

CS第1 テーマ2

テーマ2の目標 プログラミング体験

プログラミングのために開発された技法を学ぶ

- 配列, 文字列
- 関数(サブルーチン)

レポート課題

課題2 循環小数

課題3 暗号解読に挑戦

本日の講義内容

1. 配列とその使い方 宿題

2. 文字列の処理方法 宿題

CS第1 演習ガイド

1. ログインする.
2. Terminal を動かす
3. 講義のウェブページから プログラム をダウンロードする.
 - Downloads(ダウンロード)フォルダに day3.zip が展開される
4. Terminal で day3 をCS1 に移動
 - `cd Documents/CS1`
 - `mv ~/Downloads/day3 ./`
 - `cd day3`

CotEditor でファイルを開くには

```
$ open -a coteditor ファイル名
```

ターミナルの便利な機能

- 補完: **TAB** キーで, ファイル名等を補完できる
- 履歴: **↑** キーで過去に入力したコマンドを出せる

```
$ ls  
abcPrint.py junkan.py max.py stringPrint.py sum.py  
$ ls abc ← ここで TAB  
  
$ ls abcPrint.py ファイル名が補完される  
abcPrint.py  
  
$ ←  
$ ls abcPrint.py ここで ↑  
一つ前のコマンドが出てくる
```

複数回 ↑ を押すと古いコマンドに戻っていく

1. 配列(リスト)

- 要素(値)を一行に並べたもの

$[v_0, v_1, \dots, v_{n-1}]$

配列 a

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|----|----|
| 2 | 4 | 6 | 8 | 10 | 12 |

- 要素を上書きできる

要素の参照: $a[i]$

- 添え字(インデックス) i の要素
- 添え字 \approx 位置 (0 から 要素数 - 1)

要素の代入: $a[i] = e$

- 添え字(インデックス) i の要素を e の計算結果で置き換える

配列の長さ(要素数): $\text{len}(a)$

配列：使ってみよう

```
$ python
...
>>> a = [3, 1, 4, 1, 5, 9, 2]
>>> a
[3, 1, 4, 1, 5, 9, 2]
>>> a[0]
3
>>> a[4]
5
>>> a[4] = 7
>>> a[4]
7
>>> a
[3, 1, 4, 1, 7, 9, 2]
>>> len(a)
7
```

新しい配列を作成し変数 **a** に代入
配列 **a** の値を確認(参照)

配列 **a** の添え字 **0** の値を参照

配列 **a** の添え字 **4** に値 **7** を代入

配列の長さ(要素数)

復習：アニメーションプログラムの例

大きな数を各変数に格納(絵のデザイン)

[illegible]

めんどうだけど
これはしょうがない

```
t = 0
while t < 30:
    print(d1)
    print(d2)
    print(d3)
    print(d4)
    print(d5)
    print(d6)
    print(d7)
    print(d8)
    print(d9)
    print(d10)
    print()
    time.sleep(0.1)
    t = t + 1
```



めんどろ

1. 配列

復習: アニメーションプログラムの例

プログラム smile.py

大きな数を配列に格納

```
# 出力: スマイルマーク
d[0] = 10000000000000000000000000000000
d[1] = 10000000000110000110000000000000
d[2] = 10000000000110000110000000000000
d[3] = 10000000000000000000000000000000
d[4] = 10000011000000000000000110000000
d[5] = 10000001100000000000000110000000
d[6] = 100000001100000000011000000000
d[7] = 100000000001111111100000000000
d[8] = 10000000000000000000000000000000
d[9] = 10000000000000000000000000000000
```

配列 d

| | |
|--------|--|
| d[0] | |
| d[1] | |
| d[2] | |
| d[3] | |
| d[4] | |
| d[5] | |
| d[6] | |
| d[7] | |
| d[8] | |
| d[9] | |

