アルゴリズム入門　＃1

地引　昌弘

2024.10.03

はじめに

皆さんがこれから学ぶ**アルゴリズム** (Algorithm) というテーマについて、まずは意識合わせから始めましょう。アルゴリズムとは、一言で言えば問題を解くための手順を定式化 or 形式化した表現のことです。手順という言葉の意味は、皆さんもよく知っていることでしょう。しかし、定式化 or 形式化という言葉は、少し耳慣れない言葉ですね。ここでは、曖昧さが残る自然言語による表現形式ではなく、数学のような厳密な議論 (例えば式変形など) ができる表現形式だと捉えておきましょう (この用語は、これからもよく出て来るので、意味を覚えておいて下さい)。

さて、何か仕事をするための手順を一度形式化できれば、次はこれを機械にやらせることが可能になります。但し、機械にやらせるとしても、様々な手順毎に専用の機械を用意するより、汎用的な機械を用意した方が効率は良いですよね。様々な仕事 (要は手順) をこなせる機械の一つにコンピュータがあります。コンピュータを利用するには、手順を形式化したアルゴリズムを、さらにコンピュータが理解・実行できる表現に変える必要があります。これを**プログラム** (Program) と言います。

そこで、まずはアルゴリズムの話を始める前に、少しコンピュータについての説明をすることにしましょう。

1　コンピュータの特徴

1.1　アナログとディジタル

コンピュータとは、その名前が示す通り、計算をする機械です。計算は数値を対象に行ないますが、この数値をどう扱うかは、実は少々厄介な問題なのです。

例えば、ここに 1 個のリンゴがあり、その重さを量ったら 300g だったとしましょう。しかし、厳密に言えばピッタリ 300g ではなく、小数点以下に何桁も数字が続くはずです (300.01352684339…という感じ)。これまで皆さんが扱って来た数値体系は、連続で切れ目がありません。つまり、例として挙げたリンゴの重さと同じ状況ですが、数学の世界ではこれを文字で表わすことで計算を定義し、計算することができます。しかし、実際の値を求める場合は、文字ではなく実数値を扱わねばならず、小数点以下何桁まで数字が続くか分からない数値を扱うための仕組みを用意することは、現実的ではありません。そこで、ある決まった桁数を超えた場合は四捨五入して、常に扱う数値の桁数を一定以内にすることにします。例えば、小数点以下は 3 桁まで/ 小数点以上は上位 4 桁まで (243,578 ならば 243,600) といった具合いです。この場合、4.3548 は 4.355 として扱われ、開区間 (4.354, 4.355) にある数値がそのまま扱われることはありません。つまり、離散的な数値体系になっているわけです。以後、この二つの数値体系を次の名称で呼ぶことにします。

アナログ量 (Analog Quantity):
　連続的に変化する値を表わす量。実世界の長さ、重さ、時間、温度、速度、力の強さなどは全てアナログ量です。

ディジタル量 (Digital Quantity):
　離散的に変化する値を表わす量。モノの個数、組み合わせや場合の数など「数えられる」量はディジタル量です。

このような背景のもと、コンピュータとは、非常に突き詰めて言えば、ディジタル量を計算する装置だと言えます。そして、昔であれば「画像ならカメラ」「音声ならレコーダ」のように、扱う対象毎に個別の専用装置を用意する必要がありましたが、コンピュータは「ディジタル量であれば、計算だけに限らず様々な処理を行なえる」という特徴を備えています。

