Python 入門

1 Jupyter 上での初めての Python 実行

まずは簡単に Python に入門する。

anaconda prompt から「jupyter notebook」を立ち上げ、画面右上の「New→Text File」でファイルを作成し、出現したウィンドウの左上の「File→Rename」で hellowold.py と名前を付け、以下のコードを書き込んで欲しい。

helloworld.py

```
print("Hello world!")
```

次に、「Home」タブの「New→Terminal」でターミナルを表示し、ここに「python helloworld.py」と入力し、Enter キーを打てば

```
Hello world!
```

と表示されるはずだ。このように、`python <FILENAME>` と統合ターミナルに入力することでファイルを実行することが出来る。今後、上のようにコードが出現したら、それはそのコードを実際に入力して実行してみて欲しい。

2 Python インタプリタの実行

先ほどしてもらったようにファイルにソースコードを入力してターミナルで実行するというのが Python の最も基本的な使い方であり、これからもよく行ってもらうことであるが、初めのうちは Jupyter notebook というツールを使って Python を学習することにする。

統合ターミナルに jupyter notebook と入力することで jupyter notebook が立ち上がる。画面右上の「New→ Python3」を選択することで新しいノートブックが作られる。この段階ではこのノートに名前がついていないので、画面左上の「File→ rename」を選択し、ファイル名をつけて欲しい。今回は初めてのノートなので、「day1」とでもしておこう。

下の入力セルに

```
"Hello wold!"
```
と入力し、Shift+Enter を押すことでセルに入力したコードが評価され、

```
'Hello wold!'
```
と出力されるはずだ。このように、Jupyter Notebook 上ではコードを入力し、Shift+Enter を押すことで即座にコードを評価することが出来る。ただし、最後に評価した値のみが表示されるので注意。

3 Python の基本

3.1 整数の四則演算

`+ - * /` 等で四則演算を行うことが出来る。

まず、「+」によって足し算を行うことが出来る。

```
4 + 7
```
上のように入力し、Shift+Enter を押すことで結果が出力されるはずだ。

また、挙動がわかりにくいと感じた時は以下の例だけでなく様々な例を試してみて欲しい。それでも分からなければ TA を呼ぼう！

3.1.1 課題

先ほどのように jupyter notebook 上で実際に計算を行い、各演算子の Python 上での意味を調べてまとめよ。（「+ - * / % // **」の 7 つの演算子がどのような演算を示すか、一行ずつくらいでまとめれば良い）

3.2 整数以外の計算

Python では整数と小数が明確に区別されており、小数には小数点がついてることに注意して欲しい。

```
0.45 * 0.23
```

また、複素数型も用意されている。数学でしばしば虚数単位は i で表されるが、Python では j で表されることに注意して欲しい。

```
(1 + 1j) * (2 + 1j)
```

3.3 print

print 関数により、計算結果等をまとめて出力することが出来る。

```
print(3 + 3)
print(4 / 2)
print(3 // 2)
print( (1 + j) ** 2)
```

もし print を使わなかったならば、最後のコードの結果のみが表示されることになる。

```
3 + 3
3 / 4
```

3.4 変数

プログラムにおける変数の概念は数学の変数とは少し違った意味合いで使われ、「値を入れておく箱」等と言われる。下のコードでは「x」という変数に「10」という整数を代入している。

```
x = 10
x
```

また、変数は再代入可能である、今「10」を代入した 「x」に小数の「20.0」を代入してみよう。

```
x = 20.0
x
```

また、セルに変数名を入れて Shift+Enter を押すことで変数の確認が出来るので、変数がなんなのか分からなくなった時はすぐに上と同じ方法で確認してみて欲しい。

```
x
```

また、計算の結果を変数に代入することも出来る。

```
x = 10 + 2
x
```

また、変数の再代入に同じ変数を使うことも出来る。

```
x = x + 3
x
```

上の記法を簡単にする記法が用意されている。

```
x += 3
x
```

また、複数の値を同時に代入することも出来る。

```
x, y = 3, 4
```

同じ値をまとめて代入することも出来る。

```
x = y = 5
```

3.5 文字列

ダブルクォーテーション「"」やシングルクォーテーション「'」で囲んだ部分は文字列として扱われる。

```
x = "This is a string"
x
```

