スクリプト言語プログラミング
Pythonによる数値解析

第２回講義資料

箕原辰夫

空白とコメント

- Pythonでは、左側に空白が空いているのがアウトラインのレベルを示す
  - TABキーを使う, Deleteで戻せる
- インデントがあっていないと動作しない

- # この後改行するまでがコメント
- """ """ 囲まれた範囲がコメント

Pythonの記号

記号	呼び方	意味
;	セミコロン	文の区切りを示す
.	ドット	所属を示す「～の～」
,	カンマ	羅列を示す「～と～」
" '	クォテーション	文字列を示す
#	ナンバーマーク	行の終わりまでコメント
"""	３つのクォート	囲まれた範囲がコメント
*	アスタリスク	「すべて」または「掛け算」
:	コロン	制御構文であることを示す
@	アットマーク	アノテーション（実行時の指定）を示す

プログラムは式を記述していく

- 式の構成要素
  - 変数
  - 定数（リテラル）
  - 演算子

- 定数は、値のこと
  - 一定の値を取り続けるということで、定数(constant)と呼ばれる
  - プログラミング言語では、定数が型を持つ

値の型

- Java/C/C++/C#では、値の型が厳格に参照される。
  - 厳格な型言語（Strict type languages）
- Python/JavaScript/Rubyなどでは、結果的に値の型が決まっていく
  - 型推論（Type Inference）と呼ばれる
- プログラム上で値を記述できる型（primitive type)には次のようなものがある
  - 論理型
  - 整数型
  - 実数型
  - 複素数型
  - 文字列型
- プログラム上で記述された値（型を持つ）をリテラル（literal）と呼ぶ

論理型

- 条件が満足されたかどうかを示す真偽値。
- **bool**型（ブール代数から）と呼ばれる。

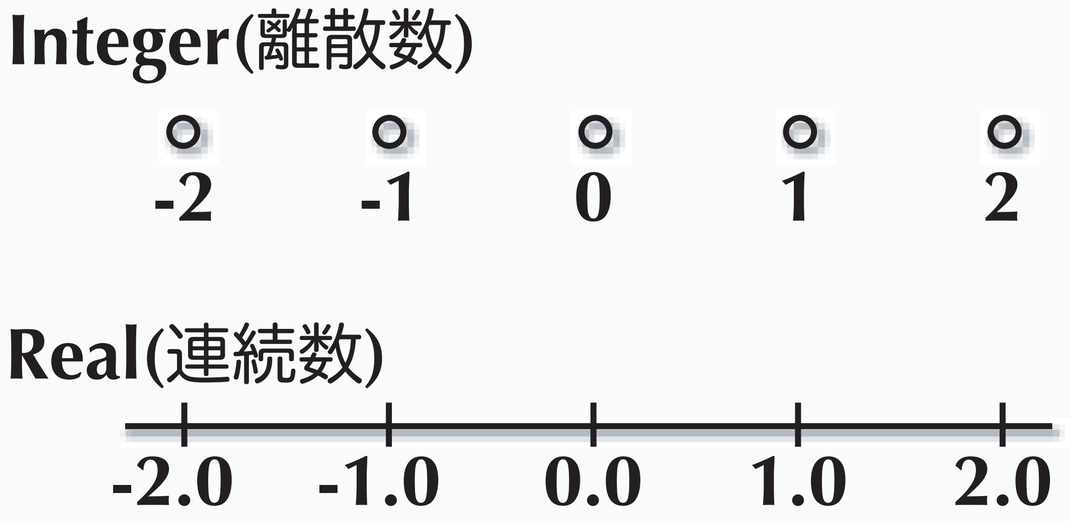
- 値としては、次の２つだけになっている。
  - **True**…条件を満足した（真）
  - **False**…条件を満足しない（偽）