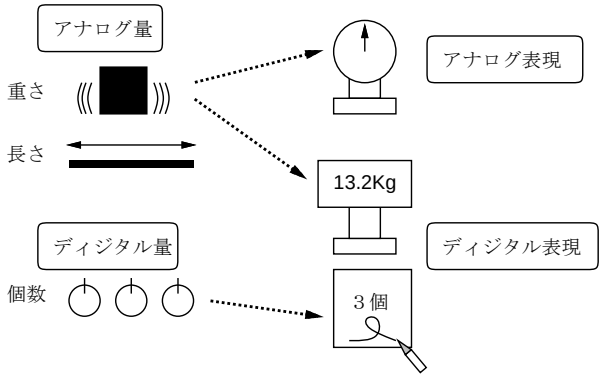

図 1: アナログとディジタル

アナログとディジタルの理解を深めるため、アナログ量やディジタル量の表わし方 (**アナログ表現** (Analog Representation), **ディジタル表現** (Digital Representation)) についても、少し説明しておきます。例えば、アナログ表現の時計や体重計では、針の位置が連続的に変化することで現在の時刻や体重を表わします。一方、ディジタル表現の時計や体重計では、時刻や体重が数字で表わされます (図 1)。現実の世界において数字で “量” を表わす場合、その多くは、何らかの事情 (測定する機器の限界など) に応じて決まる最小単位 or 桁数 (「1 秒」「0.1Kg」など) より細かい部分を省略した「離散的な」値なので、実はディジタル表現を用いていると言えます。

アナログ表現は、ぱっと見ておよそどの程度かが分かり易いという利点があるのに対し、ディジタル表現では数字で表示させるため、値を正確に読み取るのに便利だという利点があります。現実の世界では、値を記録したり伝達するにはアナログ表現よりもディジタル表現の方が優っています。例えば、モノの長さをアナログ表現で厳密に記録するには、紐などに印を付けて覚えることになります。しかし、正確性を保つことや遠くまでその情報を伝達することは、現実的ではありません[1]。これに対しディジタル表現では、物差しで長さを測って数字を書き留めるだけなので、値を再現することは容易であり、また情報の伝達も数字を読み上げるだけで済むため簡単です。但し、ディジタル表現にした時点でその最小単位より細かい情報は失われていることに注意しなければなりません (アナログによる記録とディジタルによる記録の本質的な違いを理解できましたか?)。

以下では簡単のため、「ディジタル表現によって表わされている情報」のことを単に**ディジタル情報**と呼ぶことにします。コンピュータ内部では全ての情報がディジタル表現によって表わされています。これを短く書くと「コンピュータはディジタル情報を扱う」ということになります。

1.2 コンピュータとディジタル情報

ディジタル情報とは、別の見方をすれば「何通りかの場合のどれか」という情報であると言えます。例えば、人の体重を「少数点以下 2 桁までの Kg 単位」で表わすと、「000.00Kg〜999.99Kg」までの 100,000 通りのどれか、という情報だと考えることができます (まあ、体重が 1t 以上ある人はいないでしょうという前提のもとですが)。

これ (「何通りかの場合のどれか」) を突き詰めると、ディジタル情報の最小単位は「二つのうちのどちらか」という情報だと考えることができます (つまり、これより複雑な情報は、最小単位の組み合わせで表現できるというわけです)。これを「0·1 のどちらか」で表わすこととし、「1 **ビット**の情報」と呼びます[2]。例えば、現在の天気を「雨が降っていない」「雨が降っている」の 2 通りに場合分けしたとすると、その情報を例えば次のように 1 ビットの情報として表わす (あるいは対応付ける) ことができます:

ビット表現	意味
0	雨が降っていない
1	雨が降っている

1 ビットはディジタル情報の最小単位ですが、複数のビットを並べた**ビット列**にすることで、より多くの情報を表現できます。例えば、雨が降っている · いないでは大まか過ぎるので、もっと詳しい情報として「晴れ」「曇」「雨」「雪」のどれであるかが知りたいとします。これは、例えば次のように 2 ビットに対応させて表現できます。

[1]計測に用いた紐が変形 · 変質した場合 (紐が伸びてしまった等) は、後から値を正確に再現できません。
[2]ビット (Bit) は「二進表現の 1 桁」(Binary Digit) から来ていますが、「ちょっぴり」という意味の英語でもあります。

ビット表現	意味
00	晴れ
01	曇
10	雨
11	雪

このように、ビット列の長さを 1 増やすと、表わせる場合の数は 2 倍になり、一般に N ビットのビット列では 2^N 通りの場合を表わすことができます。そして、ディジタル情報とは「何通りかの場合のどれか」という情報なので、どんなディジタル情報でも (必要なだけの長さをその都度決めることによって) ビット列で表わせるというわけです[3]。

以上より、コンピュータとは平たく言えば、ビット列を蓄積 · 転送 · 加工するための装置であり、その機能によってあらゆるディジタル情報を取り扱うことができます。さらに、これから実際に見て行くように、人間の介在なしに自動的に処理を行なえる、という点も重要です。

