

2023年度

高度セキュリティPBL / 先進セキュリティPBL

Pythonの基礎

大阪大学大学院 工学研究科

電気電子情報通信工学専攻 情報通信工学部門

宮地研究室 奥村 伸也

目次

1. Jupyter Notebook
2. 基本的な文法
3. パッケージ関数
4. いろいろなPythonの関数
5. データ表現の相互変換
6. 整数値と文字列の相互変換
7. ファイルの入出力
8. グラフの描画
9. 参考文献
10. 事前課題

1. Jupyter Notebook (1/3)

- 概要

- ウェブブラウザ上で Python を
実行・表示できるツール
- ノートのように扱うことができ、
式と答えが表示された形式のまま保存できる

1. Jupyter Notebook (2/3)

- 使い方

1. Notebookの起動

- Jupyter Notebookを起動

2. ファイルの新規作成

- New -> Python3

3. ファイルの保存

- File -> Save and Checkpoint

4. ファイル名の変更

- File -> Rename

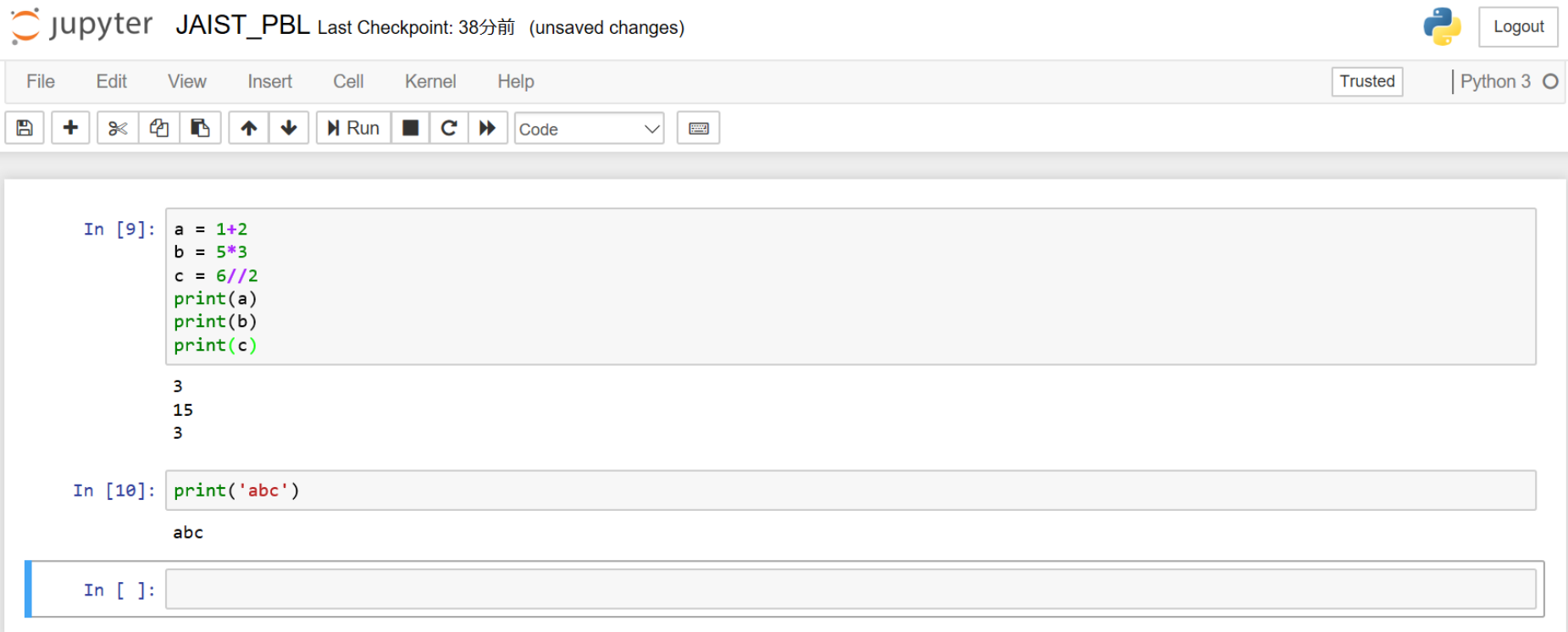
5. ファイルを開く

- File -> Open

1. Jupyter Notebook (3/3)

6. 実行

- ステートメントを記述して Shift キーを押しながら Enter キーを押す



The screenshot displays the Jupyter Notebook interface. At the top, the header shows the Jupyter logo, the text "JAIST_PBL", and "Last Checkpoint: 38分前 (unsaved changes)". On the right, there is a "Logout" button and a Python 3 logo. Below the header is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", and "Help". A toolbar contains icons for saving, adding cells, undo, redo, and running code. The main area shows two code cells. The first cell, labeled "In [9]:", contains Python code that calculates and prints the values of variables a, b, and c. Its output shows the values 3, 15, and 3 on separate lines. The second cell, labeled "In [10]:", contains the code `print('abc')` and its output is the string "abc". A third, empty code cell labeled "In []:" is at the bottom.

jupyter JAIST_PBL Last Checkpoint: 38分前 (unsaved changes) Python 3 Logout

File Edit View Insert Cell Kernel Help Trusted

`In [9]: a = 1+2
b = 5*3
c = 6//2
print(a)