特別な値

- **None** … オブジェクトを差していない特別な値
- クラスは、NoneTypeという型になっている

整数型と実数型

- 整数は、離散数（Discrete Number）と呼ばれ、実数は連続数（Linear Number）と呼ばれている。



整数型

- 整数型の型名は、int
- Pythonでは、整数の桁数の制限はない（C/C++/Java/C#などの他の言語では、42億ぐらいまでとか、922京ぐらいまでの制限がある）
- 整数の例：
  - 7　2147483647 3
  - 79228162514264337593543950336
  - 100_000_000_000 (Python 3.6から)

8進数、2進数の整数と16進数の整数

- 8進数の整数
  - 0oを先頭につける(0から7までしか使えない)
  - 0o262730
- 16進数の整数
  - 0xを先頭につける
  - 0x45a
- ２進数の整数
  - 0bを先頭につける
  - 0b01010101

実数型

- 浮動小数点数で表現される
- Pythonでの型名としては、float
- 小数点をつけると自動的に実数型として認識される
  - 2.3　4. → 4.0　.05 → 0.05
- 指数表記が可能
  - 2.45×10^{16} → 2.45e16 あるいは 2.45e+16
- 実数の例：
  - 3.14　10.　.001　1e100　3.14e-10　0e0　3.14_15_93 (Python 3.6から)

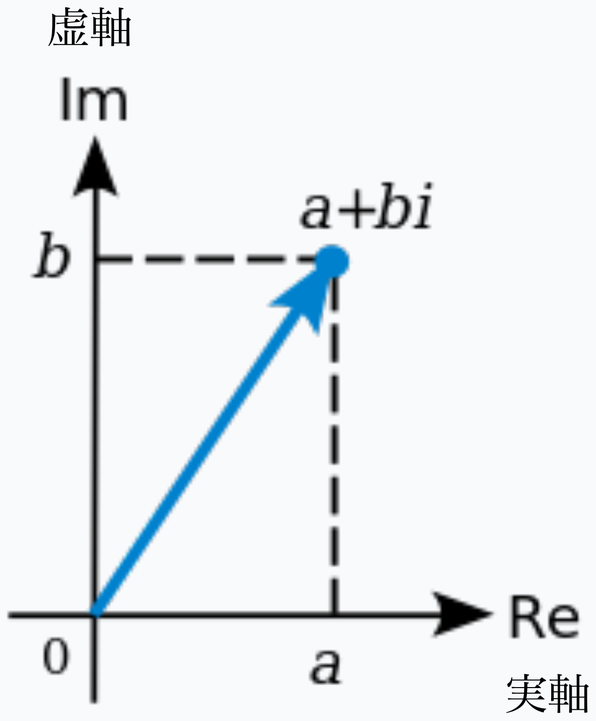
複素数型

- 虚数（数学ではiを使うが、工学系やPythonではjを用いる）

$$i \times i = -1 \qquad \sqrt{-1} = i$$

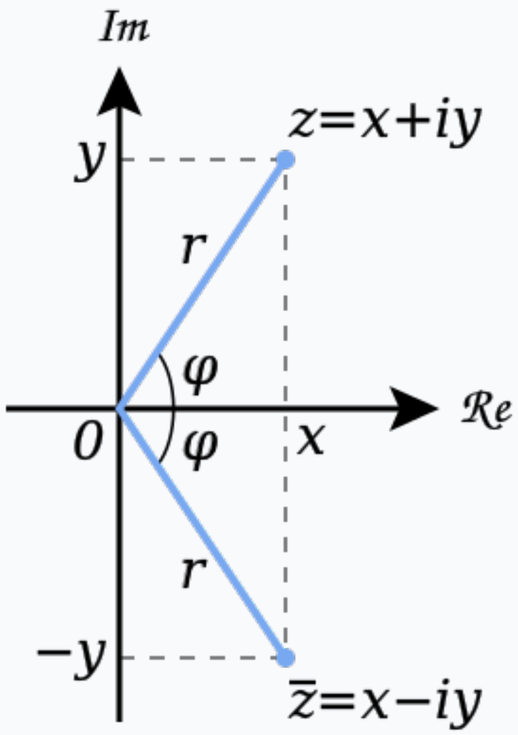
- 虚数値の例：
  - 1j　3.14j　10.j　10j　.001j　1e100j
  - 3.14e-10j　3.14_15_93j