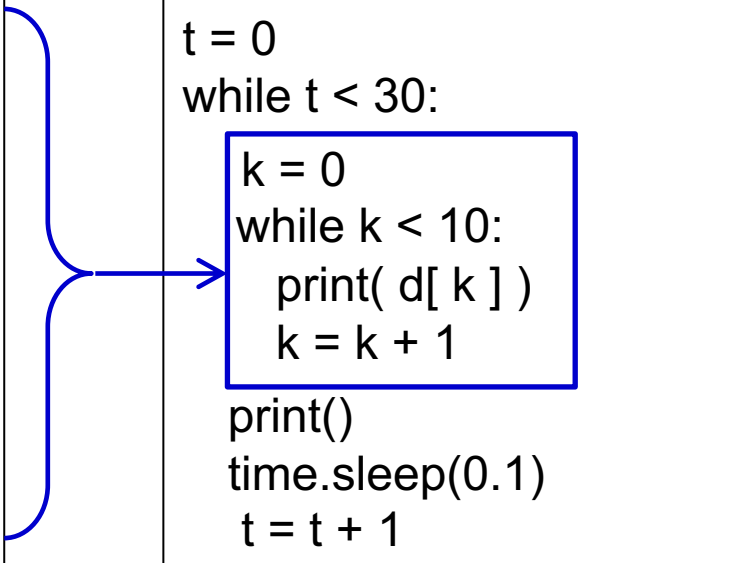
1. 配列

復習 : アニメーションプログラムの例

プログラム smile.py

画面への出力

```
t = 0
while t < 30:
    print( d[ 0 ] )
    print( d[ 1 ] )
    print( d[ 2 ] )
    print( d[ 3 ] )
    print( d[ 4 ] )
    print( d[ 5 ] )
    print( d[ 6 ] )
    print( d[ 7 ] )
    print( d[ 8 ] )
    print( d[ 9 ] )
    print()
    time.sleep(0.1)
    t = t + 1
```



```
t = 0
while t < 30:
    k = 0
    while k < 10:
        print( d[ k ] )
        k = k + 1
    print()
    time.sleep(0.1)
    t = t + 1
```


繰り返し: for 文

```
for i in range(n):  
    ブロック
```

変数 i の値を 0 から $n - 1$ まで変えて, ブロックを実行

- n : 計算結果が整数になる式

```
for i in range(m,n):  
    ブロック
```

変数 i の値を m から $n - 1$ まで変えて, ブロックを実行

1. 配列

復習: アニメーションプログラムの例

プログラム smile.py

画面への出力

```
t = 0
while t < 30:
    print( d[ 0 ] )
    print( d[ 1 ] )
    print( d[ 2 ] )
    print( d[ 3 ] )
    print( d[ 4 ] )
    print( d[ 5 ] )
    print( d[ 6 ] )
    print( d[ 7 ] )
    print( d[ 8 ] )
    print( d[ 9 ] )
    print()
    time.sleep(0.1)
    t = t + 1
```

```
t = 0
while t < 30:
    k = 0
    while k < 10:
        print( d[ k ] )
        k = k + 1
    print()
    sleep(0.1)
    t = t + 1
```

```
for k in range(10):
    print( d[ k ] )
```

先週習いたかった !!

1. 配列

同種のデータを多数扱うときに便利

例: 6個の総和を求める

| | | | | | | |
|------|---|---|---|---|----|----|
| | 0 | 1 | 2 | 3 | 4 | 5 |
| 配列 a | 2 | 4 | 6 | 8 | 10 | 12 |

プログラム sum6.py

```
a = [2, 4, 6, 8, 10, 12]
```

↑ 配列の初期値の決め方

```
s = 0  
k = 0  
while k < 6:  
    s = s + a[k]  
    k = k + 1
```

```
print(s)
```

```
s = 0  
for k in range(6):  
    s = s + a[k]
```