print(b)
print(c)`

3
15
3

`In [10]: print('abc')`

abc

`In []:`

2. 基本的な文法(1/16)

2-1. コメント

- ・ 一行コメント : # から行末まで
- ・ 複数行コメント : 3 重クオート文字列で囲う
文字列として扱われるので
無視されるわけではない

サンプルコード

```
print("abc") # コメント  
"""  
  
この中は  
コメント  
"""
```

2. 基本的な文法(2/16)

2-2. 変数

- 「変数名 = 値」で変数を作成
- 変数の値は変更することができる
- 変数はどのような型でも初期化・代入できる

サンプルコード

```
a = 1      # 変数 a を 1 で初期化
b = 2      # 変数 b を 2 で初期化
c = a + b  # 変数 c に a + b の結果を代入
print(c)   # c の中身を表示
c = c + 3  # c の値を変更
print(c)   # 変更した c の中身を表示
```

2. 基本的な文法(3/16)

2-3. 型

- 代表的な型

型名	書式例	説明
整数型 (int)	1	多倍長整数型
浮動小数点型 (float)	1.0	
文字列型 (str)	‘あいう’ “あいう”	Unicode文字列(日本語も可) ” または “” で文字列を括る
バイト列型 (bytes)	b'abc'	ASCII文字列
ブール型 (bool)	True	TrueまたはFalse
None 型 (NoneType)	None	空の状態を表す

- 型の確認にはtype()関数を用いる

2. 基本的な文法(4/16)

サンプルコード

```
a = 1          # 整数型 (int)
b = 1.0        # 浮動小数点型 (float)
c = 'あいう'   # Unicode 文字列での 'あいう'
c1 = "あいう"  # ダブルクォーテーションで囲っても同じ
d = b'abc'     # ASCII 文字列での abc
e = True       # ブール型
f = None       # None 型

print(type(a)) # <class 'int'>
print(type(b)) # <class 'float'>
```

2. 基本的な文法(5/16)

2-4. 演算子

- 代表的な演算子

演算子名	記号	説明
加算	+	
減算	-	
乗算	*	
除算	/	浮動小数点型を返す
商	//	
剰余	%	
論理積	&	
論理和		
排他的論理和	^	
ビット左シフト	<<	
ビット右シフト	>>	

2. 基本的な文法(6/16)

サンプルコード

```
a = 1 + 2 - 3 * 4 # a = -9
b = 6 / 3          # b = 2.0
c = 16 // 5        # c = 3
d = 16 % 5         # d = 1

# ビット演算
e = 1 & 1 # e = 1
f = 1^0   # f = 1, a^b は a の b 乗という意味ではない
g = 1 | 0  # g = 1

# 10進→2進
a = 24535
print("a=",format(a, 'b')) # 101111111010111と表示
# 2進→10進
a = b'101011110101'
print("a=", int(a, 2)) # 2805と表示

# 文字列の連結
print("Sec" + "Cap") # SecCap と表示
```

