオブジェクトの鋳型を定義し、この鋳型をもとに個々の変数を作成するという方法でオブジェクトを作成するので、量産化が容易であり、このおかげで大規模プログラム作成が簡単にできるようになる。

**4) クラスの階層化：**

鋳型により作成されたオブジェクト(instance)は変数であり、実数型変数や文字列型変数と同じように扱うことができる。このため、class文でオブジェクトを定義するとき、組み込む変数は既存の整数型やリスト型だけでなく別なクラスのオブジェクトでもよい。これにより、階層構造を構築することができる。例えば、calss cell:を定義して細胞の働きをするオブジェクトを作成したとする。人間の内蔵は細胞が多数集まった物なので、class organ:オブジェクトを定義し、この中の変数に多数のcell オブジェクトを入れて、内蔵の働きをさせる。人間を内蔵の集合体とすれば、class human:オブジェクトには変数としてorganオブジェクトが多数含まれ、相互に機能しあうように作ることができる。humanオブジェクトでmitsuiというinstanceを作成して、身長など必要なパラメータを代入し、このinstanceの関数を実行して10年後を予測するという具合である。また、humanオブジェクトで1000人くらいのinstanceを作り、人間どうしの相互作用・社会性の観察を行うことができる。このようなプログラミングスタイルが**上から下に一筆書きで実行していく**スタイルと大きく異なるのは理解できるだろう。

**5) プログラミングを始める前にどのようにオブジェクト化するか入念に設計する必要がある:**

　オブジェクト化プログラミングでは何をどのようにオブジェクト化するかが最重要課題であり、プログラム作成に先立ち、入念に設計する必要がある。従来型のcodeであれば、何気なくcodeを書き始めても試行錯誤すればゴールにたどり着けるが、オブジェクト指向ではそうはいかない。

**まとめ：**変数と関数の一体化、これにより単体で動作する1つの製品(object)となる。オブジェクトは鋳型(class)を定義して鋳型を基に作成(instance)するので、量産化ができる。別なオブジェクトを組み合わせ、階層化を行うこともできる。事前にどのようにオブジェクト化するか入念に設計する必要がある。

　諸君の成績管理をオブジェクト指向プログラミングで行う場合を考えよう。先に、phys[n]があり、これを処理するために、set(n,score)、getscore(n)、getmean()、getgrade()付随する例を挙げたが、これは、上から一筆書き型のプログラムを作成する場合には適切だがオブジェクト指向では不適切である。**オブジェクト指向ではコードを書き始める前にどのようにオブジェクト化するか入念に設計する必要がある**。すなわち、**何をひとまとまりの完成品として扱うか**である。学生の成績管理なのだから、学生個人をひとまとまりの完成品として扱うべきである。class studentphys: には、name(文字列型変数)、bango(整数)、phys(物理の点数、実数型)などの変数(object内で使い捨てにしない変数は**データ属性data attribute**という)が含まれる。また、これらのデータ属性を処理する関数(object内の関数を**method**という)には、getphys(), setphys()などがある。

　これが、先の**上から一筆書き型**とどこが違うのか？一人分のデータを扱う場合には大差ないが、多数の学生を扱う場合には、class studentphys:を鋳型にしてinstanceを作成する。これらのinstanceを配列の要素として多数の学生を扱う。残念であるが、この程度の小規模だと有難さは体感できないかも知れない。

windowsやmacでは1つのディスプレーにword,excel, web browserなど多数の画面を配置できる。1つの画面をオブジェクトとして扱い、window class:を鋳型にした個々のinstanceがwordの画面であり、excelの画面であり、web browserの画面である。いくつ画面があろうと、1つのclassで管理できるから、手間は1個の場合と同じである。長らく説明したが、数学者は頭が？？？?としか言いようのないほど優れた発明・発想の転換であり、相応に高度な概念なので当然。