数学でも
似た表現があるよ

$$\text{総和} = \sum_{k=0}^5 a_k$$

↑
添え字

添え字

Python では配列の添え字は0 から

1. 配列

例: 不定個の総和を求める

配列 a

| 0 | 1 | 2 | 3 | 4 | 5 | ... |
|---|---|---|---|---|---|-----|
| | | | | | | ... |

$$\sum_{k=0}^n a_k$$

プログラム sum.py

```
a = list(map(int, input().split()))
n = len(a)
# 以下が総和の計算部分
s = 0
for k in range(n):
    s = s + a[k]
print(s)
```

整数を配列に入力する方法. (これは決まり文句)

「len(配列)」で配列の要素数が得られる.

```
$ python sum.py
-3 8 19 -4
20
```

実行例

個々に空白で区切る
改行がデータの終わり

1. 配列 (Python での使い方)

例: 最大値を求める

配列 a

| | | | | | | | |
|--|---|---|---|---|---|---|-----|
| | 0 | 1 | 2 | 3 | 4 | 5 | ... |
| | | | | | | | ... |

$\max(a_0, a_1, \dots, a_n)$

max.py

```
a = list(map(int, input().split()))
n = len(a)

max = -100000 # マイナス無限大と言える数
maxj = -1     # 最大値のインデックス
for j in range(n):
```

宿題 (1)

```
print(max)
print(maxj)
```

```
$ python max.py
-3 8 19 -4
19
2
```

実行例

最大値19が
添え字2にある

2. 文字列

文字が並んだもの(1文字の場合もある)


str="Coffee"



引用符

n = len(str)

| | 0 | 1 | 2 | 3 | 4 | 5 |
|-----|---|---|---|---|---|---|
| str | C | o | f | f | e | e |



str[2]

str に格納されている文字列の長さ

引用符は, シングルクォート, ダブルクォートどちらでも良い

文字列：使ってみよう

```
$ python
...
>>> 'Japan'
'Japan'
>>> "Japan"
'Japan'
>>> s = "Tokyo"
>>> s
'Tokyo'
>>> s[0]
'T'
>>> s[2]
'k'
>>> len(s)
5
>>> s + "Tech"
'TokyoTech'
```

文字列定数：シングルクォート

文字列定数：ダブルクォート

文字列 Tokyo を変数 s に代入
変数 s の値を確認(参照)

文字列 s の添え字 0 の文字を参照

文字列 s の長さ

文字列の連結

多重定義された演算子

Python では, 演算子 `+` が整数にも文字列にも使える

⇒ 多重定義(オーバーロード) されている

```
$ python
```

```
...
```

```
>>> 3 + 8
```

```
11
```

```
>>> "Tokyo"+"Tech"
```

```
'TokyoTech'
```

```
>>> "Tokyo" + 8
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: can only concatenate str (not "int") to str
```


2. 文字列

文字が並んだもの(1文字の場合もある)

stringPrint.py

```
puts("文字列を入力しよう")
str = input()
n = len(str)
for i in range(n):
    print( str[i] )
```

文字列を入力する方法



```
$ python stringPrint.py
```

```
文字列を入力しよう
```

```
Ice%%cream
```

```
I
```

```
c
```

```
e
```

```
%
```

```
%
```

```
c
```

```
r
```

```
e
```

```
a
```

```
m
```

```
m
```

実行例

文字と文字コード ASCII

- ・ 文字もコンピュータ内では数字(正確には 2 進列)で表わされる
- ・ 文字を 2 進列で表すこと(または表したものを **文字コード** という
- ・ 文字コードはいろいろあるが, 英数字を表すもので世界的に最も普及しているのが **ASCII** である

ASCII (American Standard Code for Information Interchange)

- 1文字を7ビットで表現 (1ビット付加して1バイト使う)
コード 0 ~ 127
 - アルファベット a ~ z : 97 ~ 122
 - アルファベット A ~ Z : 65 ~ 90
 - 数字 0 ~ 9 : 48~57
 - その他の記号など
 - スペース : 32, + : 43

