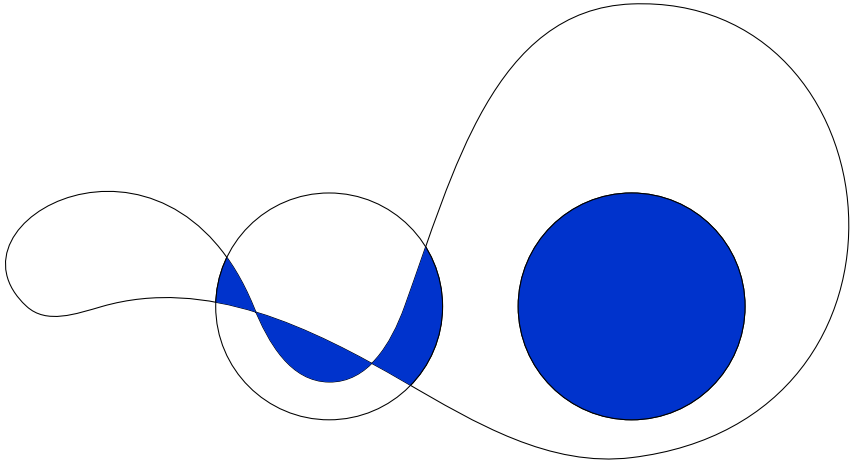

計算機プログラミング

学際科学科　松島慎 2024 年 4 月 11 日

この講義（前半）の進め方

この講義の目標は複雑な問題を解くためのアルゴリズムを実装する力を身につけることである。初回はオンライン講義なので、進め方の説明が主な目標である。

与えられた問題の解決手段としてアルゴリズムを理解する力も重要であるが、理解しているアルゴリズムを実装し（＝動作するプログラムに書き下すこと）実行する力も重要である。プログラミングは自分で試行錯誤をすることで覚えるものなので、この講義では DOMjudge を使って十分な試行回数を確保することを試みる。

DOMjudge は設定した問題の解法となるプログラムを提出（submit）すると自動で採点することができるシステムである。この講義を受講する学生は http://163.220.176.180:9000/ にアクセスし User の登録を済ませること。（Register now? をクリック）

- Username：学生証番号。半角ハイフンあり（例：J3-220465）
- Full name (optional)：氏名。学生証の記述の通り（例：東大太郎）
- Email address (optional)：空欄（連絡は UTOL を通じて行う）
- Team name：任意だが他の人とかぶらないようにする（公開情報となるので注意。例：J3-220465）
- Affiliation：**Use existing affiliation を選択**し、自分の学年科類を選択
- Password：任意（教員に見られても問題ないものを使うこと。**忘れないように!**）

第 0 章

Python の実行環境

0.1 python2 と python3

この講義で扱う言語は Python であるが、Python は Python 3.0 系と Python 2.0 系では大きく仕様が異なるので注意が必要である。この講義では Python 3.0 系を用いたプログラミングを習得する。

0.2 インタラクティブモードと標準モード

Python のプログラムを実行する方法は二通りある。一つは**インタラクティブモード**と呼び、インタプリタがセルと呼ばれる部分ごとにプログラムを処理し、その都度出力を受け取ることを繰り返す方法である。もう一つは**標準モード**と呼びインタプリタがファイルに書かれた文を上から一つずつ処理して最後の文を処理したら終了する方法である。

python3 のプログラムの仕様を理解するためにはインタラクティブモードで実行するのが手軽で便利

である。少し長いプログラムの場合は標準モードで実行すると良い。

0.3　インタラクティブモードの実行環境

インタラクティブモードの実行環境としては Python Notebook があげられる。情報教育棟の iMac を使う場合はすでにインストールされている Anaconda Navigator を使う方法がある。Finder からアプリケーションの下にある Anaconda Navigator を起動して、そこから Jupyter Notebook を選択する。またはターミナル[*1]から以下のコマンドを実行する。ここではプロンプト文字列は$としてあり、$以降が実際に入力する文字列である（今後も実行例については同様に黒字は入力、灰色字は計算機の出力を表す。）

```
$ jupyter-notebook
```