1.3 モデル化とコンピュータ

モデル (Model) とは、何らかの扱いたい対象があるものの、それを直接扱うことが難しい場合に、その対象が備える特定の側面 (扱いたい側面) だけを取り出したものを言います。例えば、プラモデルであれば飛行機や自動車などの「大きさ」「重さ」「機能」などは捨ててしまい、縮小して「形」「色」だけを取り出したもの、と言えます。ファッション モデルであれば、様々な人が服を着るという事象から、その「様々な」の部分を捨て、特定の場面で御洒落に服を見せるもの (“もの” という言い方も少し変ですが) だと言えます。

さて、コンピュータの話なのに、何故モデルの話をしているかと言うと、それは、コンピュータによる処理自体が、ある意味で「モデル」だからです。例えば、「三角形の面積を求める」という計算を考えてみましょう。底辺が 10cm、高さが 8cm であれば

$$\frac{10 \times 8}{2} = 40(cm^2)$$

となり、底辺が 6cm、高さが 5cm であれば

$$\frac{6 \times 5}{2} = 15(cm^2)$$

となります。これを「電卓」で計算するのなら、以下のようにキーを叩きますね:

[1] [0] [×] [8] [÷] [2] [=]

しかし、コンピュータでの計算は、これとはちょっと違います。コンピュータは非常に高速な計算ができるので、いちいち人間が「計算ボタン」を押していたら人間の速度でしか計算が進まず、また間違えなく押せる数値も限りがあるので、あまり嬉しくありません。そこで、コンピュータの利点を活かして高速 · 正確に計算させるために、「どうやって計算をするか」という**手順 · 手続き** (Procedure) を予め用意しておき、実際に計算する時は**データ** (Data) を与えてその手順を実行させる、ということを行ないます。冒頭でも説明しましたが、この手順 (or 手続き) がプログラムです。

これを実現するには、計算の手順とデータを分けることが必要です。例えば面積の計算であれば、手順は

[☆] [×] [◇] [÷] [2] [=]

と書いておき、後で「☆は 10、◇は 8」というデータを与えて一気に計算する、ことになります[4]。これを捉え直すと、「個々の三角形の面積の計算」から「具体的なデータ」を取り除いた「計算のモデル」(この場合では、計算方法だけを取り出したもの) が手順だ、ということになります。このようなモデルを作る際に「不要な側面を捨てる」という作業を、**抽象化** (Abstraction) と言います。具体的な計算を抽象化したものが手順、という言い方をする場合もあります。

コンピュータでの計算を “モデル” と見なす考え方には、もう一つ別の意味もあります。三角形は三つの直線 (正確に言えば線分) から作られますが、現実の世界には完璧な直線など存在しません。例えば、鉛筆で紙の上に引いた線は明らかに「幅」を持っていて、縁はギザギザ曲がっています。また、10cm とか 8cm とか「ちょうどぴったり」の長さも世

[3] 興味のある人のために、もう少し深い話をすると、既に知っていることを再度伝えられても全く嬉しくないし、また、知らないことであっても「ほとんど雨が降らない地方の天気」であれば、「雨が降っていない」という知らせには、やはり新たな価値がほとんどないと言えます。ビット列で表わした情報にどのくらいの価値があるかという観点から**情報量**を測る研究分野があります。

[4] もちろん、「☆は 6、◇は 5」とすれば別の三角形の計算ができます。

の中には存在しません。コンピュータを使った実際の計算では、このような細かい事情を捨てて、「理想的な三角形」に抽象化してその面積を計算しているわけです。逆に言えば、コンピュータで計算する時は常に、現実世界のモノをそのまま扱うわけではなく、必要な部分だけをモデルとして取り出し、それを計算している、ということになります。皆さんはこれまで、このような抽象化やモデル化に対し、数学の一環として多く接して来たと思いますが、これからはコンピュータでプログラムを扱うという観点から考えてみて下さい。

2 アルゴリズムの特徴

2.1 アルゴリズムとその記述方法

前節における「三角形の面積の計算方法」のような、計算 (や情報の加工) の手順のことを**アルゴリズム** (Algorithm) と言います。そして、アルゴリズムをコンピュータで実行できる表現にしたものを**プログラム** (Program) と言います。ある手順がアルゴリズムであるためには、次の条件を満たす必要があります。