Python: 文字と文字コードの変換

- `ord` : 文字からASCIIコードに変換
- `chr`: ASCIIコードから文字に変換

実行例

```
>>> ord("a")
97
>>> ss = "Cabcz"
>>> ss[2]
'b'
>>> ord(ss[2])
98
```

```
>>> chr(97)
'a'
>>> chr(97+25)
'z'
>>> chr(ord("b"))
'b'
```

その他の文字コード

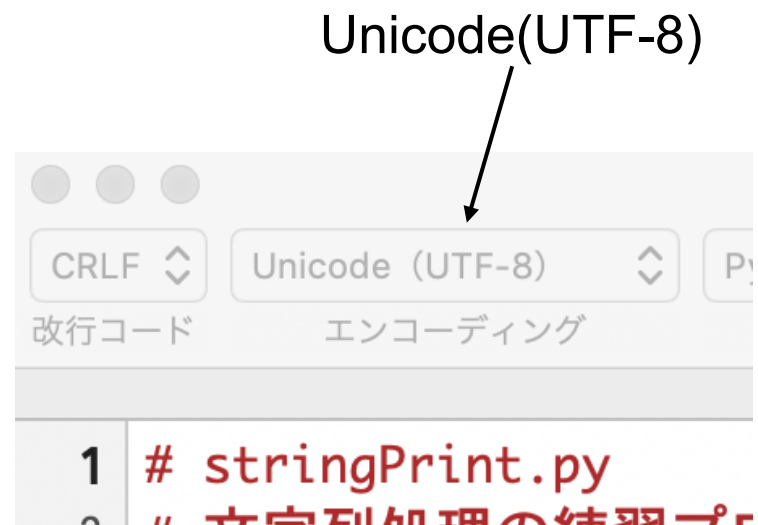
- 日本語
 - JIS(ISO-2022-JP)
 - 漢字：2バイト
 - 漢字とASCII を特殊な文字列（エスケープシーケンス）で切り替える
 - Shift-JIS
 - EUC

その他の文字コード

- Unicode (世界中の文字を表現)
 - UTF-8
 - 1文字: 1バイト ~ 4バイト
 - 0 ~ 127 は ASCIIと一致
 - Python (バージョン3)はUTF-8を使う
 - UTF-16
 - 1文字: 16ビット(2バイト) or 4バイト
 - UTF-32

文字コードを確認してみよう

```
$ open -a coteditor stringPrint.py
```



2. 文字列 (Python での使い方)

例: 英小文字のみ画面に出す

英小文字のみ画面に出力するプログラムを作ろう!

abcPrint.py

```
print("文字列を入力しよう")
ss = input()
leng = len(ss)
for i in range(leng):
    if ???:
        print(ss[i])
```

ヒント: a の ASCII = 97
~
z の ASCII = 122

```
$ python abcPrint.py
文字列を入力しよう
Ice%%cream
c
e
c
r
e
a
m
```

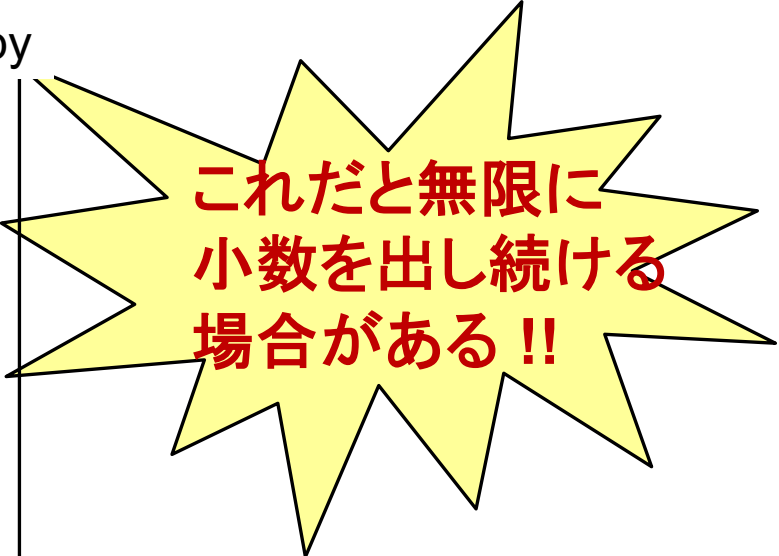
CS第1 レポート課題2(予告)