より単純にインタラクティブモードを使う方法として、ターミナルから以下を実行することもできる：

```
$ python3
```

すると以下のような python3 のプロンプト文字列「>>>」が表示されるので、その後に入力される文は python3 のプログラムと解釈されて実行される。例えば 1+2 を入力して Enter キーを押すとこの式の評価結果が出力される

[*1] ターミナルがわからない時はhttps://hwb.ecc.u-tokyo.ac.jp/wp/information-2/cui/をみよ。

インタラクティブモードでの実行例

```
>>> 1+2
3
```

自分の PC が使いたい場合はhttps://utokyo-ipp.github.io/index.html を参考にして Google Colaboratory を使うのが簡単である。Google Colaboratory で Jupyter Notebook が使えて、Python を動かすことができる。または自分の PC に Anaconda をインストールする。次に Anaconda Navigator を起動して、そこから Jupyter Notebook が使用することができる。

0.4　標準モードの実行環境

標準モードではプログラムはテキストファイルとして編集・保存し、この段階ではプログラムは実行されない。単純なテキストエディタとしては Vim, Emacs[*2]の使用を勧めるが統合開発環境として Visual StudioCCode (VSCode) や pyCharm などを使うのも便利である。例えば、`hello.py`[*3] という名前でテキストファイルを作成し、内容を以下のように「`print('Hello World!')`」という一行だけにして保存すれば、これは`Hello World!`と出力するだけのプログラムである。

`hello.py` の内容

```
print('Hello World!')
```

標準モードでは**標準入力**を受けとり、**標準出力**を返すことでプログラムを実行する。おおざっぱには、標準入力とはキーボードから入力された文字列であり標準出力とはスクリーンに現れる文字列である。ターミナル上でプログラム hello.py を実行するには以下のようにコマンドを入力する。

[*2] Emacs の使い方は HWB の説明http://hwb.ecc.u-tokyo.ac.jp/current/literacy/editor/emacs/参照
[*3] Python のプログラムは拡張子を `.py` とするのが慣例である。

標準モード（ターミナル）での実行例

```
$ python3 hello.py
Hello World!
```

この例では標準入力をうけとらず、標準出力にHello World!という文字列を出力する。カレントフォルダに hello.py がないとこのコマンドはエラーとなるので注意すること。

0.4.1　標準入力と標準出力

標準入力を受け取るためには**input 関数**を用いることができる。a=input() をターミナル上から実行するとプログラムはキーボードから入力された文字列を a という変数に格納する。

標準出力を返すには**print 関数**を用いることができる。print("hello") を実行すると引数の値である hello を標準出力に出力し、最後に改行文字を出力する。最後に出力する文字を改行文字以外に指定したい場合はprint("hello",end="XX")とすると、最後に XX を出力する。特に、end=""とすれば改行せずに標準出力のみを出力することができる。

例えば、以下のような関数は標準入力として得られた文字列を 2 回繰り返して出力するプログラムである。

repeat.py の内容

```
a = input() # aに標準入力の内容を文字列として格納する
print(a) # aに格納された内容を標準出力に出力する
print(a) # 再びaに格納された内容を標準出力に出力する
```

これは文字列を入力として受け取り、同じ文字列を二回改行して出力するプログラムである[*4]。この

[*4] 行の中で # 以降は**コメント**であり、インタプリタはこれを処理しない。

内容を repeat.py という名前のファイルに保存してターミナル上で以下のように実行することができる。

標準モードでの repeat.py の実行例

```
$ python repeat.py
aaaa
aaaa
aaaa
```

1 行目がプログラムを実行する命令であり、2 行目がプログラムが入力として受け取る文字列 aaaa である。どちらもキーボードから入力して最後に Enter を押す。3 行目と 4 行目がプログラムの標準出力として出力されたものである。すなわちこれは