- 実数部と虚数部を加減算することで複素数を表現することができる
- 型名は、complex
- 複素数の値の例：
  - 12+4j　3.5-5.2j　2+5j



複素数平面（Complex plane）

- 複素数平面は、ガウス平面（Gaussian plane）とも呼ばれる
- 共役複素数は、conjugateを使って求めることができる
  - 例：(1+1j).conjugate( )
- 実数部を取り出すのは、.real
  虚数部を取り出すのは、.imag
  を用いる
  - 例：(1+0.5j).real　(3+4j).imag



複素数とオブジェクト

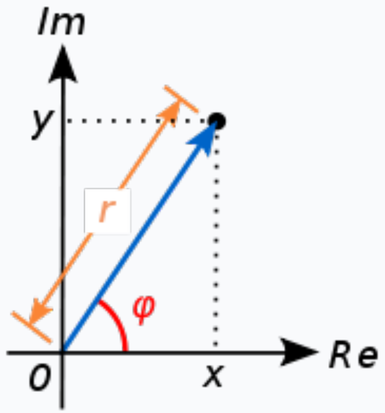
- 複素数はオブジェクトになっている（実は整数や実数もオブジェクト）
- オブジェクトは、そのオブジェクト固有の属性（attribute, フィールドfieldとも呼ぶ）を持っており、変数としてアクセスできる
- オブジェクトは、そのオブジェクト固有の関数（メソッドmethodと呼ばれる）を持っており、そのオブジェクトを介して呼び出すことができる
- 属性とメソッドを合わせて、プロパティ（property）と呼ぶことがある

Pythonでのオブジェクトのプロパティの標記法

- 属性の場合
  - オブジェクト.属性名
  - 例：(3+4j).real…3.0が返される
  - 例：(3+4j).imag…4.0が返される
- メソッドの場合
  - オブジェクト.メソッド名(パラメータ)
  - パラメータが必要ないときは括弧内には何も書かない
  - 例：(3+4j).conjugate( )…3-4jが返される

複素数のためのライブラリ

- abs(複素数)…複素数のノルム（normあるいは絶対値、大きさrのこと）を求める
- **import** cmathを入力しておく
- cmath.phase(複素数)…偏角φをラジアン角度で返す
- cmath.polar(複素数)…ノルムと偏角の対を返す
- cmath.rect(ノルム, 偏角)…ノルムと偏角から構成される複素数を返す
- ノルムと偏角で表される複素数の形式を「極形式」(polar form)と呼ぶ
- 極形式については、オイラーの公式が有名



$$z = r\left(\cos\varphi + j\sin\varphi\right) = re^{j\varphi}$$

文字列型

- 文字はすべてUnicodeで符号化されている。
- ファイルなどを読み込むときに、別の符号（JISやShift JIS）を用いるときは、符号（エンコーディング：encoding）の指定をしなければならない。

- 文字列を扱う文字列はstrクラスになっている
- Pythonでは、どちらかの引用符を用いる
  - 値を記述するときは一重引用符で囲む 'a'
  - 値を記述するときは二重引用符で囲む "a"

Unicodeによる文字列

- strクラスという形で定義されている
- \nは改行、\tは水平タブ
- \" はダブルクォーテーション、\'はシングル
- \\ はバックスラッシュ自身を表わす
- \uと16進数4桁で、Unicodeの文字を指定できる。
  - "\u4e00" → "一"
- \Uと16進数8桁で、拡張された文字領域の文字を指定できる。
  - "\U00013000" → "𓀀"
- パレットでUnicodeのコード表を参照
  - Mac OS スクリプトメニューの「絵文字と記号を表示」→左上のリストをカスタマイズで、「Unicode」を追加
  - Windows IMEパッドの文字カテゴリでUnicodeを選ぶ（基本言語面は、4桁で指定する部分、多言語面は、8桁で指定する部分）