**PythonでObject Oriented Programming**

class Student:

    def \_\_init\_\_(self, name, age): #コンストラクタ

        #コンストラクタはinstance作成時に1回だけ実行される

        self.name = name        #instance variable (データ属性)

        self.age = age

    def printNameAge(self):  #引数の必要が無くてもselfは入れる

         print(self.name,self.age )

    def setNameAge(self,name,age):

        self.name = name

        self.age = age

#instance作成

keiotaro=Student("KeioTaro", 18)  #コンストラクタ実行

keiotaro.printNameAge()

#値の変更

keiotaro.setNameAge("MitaHanako",19)

keiotaro.printNameAge()

#このインスタンス固有のデータ属性(instance variable)を追加する

keiotaro.phys=55

keiotaro.phys

#配列の要素にできる

igakubu=[keiotaro, Student('Aoki',20), Student('Sugimoto',18),Student('Fujisawa',17), Student('Hibi',22)]

for n in igakubu:

    n.printNameAge()

・**コンストラクタはこのclassのinstanceを作成するときに1回実行するmethod**.

・Pythonのmethodはself (今実行しようとしているinstanceのこと)を第1引数にする必要があるので、引数の必要が無いmethodでもselfという引数を入れる必要がある。selfが頻繁に出てきて邪魔であるが、たとえばself.mean=0の実行は、今実行しているinstanceに変数meanを定義して追加する(このinstanceのデータ属性とする)という意味になる。オブジェクトの中のどの関数でどのように実行しようと、self.mean=0が出てきた時点で定義されデータ属性としてinstanceに付加されるから分かりやすい。オブジェクトの定義の外でもinstanceの名前.変数名とすればデータ属性として加えてもらえるのでPythonは柔軟である。また、selfが無いと、meanはクラスの外部で定義していればそれであり、クラス外部で定義していないとこのinstanceで用いる一時的な使い捨て変数となる。以下のプログラムの変数a,bはselfを使っていないから、作成されたinstanceのデータ属性として残らず、method”getsum”が終わると破棄される。

class Student:

    def \_\_init\_\_(self, phys,chem): #コンストラクタ

        self.phys = phys        #instance variable

        self.chem = chem        #instance variable

    def getmean(self):

        self.mean = (self.phys+self.chem)/2.

        return self.mean

    def getsum(self):

        a=1+mean   #このmeanは外部で定義されたmean

        b=2        #a,bは使い捨て変数

        self.c=a+b

    def printx(self):

        print("x=",self.x)

        # xはクラス内部で定義していないので、このままならエラー

mean=1000.

mitsui = Student(80,70)

furuno = Student(100,100)

print("mitsui mean=", mitsui.getmean(),"mean=", mean)

mitsui.getsum()

print("mitsui c=", mitsui.c)

#print("mitsui c=", mitsui.a, mitsui.b) #エラーになる

#mitsui.printx() #xはクラス内部で定義していないので、このままならエラー

mitsui.x=1234  # instance mitsuiのデータ属性としてxが加えられた。

mitsui.printx()

**まとめ：**オブジェクトは変数と関数をひとまとめにしたもので、classで定義する。classを鋳型にして作成した物がinstance(実体化)である。instanceは作成されるときconstructorが1回実行される。constructorはdef \_\_init\_\_(self, ... ):のように書く。class内部で定義される関数をmethodと呼び、instanceと共に残る変数をデータ属性(data attribute)といいself.変数名として扱う。methodにも種類があり、class内部で定義した他のmethodのみが実行を許可されるprivate method(このテキストでは説明しない)とclass外部で普通に使えるpublic method(説明したmethodは全てこれ)がある。データ属性にも種類があり、instance variable, class variable(下記参照)がある。

データ属性には、以下の種類がある：

instance variableは作成した特定のinstance内のみで決まる変数。

class variableは同じclassの全てのinstanceで共通の値となる変数。

class Student:

    gradeS=90  # class variable

    gradeA=80  # classの直下で定義するとclass variable

    gradeB=70

    gradeC=60

    def \_\_init\_\_(self, name, phys): #コンストラクタ

        #コンストラクタはinstance作成時に1回だけ実行される

        self.name = name        #instance variable

        self.phys = phys        #instance variable

    def getNameSco(self):

         return self.name,self.phys

    def setSco(self, phys):

        self.phys = phys

    def getgrade(self):

        if self.phys >= Student.gradeS:

            return "S"

        if self.phys >= Student.gradeA:

            return "A"

        if self.phys >= Student.gradeB:

            return "B"

        if self.phys >= Student.gradeC:

            return "C"

        return "D"