入力例

```
aaaa
```

出力例

```
aaaa
aaaa
```

に対応するプログラムの例である。

目次

第 0 章　Python の実行環境 2
0.1　python2 と python3 2
0.2　インタラクティブモードと標準モード 2
0.3　インタラクティブモードの実行環境 3
0.4　標準モードの実行環境 4
0.4.1　標準入力と標準出力 5

第 1 章　基本の文法 8
1.1　条件分岐（if 文） 10
1.2　反復処理（for 文・while 文） 13
1.3　モジュールのインポート 15

第 1 章

基本の文法

変数は値の入れ物である。英字の後に英数字か「_」を並べた**変数名**で指定する。「変数名 = 式」を実行すると、式の値が計算され、指定された変数に**代入**される。代入の効果は、式のなかで「変数名」が出てきたときにその値として使われることである。変数名の中で大文字と小文字は区別される。

インタラクティブモードでの実行例

```
>>>a=3
>>>a*a
9
```

変数には**型**があり、型によって内部で行われる演算も変わってくるので、意識しておくことが必要である場合が多い。型により整数と実数は区別され、変数を初期化する場合は小数のあるなしで整数と実数[*1]が区別される。例えば、`a=1`をあえて`a=1.0`と書くと実数型の 1 となる。変数の型は type 関数で確

[*1] 浮動小数

かめることができる

インタラクティブモードでの実行例

```
>>>a=3
>>>type(a)
<class 'int'>
>>>b=3.0
>>>type(b)
<class 'float'>
```

1.22464679914735e-16は $1.22464679914735 \times 10^{-16}$ の意味である。/は実数の割り算、//は整数の商、%は余り、**は巾乗となる。

実数を整数に変換するには「int(」「)」で囲む。数値を文字列に変換するには「str(」「)」で囲む。一般にデータを、ある「型」に変換するには「型 (」「)」で囲む。”Hello” のように” で囲まれたものは文字列である。” の代わりに’ を使うこともできる。

以下の様に行うと複数の変数を同時に代入することができる。

インタラクティブモードでの実行例

```
>>>a,b,c=1,10,100
>>>a+b+c
111
```

変数名に使える文字はアルファベット以外にもある。例えばひらがなやギリシャ文字なども使用可能である。詳しくはリファレンスを参照のこと。一方、構文で使う for などのキーワード（予約語）は変数名としては使えない。定義されているキーワードは keyword.kwlist に格納されており以下のようにして確認できる。

インタラクティブモードでの動作例

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif',
 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', '
nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

1.1　条件分岐（if 文）

条件分岐は以下の形式で書くことができる。

```
1 if 条件:
2     文1
3     ...
4 else:
5     文2
6     ...
```

この場合まず 1 行目の文にある条件が評価され、真が返ってきた場合文 1 のブロック、偽が返ってきた場合文 2 のブロックが実行される。

条件は以下の書き方がある。

書き方	数学	意味
x > y	$>$	x が y より大きい
x >= y	$\geq$	x が y 以上
x == y	$=$	x と y が等しい (x=y とは書かない)
x < y	$<$	x が y より小さい
x <= y	$\leq$	x が y 以下
x != y	$\neq$	x と y が異なる

真偽値の計算例を示す。

```
print((1 >= 0)) # True
print((0 >= 1)) # False
print((3 != 3)) # False
print(False or True) # True
print(True and False) # False
print(False and True) # False
print(not True) # False
print(not False) # True
```

引数の大きい方の値を返す max 関数を考える。

インタラクティブモードでの実行例

```
>>> x,y = 1,2
>>> if x > y:
...     print("x is larger than y")
...else:
...     print("x is smaller than or equal to y")
...
x is smaller than or equal to y
```

3 つ以上の分岐は if ... elif .. else ... を組合せて実現する。例えば、符号を判定するプログラムは以下のように書ける。

sign.py

```
input_string = input()
x = float(input_string)
if x < 0 :
    print("x is negative")
elif x > 0:
    print("x is positive")
else:
    print("x is zero")
```