課題 循環小数の循環を止めよ！

配列は同じようなデータを統一的に処理するには便利な道具だが、それ以外にも賢い使い方がいくつかある。その例を考えてみよう。

```
print("分母 d を下さい")
d = int(input())
print("1 / ", d, " を求めます")
stop = False
leng = 0
x = 1
while not stop:
    x = x * 10
    q = x // d
    leng = leng + 1
    print(leng, ":", q)
    time.sleep(0.5)
    x = x % d
    if x == 0:
        stop = True
```

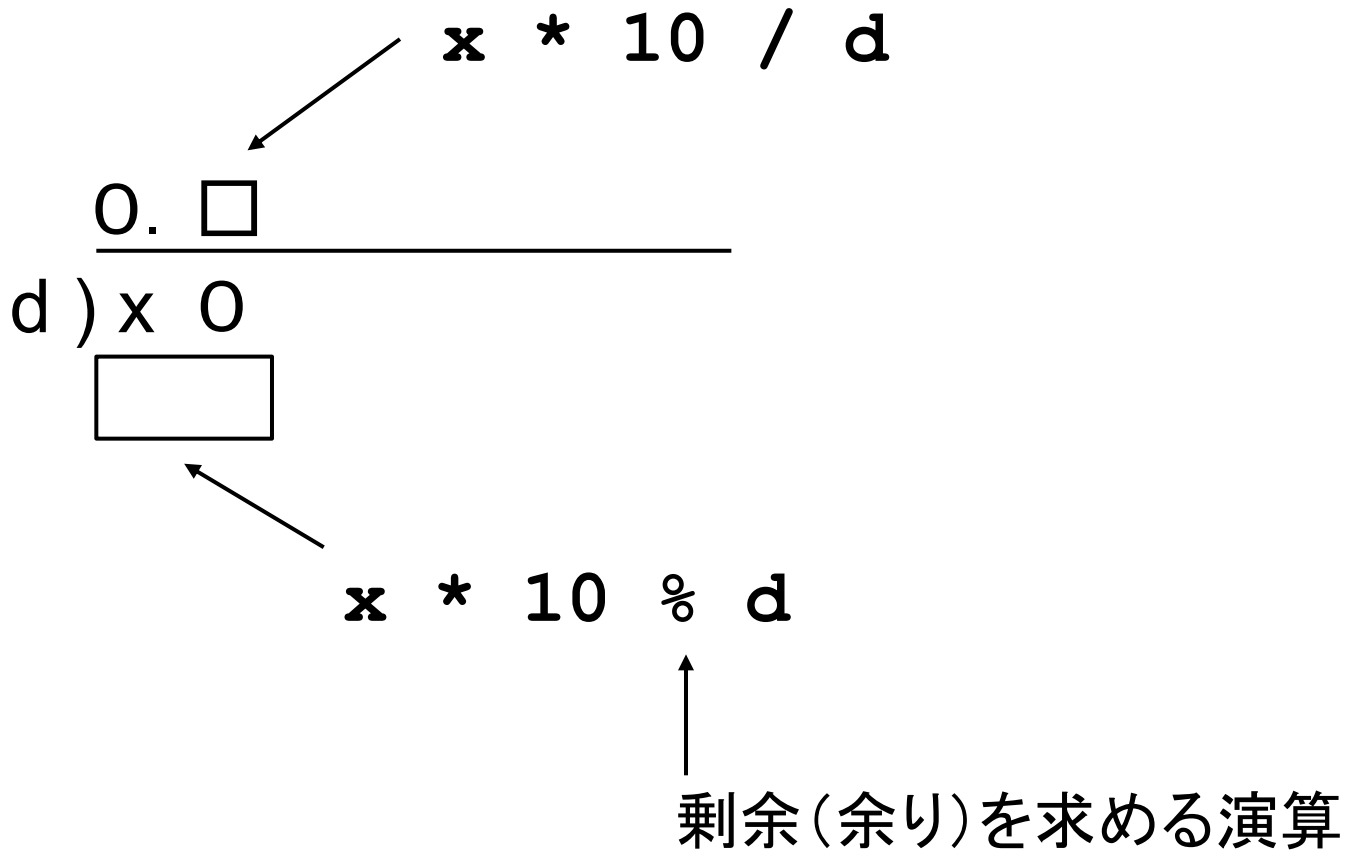
junkan.py



これだと無限に
小数を出し続ける
場合がある！！

プログラムの考え方

x / d を小数として表示する. ただし, $x < d$



まとめ: Python (1)

【演算子】

| 演算 | 使用例 | 意味 |
|----|----------|--------------|
| + | $x + y$ | x と y の足し算 |
| - | $x - y$ | x から y の引き算 |
| * | $x * y$ | x と y の掛け算 |
| // | $x // y$ | x を y で割った商 |
| % | $x \% y$ | x を y で割った余り |
| ** | $x ** y$ | x の y 乗 |

【関係演算子】

| 関係 | 使用例 | 意味 |
|----|------------|--------------------|
| >= | $x \geq y$ | x は y より大きいかまたは等しい |
| > | $x > y$ | x は y より大きい |
| == | $x == y$ | x は y と等しい |
| != | $x != y$ | x は y は等しくない |
| < | $x < y$ | x は y より小さい |
| <= | $x \leq y$ | x は y より小さいかまたは等しい |