Unicodeと組込み関数

- 1文字に対して
  - ord(1文字の文字列)…文字に対応するUnicodeのコードを整数として返してくれる
    - 例：ord("🏂")→77848　　ord("A")→65　　ord("漢")→28450
  - chr(整数)…Unicodeの整数に対応する1文字の文字列を返してくれる
    - 例：chr(105)→'i'　　chr(0x3fe9)→'𪾩'　　chr(0x103b5)→'𐎵'
- 文字列に対して
  - ascii(文字列)…ascii文字はそのまま、それ以外の文字については、文字列中の各文字に対して対応するUnicodeの16進数を「文字列リテラル」として返してくれる。もちろん、1文字の文字列でも使用可能
    - 例：ascii("ABCあい🏂")→'ABC\\u3042\\u3044\\U00013018'
    - 例：ascii("鮪") → '\\u9baa'　　ascii("🏄") → '\\U00013009'

変数の名前と代入

- 変数の名前
  - 変数の名前に使えるのは、半角の英数字、およびアンダーバー（ _ ）
  - 名前の先頭の文字は英字でなければならない
  - 変数名は、半角の小文字の英字でつけるように心掛けたい

  例：

  x y z one two na99 take_a_cup

- 代入
  - 変数名 = 式

  例：　$x = 10$

変数の型宣言

- 変数の型を予め宣言しておくことができる（Python 3.7より）
- 変数の型宣言は、代入より前か、初期値を代入するときに行なう
- 書式
  - 変数名 : 型名
  - 変数名：型名 = 初期値の式
- 例：
  - x : str
  - y : float
  - z : complex = 4j
  - i : int = 7

変数の参照

- 参照
  - 変数が保持する値に置き換えられる。
  - 変数が保持するオブジェクトが参照される。
  - 例：

```
x = 8
print( x * 12 )   # 8 * 12に置き換えられる
```

- 自己参照代入
  - 元の値を利用して、新しい値が代入される。

$x = x + 1$

$x = -x$

$x = x - 20$

代入としての記号 =

- 右辺と左辺は同一ではない
- 等しいという意味ではなく、右辺の式を評価して左辺にAssignするもの。

左辺 = 右辺

  - この意味は、「左辺の変数　←　右辺の評価値」

- なお、左辺の値が評価値として残る（右結合性）

$$x = y = z = 0 \quad \rightarrow \quad (\, x := (\, y := (\, z := 0\,)\,)\,)$$

式に何が書けるか

- 式の定義
  - 定数
  - 変数名
  - 式 + 式　←加算　add
  - 式 – 式　←減算　subtract
  - 式 * 式　←乗算　multiply
  - 式 / 式　←実数除算 true divide
  - 式 // 式　←整数除算 floor divide
  - 式 % 式　←剰余　modulo
  - 式 ** 式　←べき乗 power
  - 変数名 := 式　←代入式 assign expression
  - (式)

式の構文

- ４５＊（３４＋２３）／（ｙ－５）
- 式＊（式　+　式）／（式–式）
- 式＊（式）　／　（式）
- 式＊式　／　式
- 式　／　式
- 式

- ×　45x + 65y

代入演算子 :=

- Python 3.8からの導入で、式の中に演算子として代入演算子を指定することができる
- = は、代入文を形成するので、左辺に代入する変数、右辺に式を記述するが、:=は式の中に代入式を記述することができる
- 例：

```
print( n := 10, f"binary value of {n} is {n:b}" )
```

- :=は、優先度が低いので、場合によっては、()をつけて優先度を高くする必要がある
- また、関数呼出しの実引数の部分以外では「(変数名 := 式)」という形で、丸括弧をつけてあげないと文法エラーになる
- 例：

```
t = ( n := 12, m := 13 )
```