このように、変数に代入出来るものは整数や小数等の数値に限らない。また、数値に計算用の演算子が容易されているのと同様に、文字列にもいくつかの演算が用意されている。例えば、文字列同士を足すと、二つの文字列を結合したものになる。

```
"Hi" + " Tom!"
```

また、format を用いることで変数を文字列に埋め込むことが出来る。

```
x = 10
"Tom is {} years old".format(x)
```

3.6 Bool

先程「x = 10」としたが、実際のコードでは「x」に何を代入したのか分からなくなることが多々あるほか、「x」の値に応じて挙動を変えるコードを書くことがある。そのため、値を判定する

「==」は比較を表す。「=」は代入なのに注意。ちなみに「!=」で「==」の逆を表す。他にも「<」,「<=」,「>」,「>=」等の演算子が利用可能である。

```
x = 10
y = 8
print(x == y)
print(x != y)
print(x > y)
print(x < y)
```

3.7 コメント

Python では、「#」以降は改行まで全てコメントとして扱われ、実行されない。

```
#test
```

「#」は行頭においても良いし、途中においても良い。

```
print("test")  # it prints "test"
```

また、コードの直前に「#」を置くことでその行をコメント扱いし、実行されなくすることが出来る。これをコメントアウトと言う。

```
# print("test")
```

3.8 制御文

「if」「for」「while」 を用いて複雑なプログラムを書くことが出来る。これらを制御文と呼ぶ。

3.8.1 if

「if」は分岐を表す。

```
x = 1
if x == 10:
    print("x is 10")
elif x == 20:
    print("x is 20")
else:
    print("x is other")
```

このコードではまず、「x」が 10 であるかどうかをチェックする。

もしも「x」が「10」であればその直下のブロック内の「print("x is 10")」が実行される。

違えば、その下の「elif x == 20」へジャンプする。ここで、「elif」とは「else if」の略である。

ここでも「x」が「20」であるか判定することになる。もしも「x」が「20」であれば「print("x is 20")」が実行される。違えば、最後に「else」の中が実行され、「x is other」と表示される。

3.8.1.1 課題

最初の「x = 1」を別の数字に変えて再度実行して見て欲しい。「x = 10」や「x = 20」の場合を試し、x の値と出力内容の関係を一行程でまとめよ。

3.8.2 for

「for」はループを表す。

```
for i in range(10):
    print(i)
```

ここで「range(10)」は 0~9 の数字を表し、0~9 に関してループを回している。「i」に 0~9 の数字を代入し、「print(i)」を実行する。

「for」文を有効に使うことで多くの数の和等を求めることが簡単になる。

```
x = 0
for i in range(10):
    x += i
print(x)
```

このコードの結果として、「x」は 0~9 の合計となる。

また、for 文は range だけでなく、リスト等に関してもループを回すことが出来る。

```
for i in [10, 20, 1]:
    print(i)
```

3.8.3 while

「while」「for」と同じくループを表す。

```
x = 10
while x > 0:
    print(x)
    x -= 1
print("Finish!")
```

ここで、「while」文の中身は「x > 0」である限り実行され続ける。最初「10」であった「x」をループごとに「print」した後「1」減らす。そして「x」が「0」になったらループを離脱し、「print("Finish!")」に到達する。

「for」「while」は共にループであるが、決まった長さの何かについてループを回したい時は「for」を使い、終了条件が複雑な時は「while」を使うと楽である。

なお、化学プログラミングでは決まった長さの何かについてループを回すことが多いので「for」を多用する。

もちろん制御文を組み合わせて使うことも出来る。

```
for i in range(10):
     if i % 2 != 1:
    print(i)
```

3.9　リスト

　リストとは、値を順番付きで並べたものである。他言語の経験者であれば、配列、Array、Vector 等の名前で聞いたことがあるかも知れない。次のコードは一行一行実行して欲しい。

```
x = [1, 2, 3]
x
```