- 有限の記述で構成される。
- 手順の各段階は曖昧さを排除した厳密な表現で記述される (このような表現を形式化 · 定式化と呼びます)。
- 実行すると必ず停止して答えを出す。

これらの条件が必要な理由を簡単に説明しておきます。1 番目は、「無限に長い」記述だと書くこともコンピュータに読み込ませることも不可能だからです。2 番目は、曖昧さがあると、それをコンピュータで実行させられないからです。ここでは、コンピュータによる云々という言い方をしていますが、別の言い方をすれば、この 2 条件を満たしていないとアルゴリズムの正当性を判断できないということです。ここでいう正当性は、同じデータを渡せば必ず同じ結果になることを意味しています。同じデータを渡したにも関わらず毎回違う結果になってしまう手順は、アルゴリズムとは言えませんね。3 番目はどうでしょうか。この条件を入れるかどうかについては、様々な意見があります。しかし、例えば「永遠に答えが出ないかも知れないが、答えが出た時は億万長者になる方法を教える手順を発見した」と主張された場合を考えてみましょう。その手順を実行したものの、いつまでも止まらないのであれば、やはり (上の説明と同じく) その主張が正しいかどうか確かめようがありません。つまり、停止することを条件にしておかないと、アルゴリズムの正当性について論じることが難しくなるという考え方です。この講義では、停止性を条件に入れておくことにします。

参考)
“与えられたプログラムが有限時間に停止するかどうかを判断できるプログラムは存在しない” ことが証明されています。

略証: プログラム A とデータ x に対し、A(x) が (つまり、プログラム A にデータ x を入力した場合に A が) 有限時間で停止するかどうかを判定できるプログラム H(A, x) が存在したと仮定する。

A(x) は停止する ⇔ H(A, x) は YES を出力

A(x) は停止しない ⇔ H(A, x) は NO を出力

H に対してデータ x の代わりにプログラム A を渡し、次のように動くプログラム S [5]を作る。

H(A, A) = YES ⇔ S(A) は停止しない。

H(A, A) = NO ⇔ S(A) は停止する。

ここで、S(S) を考えると、その挙動は下記のどちらかとなる。

S(S) が停止しない場合、H(S, S) = YES となるが、H の定義より S(S) は停止する。

S(S) が停止する場合、H(S, S) = NO となるが、H の定義より S(S) は停止しない。

どちらの場合も矛盾が生じるので、H(A, x) は存在しない。

[5]プログラム S の例としては、H(A, A) = NO であれば終了し、H(A, A) = YES であれば再度 H(A, A) を実行する (つまり、H(A, A) = YES である限り H(A, A) を再実行し続けるので、結果として停止しない) ようなプログラムがあります。

アルゴリズムを考えたり検討するためには、それを何らかの方法で記述する必要があります。その記述方法としては様々なものがありますが、ここでは手順や枝分かれ等をステップに分けて日本語で記述する、**擬似コード** (Pseudo Code) と呼ばれる方法を使います。**コード** (Code) とは「プログラムの断片」という意味で、「擬似」というのは実際のプログラミング言語ではなく、日本語を使うから、と考えておいて下さい。三角形の面積計算のアルゴリズムを擬似コードで書いてみます:

- triarea: 底辺 w、高さ h の三角形の面積を返す[6]
- s ← $\frac{w \times h}{2}$
- 面積 s を返す。

2.2 変数と代入 · 手続き型計算モデル

上のアルゴリズム中で次の部分をもう少しよく考えてみましょう:

- s ← $\frac{w \times h}{2}$

この「←」は**代入** (Assignment) を表わします。代入とは、右辺の**式** (Expression) [7]で表わされた値を計算し、その結果を左辺に書かれている**変数** (Variable — コンピュータ内部においてデータの記憶場所を表わすもの) に「格納する」「しまう」ことを言います。つまり、「w と h を掛けて、2 で割って、結果を s が示す場所に書き込む」という**動作** (Action) を表わしていて、数学で扱う数式のような定性的 (項間の関係を表わす) 記述とは別物だと考えて下さい。数学の数式であれば、$s = \frac{w \times h}{2}$ ならば $h = \frac{2s}{w}$ のように変形できるわけですが、アルゴリズムの場合では、式は「この順番で計算する」ことを意味し、代入は「結果をここに書き込む」ことを意味するので、数式のような変形はできない点に注意して下さい。なぜ、このような注意を述べるかと言うと、困ったことに多くのプログラミング言語では、代入を表わすのに文字「=」を使うため、普通の (数学的な) 数式であるかのような混乱を招き易いからです。