式と評価

- 評価（Evaluation）とは
  - 単一の値になるまで計算すること
- 式の書式に合っているか
  - 書式に合っていないと文法エラー
- 式の評価の優先順位
  - 優先順位がある
  - 単項の±は一番優先される
  - べき乗の演算子（**）の優先度は高い
  - 乗除算の演算子（* / // %)の方が優先される
  - 加減算の演算子（+ -)が優先度低い
  - 代入演算子:= は優先度が一番低い
  - （）で囲むと優先度を高くする

結合性

- 同じ優先順位の演算子は、左から評価されていく（加減乗除などの場合）
  - 左結合性（Left associative）と呼ぶ

56 * 34 / 28 * 60 / 30 % 89

→((((56 * 34) / 28) * 60) / 30) % 89

83 + 45 - 23 + 38

→((83 + 45) - 23) + 3

整数演算

- 整数除算は、小数点以下が切り捨てられる
  - 5 // 2 → 2
  - 1 // 8 → 0　分母の方が大きいと0になる
- どこに整数除算があるか重要
  - 5 // 2 * 2 → 4
  - 5 * 2 // 2 → 5
- 剰余算は、余りを計算する（実数でも可）
  - 365 % 20 → 5
  - x % n → 0 〜 n-1の数しか出てこない

    計算結果が0になるときは、割り切れるということ

整数剰余・整数除算

- x // n * n
  - xと等しいか、xを超えない最大の数で、nで割り切れる数が求まる
  - 例： 10 // 3 * 3 → 9

- n // m
  - n < mの場合は、0になる
  - 例： 3 // 4 → 0

- n % m
  - n < mの場合は、nになる

整数除算の計算方法

- x % n → x – (x // n * n)
- x // n → (x – x % n) // n

- x // n * n ≠ xのときがある
  - 10 // 3 * 3 → 9
  - 11 // 3 * 3 → 9
  - 12 // 3 * 3 → 12
  - 13 // 3 * 3 → 12
  - 14 // 3 * 3 → 12
  - 15 // 3 * 3 → 15

基数と整数剰余・整数除算

- 各桁に分解できる
  - 3456 % 10 = 6　　最下位の一桁
  - 3456 // 10 % 10 = 5
  - 3456 // 10 // 10 % 10 = 4
  - 3456 // 10 // 10 // 10 % 10 = 3
- n進数でも同じ
  - 234 % 7 = 3
  - 234 // 7 % 7 = 5
  - 234 // 7 // 7 % 7 = 4

負の数を伴う整数除算・剰余

- どちらかに（あるいは両方）負の数がはいる場合は、もともと整数除算は、床関数（floor関数：⌊x⌋…xと等しいか、xよりも小さいなかでの最大の整数）で計算されるので、マイナス方向に引っ張られた値になる。
- 例：
  - 10 // -3 ⇒ 10/-3 = -3.333..., floor(-3.3...) = -4
  - -10 // -3 ⇒ -10/-3 = 3.333..., floor(3.3...) = 3
  - -10 // 4 ⇒ -10/4 = -2.5, floor(-2.5) = -3

- 負の値が入った場合の剰余の計算方法は、除数で整数除算を行なった結果と除数を乗算して、被除数からの差分が計算される
  - m % -n ⇒ m - (m // -n) * -n
  - -m % n ⇒ -m - (-m // n) * n
  - -m % -n ⇒ -m - (-m // -n) * -n
- 例：
  - 12 % -7 ⇒ 12 // -7 = -2, 12 - (- 2 * -7) = 12 - 14 = -2
  - -4 % 12 ⇒ -4 // 12 = -1, -4 - (-1 * 12) = -4 + 12 = 8
  - -8 % -5 ⇒ -8 // -5 = 1, -8 - (1 * -5) * 1 = -8 + 5 = -3

- 参考：

  https://stackoverflow.com/questions/3883004/the-modulo-operation-on-negative-numbers-in-python