　これでリストが出来る。リストの要素には以下のようにしてアクセスできる。

```
x[0]
```
```
x[1]
```
```
x[2]
```

　要素は 0 から数える。従って、今回のような長さ 3 のリストの 3 番目にアクセスすることは出来ない。また、負の数を指定することで後ろから数えたアクセスも可能である。

```
x[-1]
```

　下のコードを入力すると、エラーが発生する。

```
x[3]
```

　以下のコードは全て実行後に直下の「x」を実行することで「x」がどう変化したか確認して欲しい。

```
x
```

　リストの要素に代入することが可能である。

```
x[0] = 100
```

　リストの範囲外にアクセスできないのと同様に範囲外に代入することも出来ない。

```
x[3] = 100
```

　リストの要素数を増やしたい時は専用コマンド「append」が用意されている。

```
x.append(20)
```

大きなリストを作りたい時は、「リスト内包表記」を使うと良い。

```
x = [i * 2 for i in range(20)]
```

この表記は以下と等価である。

```
x = []
for i in range(20):
    x.append(i * 2)
```

リスト内包表記では「if」を使うことも出来る。

```
x = [i for i in range(20) if i % 2 == 0]
```

これは以下と等価である。

```
x = []
for i in range(20):
    if i % 2 == 0:
        x.append(i)
```

また、リストへ範囲アクセスすることも出来る。範囲を指定した上で飛び飛びのアクセスも出来る。この挙動を分かりやすくするために、一旦「x」を簡単なリストにしておこう。

```
x = [i for i in range(10)]
```

「start:end」の形を使うことで範囲アクセスできる。この記法により、start 番目以降、end 番目未満の要素が抽出される。

```
x[2:7]
```

また、「start:end:step」の形を使うことで飛び飛びの範囲アクセスが出来る。

```
x[2:7:2]
```

もちろん代入も可能である。

```
x[3:9] = [i * 100 for i in range(6)]
```

また、「for」文を用いてリストに関してループを回すことが出来る。

```
for i in x:
```

```
    print(i)
```

他にも set や dict といった便利な構造や、class という機能もあるが、この授業では使わないため紹介しない。興味のある人は調べてみて欲しい。

3.10 型

「type(x)」のように「type」関数を使うことで変数等の型を調べることが出来る。以下のコードを実行すると「int」と表示されるが、これは「integer」すなわち、「整数」の省略である。

```
x = 1
type(x)
```

以下では「float」となるが、これは「浮動小数点型」という小数のコンピューター上での実現方法の名前であり、「floating point」の省略である。

```
x = 1.0
type(x)
```

「str」は「文字列」を意味する「string」の省略である。

```
x = "x"
type(x)
```

「complex」は複素数のことである。

```
x = 1j
type(x)
```

同じ値でも型によって役割が異なる。例えばリストにアクセスする時、「int」でアクセスすることは出来るが、「float」でアクセスすることは出来ない。今、リストを用意しよう。

```
x = [1, 2, 3]
```

先ほどまで見たように、「1」番目の要素を取得することが出来る。

```
x[1]
```

しかし、「1.0」番目の要素は取得できない。

```
x[1.0]
```

このように型を意識してプログラムを書かないと、予期せぬエラーが起こることがある。（今回は予期出来ているが、実際に書き始めると本当に予期できないので注意）

4 計算機の限界

Python に限らず、計算機では数値を表現するのに有限桁の二進数を用いている。桁が有限であるせいである程度を超えた精度は出ないし、大きすぎる数は表現出来ない。一般に Python で用いられる精度は double precision と呼ばれ、せいぜい 16 桁の精度である。これは計算機にもよるが、多くの場合 8 バイト、すなわち二進数で$8 \times 8 = 64$桁である。

実はフォーマット文字列には桁を指定した表現方法があり、以下のコードを実行することで 17 桁まで表示することが出来る。

```
"{:.17f}".format(0.1)
```

17 桁目は精度が出ないことが分かるだろう。

また、「2E5」と表記することで2×10^5を表すが、

```
2E5
```

この記法を使うとあまりに大きな数は表せないことが容易にわかる。計算機の限界を超えた大きすぎる数は「Infinity」すなわち「無限大」扱いとなる。

```
1E1000
```