python™ プログラミング勉強会

公式サイトから資料を
DLできます。
パスワード: pk_py1130
puzzleknot.wp.xdomain.jp

puzzleknot
Tohoku University Programming Club

今日の流れ

- 第3回
- 関数とは
- プログラム設計の考え方 その2
- Python文法
  - 関数の定義
  - import文
  - while文
  - pipの使い方

2

関数とは

3

関数とは

- プログラミングで書く処理には、常に何らかの意味や目的がある
- それらは大抵、複数行に渡って同じ目的を持つ
- 例えば…

4

関数とは

- 例えば… 2点(x_1, y_1), (x_2, y_2)の距離を求めたい場合

```
d = ((x1-x2)**2 + (y1-y2)**2)**(1/2)
```

- $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
- 1回きりなら良いかもしれませんが、複数回となると何をやっているか若干わかりづらいです

5

関数とは

```
d = ((x1-x2)**2 + (y1-y2)**2)**(1/2)
```

- $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
- そもそも、この公式を知らない人からすれば何をやっているかわかりません
- 複雑な処理、計算を何行にも渡って繰り返す場合は、後に見返したときに自分がわからなくなってしまうかも…

6

関数とは

- 処理や計算に何かしら意味があり、繰り返し使うかもしれないときは**関数**にしましょう
- 関数: 処理や計算をひとまとめにしたもの

7

関数とは

- 関数: 処理や計算をひとまとめにしたもの
- 数学でいう関数とは若干異なります
  - 必ずしも値を入れたら返ってくる構造ではない
  - 処理をするだけだったり、複数の値が返ってきたり

8

関数とは

- 先程の「2点間の距離を求める」計算を関数にしてみましょう

9

関数とは

- **関数の宣言**は

```
def 関数名(引数1, …):
    処理
    return 返り値
```

という構造になっています

10

関数とは

```
def 関数名(引数1, …):
    処理
    return 返り値
```

- 引数: 関数内で利用するために、
  外部から受け取る値

  今回の場合、2点の座標

11

関数とは

```
def 関数名(引数1, …):
    処理
    return 返り値
```

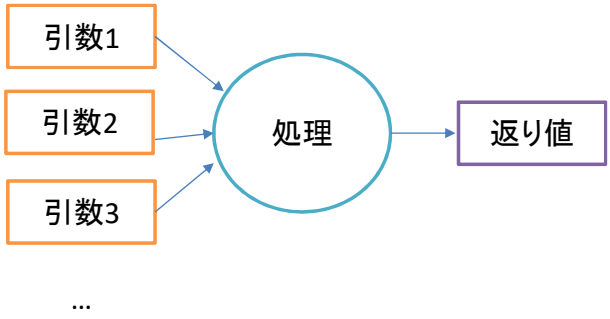
- 返り値: 関数の呼び出し元で利用して
  もらうために返す値

  今回の場合、2点の距離

12

関数とは

```
def 関数名(引数1, …):
    処理
    return 返り値
```



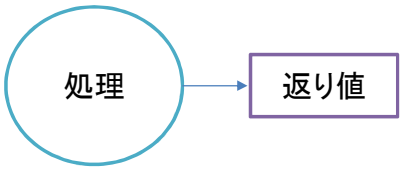
…

13

関数とは

```
def 関数名():
    処理
    return 返り値
```

- 引数はなくても大丈夫です



14

関数とは

```
def 関数名():
    処理
    return
```

- 返り値もなくても大丈夫です

15

関数とは

- 先程の「2点間の距離を求める」計算を関数にしてみましょう

```
def EuclidDist(x1, y1, x2, y2):
    dist = ((x1-x2)**2 + (y1-y2)**2)**(1/2)
    return dist
```

- 2点の座標を受け取って、(引数)
  距離を計算し、(処理)
  それを呼び出し元に返す(返り値)

16

関数とは

- 先程の「2点間の距離を求める」計算を関数にしてみましょう

```
def EuclidDist(x1, y1, x2, y2):
    dist = ((x1-x2)**2 + (y1-y2)**2)**(1/2)
    return dist
```