整数におけるビット演算子とシフト演算子

- ~　ビットの反転（単項演算子）
  - 例：~x
- &　ビットのAND（二項演算子）
  - 例：　x & 7
- |　ビットのOR（二項演算子）
  - 例：　x | 5
- ^　ビットの排他的論理和（二項演算子）
  - 例：x ^ 0b1101
- シフト算は２のべき乗とのかけ算・割り算になる
- >> 右シフト　２のべき乗で割った
  - 例：　12 >> 2　≡　12 // (2*2)
- << 左シフト　２のべき乗と掛けた
  - 例：　3 << 4　≡　3 * (2 * 2 * 2 * 2)

ビット演算の例（8bit演算で）

- 0xa3と ~0xa3
  - 0b10100011 各ビットを反転する(Pythonでは-(n+1))
    ↓↓↓↓↓↓↓↓
  - 0b01011100
- 0xa3 & 0x7a
  - 0b10100011 0xa3
  - 0b01111010 0x7a
  - 0b00100010 0x22 両方とも1のビットだけ1
- 0xa3 | 0x7a
  - 0b10100011 0xa3
  - 0b01111010 0x7a
  - 0b11111011 0xfb いずれかのビットが1なら1
- 0xa3 ^ 0x7a
  - 0b10100011 0xa3
  - 0b01111010 0x7a
  - 0b11011001 0xd9 どちらかのビットが1なら1

2進数への変換と表示

- 入力された２つの数をそれぞれ２進数で表示する
- 入力された文字列の２進数を整数に変換するには、int(文字列, base=2)を用いる
- 整数を２進数で表示するには、bin( )関数を用いる
- format関数を使って数値を文字列に変換する
  - format(数値, 書式文字列)
  - "b"…２進数　"d"…１０進数　"o"…８進数　"x"…16進数
- 文字列のzfill関数
  - 文字列.zfill(0で埋める桁数)
  - 使用例の書式：format(式, "b").zfill(桁数)
    - 式の値を、桁数分の0で埋めてから、２進数を表示する
- 表示する数の下位の有効桁数を指定したいときは、「値 & 0b1111」などを使う
  - 例：format(~(0b101) & 0xff, "b").zfill(8)
    - 下位８桁だけを表示する

整数の補数

- 足してその数になる数の組み合わせ
- １０の補数
  - 足して１０になる数の組み合わせ
  - 例： 1と9, 2と8, 3と7, 4と6, 5と5　　6に対する10の補数は？　　… 4
- 9の補数　0と9, 1と8, 2と7, 3と6, 4と5　　8に対する9の補数は？　　… 1
- 補数は引き算の替わりに用いられる
  - 基数の補数（2の補数、10の補数）と基数-1の補数（１の補数、9の補数）を利用
  - 基数の補数 = 基数-1の補数　＋ 1
  - 例： 10000 – 5678 = ？
  - 9999 – 5678 + 1 = 4321 + 1 = 4322
  - 15678 – 9876 = 5678 + 10000 – 9876 = 5678 + 9999 – 9876 + 1 = 5678 + 123 + 1 = 5802

プログラム上に現れる英字

- a … 変数名
- "a" あるいは 'a' … a一文字から成る文字列

- abc1234 … 変数名
- "abc1234" あるいは 'abc1234' … 文字列

- 以下は予約語なので、変数名としては使えない
- and as assert async await break class continue
- def del elif else except finally for from
- global if is import in lambda nonlocal not
- or pass raise return try yield while with

指数演算則

- 数学上の表現

$$a^0 = 1$$

$$a^1 = a$$

$$a^{-N} = \frac{1}{a^N}$$

$$a^{M+N} = a^M \times a^N$$

$$a^{M-N} = a^M \div a^N$$

$$a^{\frac{1}{N}} = \sqrt[N]{a}$$

$$a^{\frac{M}{N}} = \sqrt[N]{a^M}$$

$$a^{MN} = (a^M)^N$$

- Python上の表現

```
a ** 0
a ** 1
a ** -N
a ** (M+N)
a ** (M-N)
a ** (1/N)
a ** ( M/N)
a ** (M * N)
```