これをモデルという立場から見てみると、式は「コンピュータ内の演算回路による演算」を抽象化したもの、変数は「コンピュータ内部の**主記憶** (Main Storage) ないし**メモリ** (Memory) 上のデータ格納場所」を抽象化したもの、そして代入は「格納場所へのデータの格納動作」を抽象化したもの、と考えることができます。このような、式による演算とその結果を変数へ代入することによって計算が進んで行くようなモデルを**手続き型計算モデル** (Procedural Computational Model) と呼び、そのようなモデルに基づくプログラミング言語を**命令型言語** (Imperative Language) ないし**手続き型言語** (Procedural Language) と呼びます。手続き型計算モデルは、上述のように現在のコンピュータとその動作をそのまま素直に抽象化したものになっています。このため手続き型計算モデルは、最も古くから存在し、また現在のところ最も広く使われている計算モデルです。コンピュータによる計算を表わすモデルには、手続き型計算モデルの他に、関数とその評価に土台を置く関数型モデルや、論理に土台を置く論理型モデルなどがあります。

3 アルゴリズムとプログラミング言語

3.1 プログラミング言語

プログラムとは、アルゴリズムを実際にコンピュータへ与えられる形で表現したものであり、その具体的な「書き表わし方」ないし「規則」のことを**プログラミング言語** (Programming Language) と呼びます。これはちょうど、人間が会話をする時の「話し方」として「日本語」「英語」など様々な言語があるのと同様です。但し、**自然言語** (Natural Language — 日本語や英語など、人間同士が会話したり文章を書くのに使う言語) とは違って、プログラミング言語はあくまでもコンピュータに読み込ませて処理することが前提の人工的な言語であり、そのため書き方も杓子定規 (良く言えば、曖昧性を排除した厳密な表現) だと言えます。

実際のプログラミング言語には、様々な特徴を持つ多くの種類が存在します。この講義では、プログラムが簡潔に書けて、また簡単に試せるという特徴を備えた **Python** という言語を用います。

---

[6]以下ではこのように、何を受け取って何を行なう手順 (アルゴリズム) かを明示するようにします。上の例で「返す」というのは、底辺と高さを渡されて計算を開始し、求まった結果 (面積) を渡された場所で渡した相手に (答えとして) 引き渡す、と考えて下さい。

[7]プログラミングで言う式とは、数値などの具体的な計算方法を**数式** (Mathematical Expression) に似た形で記述したものを言います。

3.2 Python 言語による記述

では、三角形の面積計算アルゴリズムを Python プログラムに直してみましょう。コンピュータの内部で、入力された Python プログラムやデータを実行して結果を出力するものを、Python の処理系と呼びます。この講義では、人間と Python 処理系との入力や出力は、基本的に **IDLE コマンド** (IDLE Command) [8]の機能を利用することとし、皆さんは、計算部分だけを Python の**関数** (Function) [9]として書くことにします。先ほどアルゴリズムを示した三角形の面積計算を行なう関数は、次のようになります:

```
def triarea(w, h):
  s = (w * h) / 2.0
  return(s)
```

Python では、関数定義は複合文に分類されます。複合文の意味・書き方については、“Python によるプログラミングの初歩” にある “文” を参照して下さい。ここでは、関数定義に焦点を当ててその詳細を説明しましょう。

1. 「`def` 関数名:」+「ブロック」の範囲が一つの関数定義になる。
2. 関数名の後に丸括弧 “(” および “)” で囲まれた名前の並びがある場合、それらは関数に引き渡す**パラメタ** (Parameter)[10]名となる。パラメタは、**引数 (ひきすう)**(Argument) とも呼ばれる。関数を呼び出す時、これらの引数に対応する値を指定する。
3. 関数内には**文** (Statement)[11]が何個あってもよい。この文の集合が関数の本体 (つまり、ブロック) となる。
4. 式は原則として先頭から順に一つずつ実行される。
5. **return 文** (Return Statement)「`return(式)`」を実行すると、関数の実行は終わり、その “式” の値が関数の値として呼出し元・実行者へ返される。