- 2点の座標を受け取って、(引数)
  距離を計算し、(処理)
  それを呼び出し元に返す(返り値)

17

関数とは

- 先程の「2点間の距離を求める」計算を関数にしてみましょう

```
def EuclidDist(x1, y1, x2, y2):
    dist = ((x1-x2)**2 + (y1-y2)**2)**(1/2)
    return dist
```

- 2点の座標を受け取って、(引数)
  距離を計算し、(処理)
  それを呼び出し元に返す(返り値)

18

関数とは

- 関数は、宣言しただけでは使ったことになりません
- プログラム内で使うには、呼び出しが必要です

19

関数とは

- 呼び出し

```
d = EuclidDist(1, 2, -1, -2)
```

- これは、
  2点(1,2), (-1,-2)の距離を計算して、
  その結果をdに代入しろ
  という命令です

20

関数とは

```
d = ((1-(-1))**2 + (2-(-2)**2)**(1/2)
```

関数を使って書き換えると

```
def EuclidDist(x1, y1, x2, y2):
    dist = ((x1-x2)**2 + (y1-y2)**2)**(1/2)
    return dist


d = EuclidDist(1, 2, -1, -2)
```

- 長くなっただけ？

21

関数とは

```
d = ((1-(-1))**2 + (2-(-2)**2)**(1/2)
```

関数を使って書き換えると

```
def EuclidDist(x1, y1, x2, y2):
    dist = ((x1-x2)**2 + (y1-y2)**2)**(1/2)
    return dist


d = EuclidDist(1, 2, -1, -2)
```

- しかしdがどういう値かは一目瞭然
- 違う点との距離を求めたい場合に
  修正する部分もすぐわかる

22

関数とは

```
d = ((1-(-1))**2 + (2-(-2)**2)**(1/2)
```

関数を使って書き換えると

```
def EuclidDist(x1, y1, x2, y2):
    dist = ((x1-x2)**2 + (y1-y2)**2)**(1/2)
    return dist


d = EuclidDist(1, 2, -1, -2)
```

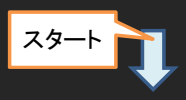
- しかしdがどういう値かは一目瞭然
- 違う点との距離を求めたい場合に修正する部分もすぐわかる

23

関数とは

- defで定義した関数は、呼び出されたときにはじめてその処理を行います
- 順番としては…

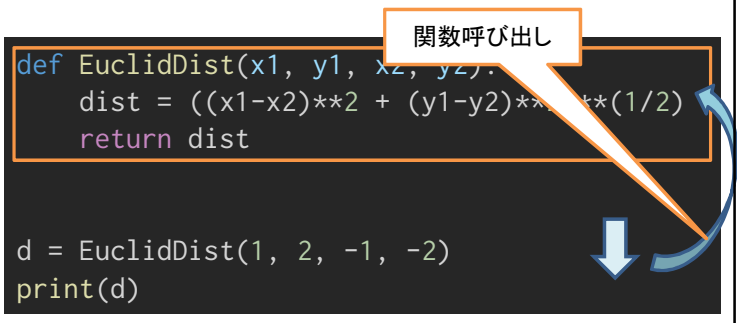
```
def EuclidDist(x1, y1, x2, y2):
    dist = ((x1-x2)**2 + (y1-y2)**2)**(1/2)
    return dist


d = EuclidDist(1, 2, -1, -2)
print(d)
```



24

関数とは

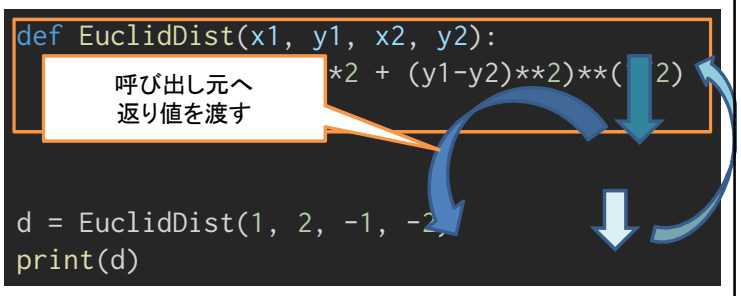
- defで定義した関数は、呼び出されたときにはじめてその処理を行います
- 順番としては…