複素数の四則演算

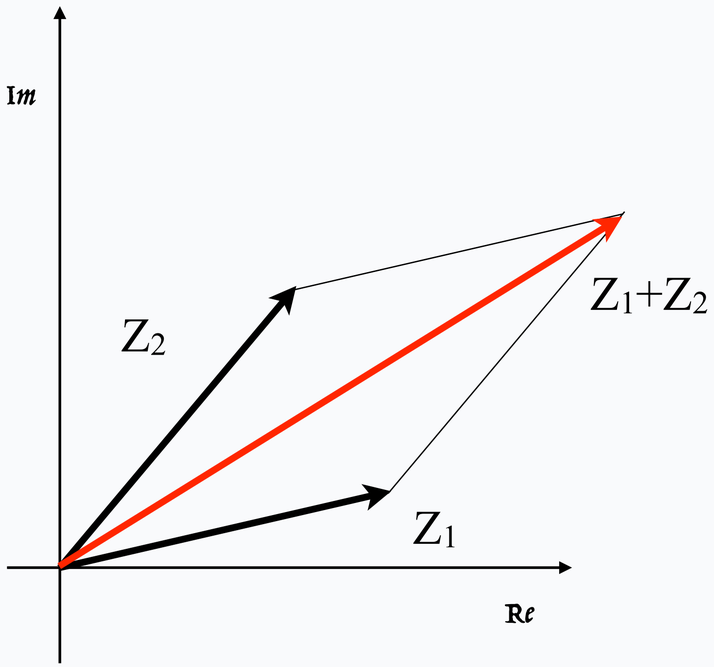
- $(x_1 + y_1 j) + (x_2 + y_2 j) \Rightarrow (x_1 + x_2) + (y_1 + y_2) j$
- $(x_1 + y_1 j) - (x_2 + y_2 j) \Rightarrow (x_1 - x_2) + (y_1 - y_2) j$
- $(x_1 + y_1 j) * (x_2 + y_2 j)$
  $\Rightarrow (x_1 * x_2 - y_1 * y_2) + (x_2 * y_1 + x_1 * y_2) j$

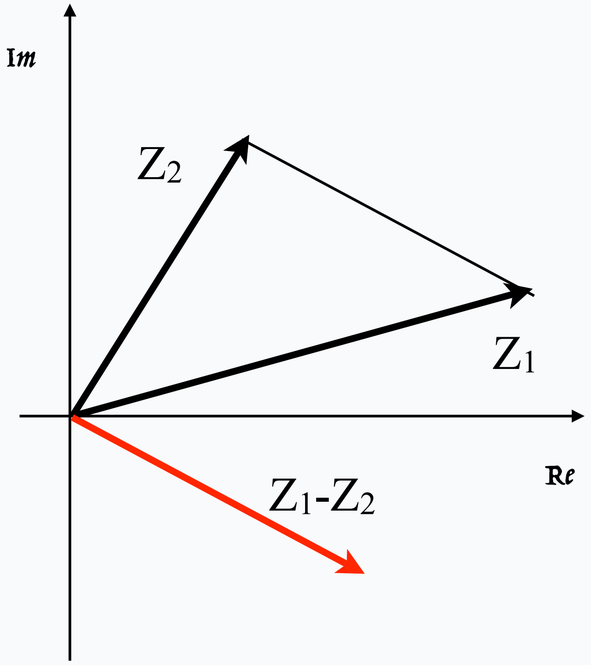
$$\frac{x_1 + y_1 j}{x_2 + y_2 j} \Rightarrow \frac{x_1 x_2 + y_1 y_2}{x_1^2 + x_2^2} + \frac{x_2 y_1 - x_1 y_2}{x_1^2 + x_2^2} j$$