まとめ: Python (2)

【論理演算子】

| 論理記号 | 使用例 | 意味 |
|------|---------|----------------------------|
| and | x and y | x と y の論理積 (両方が真のとき真) |
| or | x or y | x と y の論理和 (少なくとも一方が真のとき真) |
| not | not x | x の否定 (x が真のとき偽, x が偽のとき真) |

まとめ: Python (3)

【配列】 配列の作成 `[0, 0, -5, 4]`
`[0] * 4` 各要素の初期値が 0, 要素数が4の配列

要素の参照 `aa[i]` `aa` の i 番目 (添え字 i は 0 から)

大きさ `len(aa)` 配列 `aa` の長さ (= 要素数)

※「添え字」は「インデックス」(index) ともいう.

【文字列】 文字列定数 `"TokyoTech"`

指定方法 `s[i]` `s` の i 文字目 (添え字 i は 0 から)

長さ `len(s)` 文字列 `s` の長さ

連結 `"Tokyo" + "Tech"`

まとめ: Python(4)

【繰り返し文】

while 条件式:
 ... ← 条件式が成立している間
 ... を繰り返す

for m in range(a,b): ← 変数 m の値を a から b - 1 まで
 ... 1 ずつ増加させながら ... を繰り返す

【条件分岐文】

if 条件式:
 ...(A) ... ← 条件式が成立したときは ...(A) ...を実行

| |
|---|
| else: ...(B) ... ← そうでないときは ...(B) ...を実行 |
|---|

省略可

まとめ: Python (5)

【文字】

文字からコード `ord("a")` ← 文字 a のASCIIコード

コードから文字 `chr(97)` ← ASCIIコード が 97 の文字

【入出力】

`print(a, b, "hello", c)` ← 変数 a, b の値, 文字列 hello, 変数 c の
値を空白で区切って画面に表示する