#instance作成

taro=Student("KeioTaro", 71)  #コンストラクタ実行

print(taro.getNameSco(),taro.getgrade())

hanako=Student("MitaHanako",93)

print(hanako.getNameSco(),hanako.getgrade())

taro.setSco(82)

#instance variableは、このインスタンスのみでの値

print(taro.getNameSco(),taro.getgrade())

print(hanako.getNameSco(),hanako.getgrade())

Student.gradeS=95

Student.gradeA=94

#class variable は全てのinstanceで同じ値になるから変更すると

#全てのinstanceに影響を及ぼす

print(taro.getNameSco(),taro.getgrade())

print(hanako.getNameSco(),hanako.getgrade())

階層構造化の例

class Student:で個々学生をObject化し、class Group:でAおよびBクラスを個別にObject化、class gakunen:で1年生全体をObject化。処理の内容が単純なのにcodeが極端に複雑なので、階層化の良さが分からない例であるが、こんな感じ。

# 個々の学生

class Student:

    gradeS=90  # class variable

    gradeA=80  # classの直下で定義するとclass variable

    gradeB=70

    gradeC=60

    def \_\_init\_\_(self, name, phys): #コンストラクタ

        #コンストラクタはinstance作成時に1回だけ実行される

        self.name = name        #instance variable

        self.phys = phys        #instance variable

    def getNameSco(self):

         return self.name,self.phys

    def setSco(self, phys):

        self.phys = phys

    def getgrade(self):

        if self.phys >= Student.gradeS:

            return "S"

        if self.phys >= Student.gradeA:

            return "A"

        if self.phys >= Student.gradeB:

            return "B"

        if self.phys >= Student.gradeC:

            return "C"

        return "D"

# 医学部1年　A,Bクラス

class Group:

    def \_\_init\_\_(self, groupname,meibo): #コンストラクタ

        self.groupname=groupname

        self.Data=[ ]   # class Studentを入れる予定

        for n in meibo:  #n[0] is name, n[1] is pyhs

            self.Data.append(Student(n[0],n[1]))

    def printData(self):

        for n in range(len(self.Data)):

            print(self.Data[n].name,self.Data[n].phys)

    def getMean(self):

        self.mean=0

        for n in range(len(self.Data)):

            self.mean += self.Data[n].phys

        self.mean /= len(self.Data)

        return self.mean

    def getN(self):

        return len(self.Data)

# A,Bクラスで1学年をまとめる

class gakunen:

    def \_\_init\_\_(self, gakunenname,kurasuname, meibo1nen): #コンストラクタ

        self.gakunenname=gakunenname

        self.Data=[ ]   # class Groupを入れる予定

        for n in range(len(meibo1nen)):  # A,Bクラスなので長さは2

            self.Data.append(Group(kurasuname[n],meibo1nen[n]) )

    def printData(self):

        for n in range(len(self.Data)):

            self.Data[n].printData()

    def printMean(self):

        for n in range(len(self.Data)):

            print(self.Data[n].groupname,  self.Data[n].getMean())

# 名簿などのデータ

meiboA=[["Mitsui",70],["Furuno",75],["Terasawa",81],["Aoki",82],["Kobayashi",90]]

meiboB=[["Amagai",50],["Monkawa",65],["Nakajima",85],["Fukuzawa",52]]

meibo1nen=[meiboA,meiboB]

kurasuName=["Aguni","Bgumi"]

Igaku1nen = gakunen("Keio univ. 1nen", kurasuName,meibo1nen)

Igaku1nen.printData()

Igaku1nen.printMean()