25

関数とは

- defで定義した関数は、呼び出されたときにはじめてその処理を行います
- 順番としては…



26

関数とは

- defで定義した関数は、呼び出されたときにはじめてその処理を行います
- 順番としては…

```
def EuclidDist(x1, y1, x2, y2):
    dist = ((x1-x2)**2 + (y1-y2)**2)**(1/2)
    return dist


d = EuclidDist(1, 2, -1, -2)
print(d)
```

次の処理へ

27

関数とは

- プログラムはできるだけ
  - 再利用しやすい
  - 修正しやすい
  - 読みやすい

  ものを目指しましょう

- 自分が書いたコードでも、半年後にはわからなくなってしまいがちです
- 半年後の自分は他人

28

関数とは

- 関数を利用すると、何をやっているかがわかりやすくなります
- 積極的に利用しましょう

29

補足 – コメント

- 関数にするまでもないが、注釈を入れたい場合があります
- コメント機能を使いましょう

30

補足 – コメント

- Pythonのコメントは #で始まります

```
print(d) # dを出力

# 行内のシャープの後ろの文は、
# コメントと見なされます
# 複数行にまたがる場合は、
# その分シャープを付けます

""" あまりおすすめしませんが、
クオーテーション3つで囲うと
複数行にまたがるコメントを書けます
"""
```

やってみよう 1

- 3点(x1, y1), (x2, y2), (x3, y3) を頂点とする三角形の面積を計算する関数を作りましょう

- 3点は引数で受け取る
- 面積を返り値として返す
- 関数名は TriangleAreaとしましょう

32

やってみよう 1

- 3点(x1, y1), (x2, y2), (x3, y3) を頂点とする三角形の面積を計算する関数を作りましょう
- ヒント：面積公式

$$S = \frac{1}{2} \left| (x_1 - x_3)(y_2 - y_3) - (x_2 - x_3)(y_1 - y_3) \right|$$

- 絶対値は abs()

```
def 関数名(引数1, …):
    処理
    return 返り値
```

やってみよう 2

- いま作ったTriangleArea関数を利用して
  - list P に格納された3点から面積を計算
  - その結果を変数Sに代入
  - Sをprint文で出力してください
- P = [5, 7, 3, 4, 1, 2]
- [x1, y1, x2, y2, x3, y3]の順番です
- 復習: P[0], P[1], … で順番にlist内の要素にアクセスできます

34

やってみよう３

- いま作ったTriangleArea関数を利用して
  - list Points に格納された3点からそれぞれ面積を計算
  - その結果をlist Sに代入
  - Sをprint文で出力してください
- Points = [(5, 7, 3, 4, 1, 2),
  (3, 1, 4, 1, 5, 9),
  (2, 6, 5, 3, 5, 8),
  (9, 7, 9, 3, 2, 3)]

35

やってみよう３

- Points内には 3点の座標がtupleで入っています

```
for P in Points:
    # Pにはtupleが順番に格納されてく
```

- for文の1周目では
  Pに tuple (5, 7, 3, 4, 1, 2)が、
  2周目ではtuple (3, 1, 4, 1, 5, 9)が、…
  格納されています

36

やってみよう３

- Points内には 3点の座標がtupleで入っています

```
for P in Points:
    # Pにはtupleが順番に格納されてく
```