上の例では擬似コードに合わせるように、面積の計算結果を変数 `s` に入れてからそれを `return` していましたが、`return` の後ろに計算式を直接書くこともできるので、次のようにしても同じです:

```
def triarea(w, h):
  return((w * h) / 2.0)
```

このように、プログラミング言語とは、コンピュータに対して実際にアルゴリズムの実行を指示するように決めた記述形式なのです。プログラムのどこか少しでも変更すると、コンピュータの動作もそれに相応して変わるか、そんな変更はできないと怒られることになります。

3.3 動かしてみよう!

では、このコードを動かしてみましょう。まず、エディタで上と同じ内容を打ち込み、`sample1.py` というファイル名で保存して下さい。人間が打ち込んだプログラムを (プログラムを動かす「源」という意味で) 「**ソース**」or「**ソース コード**」などと呼びます。Python のソース ファイルは、最後を「`.py`」にするというのが通例です。ファイル形式は「プレーン テキスト」(アプリ固有の形式ではない一般的な形式) で保存して下さい。

エディタとして、Mac OS にある「TextEdit」あるいは「テキスト エディット」を利用する場合、以下の注意があります。

> **重要!** テキスト エディットを使う場合は、「フォーマット」メニューで「標準テキストにする」を選んでからファイルを保存して下さい。標準テキスト以外で保存した場合、`IDLE` はそのファイルを正しく読み込めません。また、「テキスト エディット」メニューの「環境設定」を開いて、下の方にあるオプションを全部オフにして下さい。特に、「スマート引用符」がオンになっていると、「’」などの引用符が勝手に別の文字に化けて困ることになります。

[8] Python の処理系に備わっているコマンドの一つで、様々な値をキーボードから入力し、それを用いてプログラムを動かす機能を提供してくれます。

[9] 関数とは、他のプログラミング言語における**手続き**ないし**サブルーチン** (Subroutine) に相当し、一連の処理に名前を付けたもののことです。なお、英語の Procedure も、日本語では一般に手順 or 手続きと訳されますが、プログラミングの観点からは、手順と言う場合は抽象的な (プログラムとして書き下す前の) ものを指し、手続きと言う場合はプログラムに含まれる名前の付いた一連のコードを指すという具合に使い分けています。

[10] 計算の手順とデータを分けるため (計算のモデル化 or 抽象化を行なうため)、関数を使用する度に毎回異なる値を引き渡し、それを用いて関数に処理を行わせるための仕組みです。

[11] プログラム内にある個々の命令のことを、プログラミング言語の用語では “文” と呼びます。

次に、MacOS 内のターミナルを開き、その中から “**idle**” あるいは “**idle3**” コマンドを実行し、Python 処理系を起動して下さい (「%」はプロンプト文字列のつもりなので打ち込まないで下さい):

```
% idle3
(新しいウィンドウが開かれる)
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

この「>>>」というのは、IDLE の**プロンプト** (Prompt — 入力をどうぞ、という意味の表示) で、対話モードでは、この状態で Python のコードを打ち込みます。また、教育用計算機システムの環境では、“**idle**” あるいは “**idle3**” コマンドの代わりに “**IDLE**” コマンドを実行すると Python の古いバージョンに対応した IDLE が起動されてしまいます。必ず “**idle**” あるいは “**idle3**” コマンドを実行して下さい。

最後は、IDLE から作成したプログラム ファイル (sample1.py) を読み込みます。具体的な手順は、“Python によるプログラミングの初歩” にある “IDLE のファイル モード” を参照して下さい。

ところで、何故 3〜4 行程度の内容をわざわざ別のファイルに保存して、、、という面倒なことをしているのでしょうか? それは、関数定義の中に間違いがあった場合、定義を毎回 IDLE に向かって打ち直すのでは大変過ぎるからです。このため以後は、関数定義はファイルに保存して必要に応じて修正することとし、IDLE ではファイル モードにより関数を呼び出して実行する部分だけを行なう、という分担にします。

sample1.py の読み込みに成功したら `triarea` が使えるはずなので、それを実行します:

```
>>> triarea(8, 5)
20.0
>>> triarea(7, 3)
10.5
>>>
```

確かに実行できているようです。IDLE は `quit()` で終了させます:

```
>>> quit()
(IDLE のウィンドウが閉じる)
%
```

演習 1-1 例題で示した三角形の面積計算関数を対話モードで作成し、実行させてみよ。実行時に数字でないものを与えるとどうなるかも試せ。

演習 1-2 三角形の面積計算において、除数の指定を「2.0」(実数) でなく、ただの「2」(整数) にした場合に何か違いがあるか試せ。

演習 1-3 次のような計算をする関数を作成し、実行せよ[12]。

a. 二つの実数を与え、その和を返す関数を作成せよ。和ができたら、差 · 商 · 積についても作成せよ。四則演算が完成したら、これら全てを行ない、結果を表示する一つの関数を作成せよ。
b. 「%」という演算子は、**剰余** (Reminder) を求める演算である。二つの整数を与え、その剰余を返す関数を作成せよ。次に、様々な値を用いて剰余を計算してみよ。何か気付いたことはあるか。
c. 円錐の底面の半径と高さを与え、体積を返す関数を作成せよ。
d. 実数 x を与え、x を 10.0 で割った結果を返す。同様に x の 0.1 倍を返す。表示される計算結果の桁数を変えながらこれらを比較し、何か気付いたことはあるか。

[12] 一つのファイルには関数定義 (`def: ...`) を幾つ入れても構わないので、ファイルが長くなり過ぎない範囲でまとめて入れておいた方が扱い易いです。その際は、“Python によるプログラミングの初歩” にある “コーディング規約” に沿った記述を心掛けて下さい。

3.4 数値の表示に関する補足

演習 1-3d に取り組む場合は、数値を表示する時に十分な桁数がないと細かい違いが分からないので、そのための出力命令を説明しておきます。Python では先の例のように、特に指定しないと桁数などは「おまかせ」になりますが、これを自分で制御したい場合は、下記のように、`print` 関数を利用する際に “%” を用いて表示形式を指定します[13]:

```
print("書式化文字列" % ((値 1), (値 2), ...))
```

print 関数は「書式化文字列 (Format Specifiers)」を出力します。但し、その中に**書式** (Format) が指定されていた場合は、該当する箇所に対応する「値」を書式に従い文字列に変換してから出力します。書式化文字列と各値との間は “%” で区切ります。主な書式は以下の通りです (ここでも%が使われているので、混乱しそうですね)。

%d: 符号付き十進表示

%o: 符号付き八進表示

%x: 符号付き十六進表示

%e: 浮動小数点数の指数表示

%f: 浮動小数点数の十進表示

%g: 浮動小数点数を最適と想定される形式で表示

%s: 文字列

特に「`%.N` □」という書式は (□の部分には、上の表示形式のどれかが入る)、数値を有効数字 N 桁で表示する指定です。また、“\n” は、改行を意味します。簡単な例題として、渡された数およびその 3 分の 1 を 10 桁表示するプログラムを掲載しておきます。ここでは、同時に複数の値を表示させています。値の部分の書き方に留意して下さい (値の部分が式でない場合は、括弧は不要です):

```
def onethird(x):
  print("%g, %.10g\n" % (x, (x/3.0)))
```

実行結果は、こんな感じ[14]。

```
>>> onethird(1.0)
1, 0.3333333333


>>>
```

ところで、桁数を色々変えて調べたい場合に、毎回関数を作り直すのはちょっと面倒です。実は IDLE 上で直接 `print` 命令を実行させることもできるので、その様子を示しておきます:

```
>>> print("%.5g" % (1.0/3.0))
0.33333
>>> print("%.15g" % (1.0/3.0))
0.333333333333333
```

[13] Python には、他にも表示形式を指定して出力する方法がありますが、`print` 関数を利用する方法は他のプログラミング言語でも広く使われているので、ここではこれを紹介します。

[14] 書式化文字列に改行の指定を入れているので、空行が一つ表示されています。多くの言語では、表示を改行する際に改行の指定が必要ですが、Python では、print 関数毎に自動的に改行が入ります。よって、この講義で取り上げるプログラムの範囲であれば、改行の指定が必要になる場合はあまりありません。