- $z^n * z^m = z^{n+m}$
- $(z^n)^m = z^{nm}$
- $(z * w)^n = z^n * w^n$

複素数の加算・減算の意味

- 複素数の加算は、ガウス平面上では、２つのベクトルの加算として表される
- 複素数の減算は、ガウス平面上では、２つのベクトルの減算として表される





複素数の乗算・除算の意味

- 実数と複素数を乗算するのは、ノルムの拡縮
- 虚数jと複素数を掛けるのは、原点周りの90度の回転変換

- 一般に、複素数と複素数の乗算は、ノルムの積と偏角の和の組み合わせになる
- 複素数と複素数の除算は、ノルムの商と偏角の差の組み合わせになる

$$z = re^{i\alpha}$$
$$w = se^{i\beta}$$

$$zw = rse^{i(\alpha+\beta)}$$
$$|zw| = |z||w|$$
$$\arg zw = \arg z + \arg w$$

$$\frac{z}{w} = \frac{r}{s}e^{i(\alpha-\beta)}$$
$$\left|\frac{z}{w}\right| = \frac{|z|}{|w|}$$
$$\arg\frac{z}{w} = \arg z - \arg w$$

複素数の乗除算については

- デジタル・ビーイング・キッズ：画像処理の基礎講座
  - http://www.dbkids.co.jp/popimaging/seminar/complex/complexoperation.htm
- フーリエ級数まで解説されている

複素数のべき乗（実数）の意味

- 複素数の表現について、オイラーの公式を思い出そう

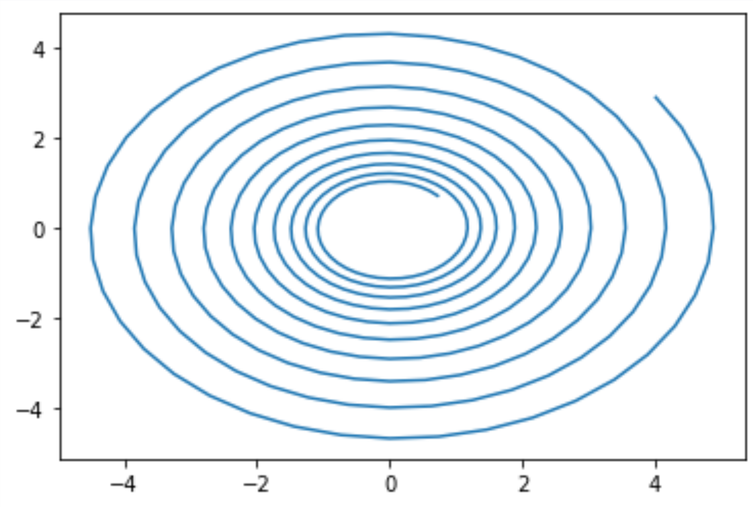
$$e^{j\theta} = \cos\theta + j\sin\theta$$

- nが整数で、θが実数のときに、ド・モアブルの定理(De Moivre's Law)が成り立つ

$$\left(e^{j\theta}\right)^n = \left(\cos\theta + j\sin\theta\right)^n = \cos n\theta + j\sin n\theta$$

- 極形式で偏角の角度のn倍に回転するということを示す
- 一般の複素数の場合には、ノルムのrが付くので、r^nの項が乗算され、nが大きくなるにつれ、原点からの螺旋状の形になる（ただし$\theta \neq 2m\pi$のとき）

複素数のべき乗をプロットしてみた例

- べき乗の場合、乗数を増やすと非常に大きくなるので、長さ（ノルム）を1.02にして、べき乗を1から0.2ずつ増やしてみた
- プログラム例：

```
from matplotlib import pyplot

value = (1+1j)*(1/2**0.5)*1.02
rlist, imlist = [], []
for n in range( 400 ):
    p = 1 + 0.2 * n
    rlist.append( (value**p).real )
    imlist.append( (value**p).imag )
    print( value**p )
pyplot.plot( rlist, imlist )
pyplot.show()
```