- ということは、
  このfor文内でTriangleArea関数を
  引数P[0], P[1] …という感じで呼べばOK
- 返り値をappend

37

やってみよう３

- 返り値をappend

```
S = []    # listの宣言
# 要素の追加
S.append(追加したい値)
```

38

import文

39

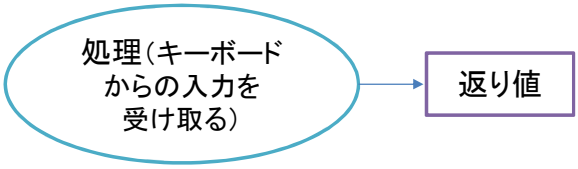
import文

- 多くの人が使うような関数は**モジュール（ライブラリ）**として配られていることがあります

40

関数とは

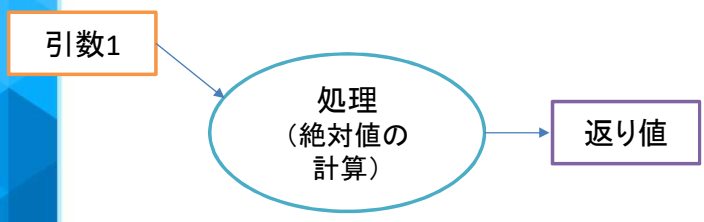
- 標準ライブラリ:
  Pythonでデフォルトで使える関数
- 例えば、input()



41

関数とは

- 標準ライブラリ:
  Pythonでデフォルトで使える関数
- 例えば、abs(値)



42

関数とは

- 標準ライブラリに入っていない関数も、Pythonをインストールするとくっついてくるモジュールに入っていることがある
- モジュールを使いたい場合は、**import文**を書きます

43

関数とは

- 例えば、数学関数
  (√, log, sin, cos, …)
- math モジュールに入っています

```
import math

print(math.sqrt(2)) # √2
print(math.log(2)) # log_e(2)
```

44

関数とは

- モジュールには定数が含まれている
  ことがある

```
import math

print(math.pi) # 円周率
print(math.e) # ネイピア数
```

- 定数なので () は付けない

45

やってみよう 4

- 範囲 $0 \leq x \leq 2\pi$ での $\sin x$ の値を
  0.1刻みで出力しましょう

- まず、xの値を0.1刻みで2πまで入れた
  listを用意する必要があります

46

やってみよう４

- まず、xの値を0.1刻みで2πまで入れたlistを用意する必要があります
- 2πを超えるまでlist xに値を追加していくことを考える
- 条件を満たす間ループし続けるには**while文**を使います

47

やってみよう４

- 条件を満たす間ループし続けるには**while文**を使います

```
while(条件式):
    # 条件式を満たす間ループを続ける
```

- ループは必ず抜けられるように条件を設定すること
- もしループが抜けられず、プログラムが止まらない(無限ループ)ときはCtrl + C で止まります

48

やってみよう４

- 2πを超えるまでlist xに値を追加していくことを考える

- list xと変数 i(=0) を宣言
- i が 2π以下である限り
  - xに i を追加
  - i に 0.1を加える

49

やってみよう４

- 2πを超えるまでlist xに値を追加していくことを考える

```
x = []
i = 0
while(i <= 2*math.pi):
    x.append(i)
    i += 0.1
```

50

やってみよう４

- 作ったlist xをfor文で回して、sin(x)を表示

51

pipの使い方

52

pip とは

- pythonをインストールした際には
  ないモジュールもある
- 第三者が作ったモジュールなど
  （モジュールは自分で作れます）
- 簡単にインストールする方法がpip

53

pip とは

- 簡単にインストールする方法がpip
- pip はpythonをインストールした際に
  付属するパッケージマネージャ

54

pip とは

- 端末（ターミナル）で
  `$ pip install (モジュール名)`
- 今回は、numpyというものを
  インストールしてみましょう
- mathより強力な算術計算モジュール
- `$ pip install numpy`

55

numpy

- numpyを使ってみましょう
- 行列計算
- 宣言

```
import numpy

a = numpy.asarray([1, 2, 3], [4, 5, 6])
b = numpy.asarray([1, 2], [3, 4], [5, 6])
```

56

numpy

```
import numpy

a = numpy.asarray([1, 2, 3], [4, 5, 6])
b = numpy.asarray([1, 2], [3, 4], [5, 6])
```

$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}$ と $\begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix}$ を宣言

57

numpy

```
print(a.dot(b))
```

aとbの積 $\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}\begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix}$

58

numpy

```
c = numpy.asarray([[1, 2], [3, 4]])
print(numpy.linalg.det(c))
```

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ の行列式

59