標準モード（ターミナル）での実行例

```
$python3 sign.py
23
s is positive
$python3 sign.py
-23
s is negative
```

複雑な条件は簡単な条件を組み合わせて書く。

書き方	意味
条件 1 and 条件 2	条件 1 かつ条件 2
条件 1 or 条件 2	条件 1 または条件 2
not 条件 1	条件 1 ではない

例えば $-1 \leq x \leq 1$ の時は x、それ以外の場合は 0 とするには次のように書くことができる。

```
1 def inbetween(x):
2     if -1 <= x and x <= 1:
3         return x
4     else:
5         return 0
6 print(inbetween(0.5))
7 print(inbetween(1.5))
8 print(inbetween(-1.5))
```

不等式は連結して評価することができる。以下にリファレンスを引用する：

比較はいくらでも連鎖することができます。例えば x < y <= z は x < y and y <= z と等価になります。ただしこの場合、前者では y はただ一度だけ評価される点が異なります (どちらの場合でも、x < y が偽になると z の値はまったく評価されません)。
形式的には、a, b, c, ..., y, z が式で op1, op2, ..., opN が比較演算子である場合、a op1 b op2 c ... y opN z は a op1 b and b op2 c and ... y opN z と等価になります。ただし、前者では各式は多くても一度しか評価されません。
a op1 b op2 c と書いた場合、a から c までの範囲にあるかどうかのテストを指すのではないことに注意してください。例えば x < y > z は (きれいな書き方ではありませんが) 完全に正しい文法です。

1.2　反復処理（for 文・while 文）

1 から 10 までの値に対して何かしたい、というようなことは多い。このような反復処理は for 文を使ってかける。

```
for 変数 in range(開始,終了+1):
    文
    ...
```

「for 変数 in 範囲:」は Python での繰り返しの指定である。for の行末の「:」を忘れることが多いので注意すること。繰り返す部分は**インデント**したブロックである。インデントとは行頭に配置されているスペースあるいはタブのことである。ファイル中で一貫していればスペースの数はいくつでもよい。range の部分に関して 0 から始める場合は 0 を省略できる。

以下は 1 から 10 までの和を計算する例である。

インタラクティブモードでの実行例

```
>>>s = 0
>>>for i in range(1,11):
...     s = s+i
...
>>>print(s)
55
```

for 文のなかに for 文を入れることもできる。この場合インデントの深さにより for の繰り返しの範囲を判断する。

インタラクティブモードでの実行例

```
>>>s = 0
>>>for i in range(1,3):
...     for j in range(1,3):
...         s = s+i*j
...         print(s)
...
1
3
5
9
```

実行は次のように進む。

```
1  i = 1
2  j = 1
3  s = s+1*1
4  j = j+1 # =2
5  s = s+1*2
6  i = i+1 # =2
7  j = 1
8  s = s+2*1
9  j = j+1 # =2
10 s = s+2*2
```

何回くりかえすかはわからないが、ある条件が正となるまである処理を繰り返したい場合もある。こ

のような反復処理は、While 文をもちいて書くことができる。

```
while　条件:
    文
    ...
```

While を用いる反復処理では、反復する文において条件の評価が変わるような処理を行わないと無限ループあるいは何も実行しないプログラムになるので注意が必要である。

1.3　モジュールのインポート

モジュールは意味がまとまったいくつかの関数や変数の定義などの文からなるプログラムであり、必要に応じて自分の実行環境に取り込む（インポートする）ことができる。Python には豊富な標準ライブラリが用意されており、これらのモジュールを適宜インポートすることができる。標準ライブラリの一覧はリファレンスを参照のこと。

```
    import math # math モジュールのインポート
    import copy # copy モジュールのインポート
```

$\sqrt{x}, \sin(x), \cos(x), \log x, \log_2 x, e^x, \pi$ を計算するには、`import math`を実行してから、`math.sqrt(x)`、`math.sin(x)`、`math.cos(x)`、`math.log(x)`、`math.log2(x)`、`math.exp(x)`、`math.pi`を使う。