2. 基本的な文法(7/16)

2-5. print関数

- プログラム実行中に文字列を表示する
- 文字列中に別の文字列を挿入するにはformat()を使う

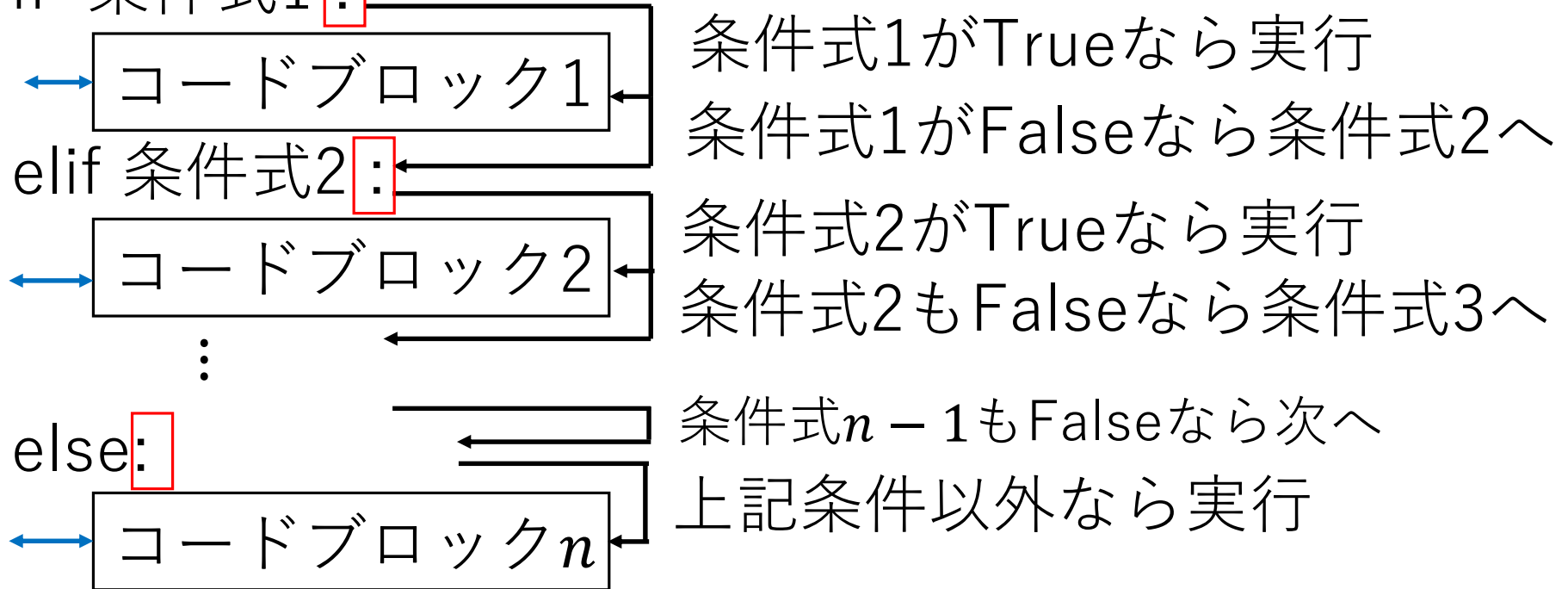
サンプルコード

```
s1 = "Hello, world."  
s2 = "Hello, {}.format("world")  
a = "Hello"  
b = "world"  
s3 = "{}, {}".format(a, b)  
print("Hello, world.")          # “Hello, world.”と表示  
print("s1=",s1)                  # “Hello, world.”と表示  
print("s2=",s2)                  # “Hello, world.”と表示  
print("s3=",s3)                  # “Hello, world.”と表示  
# “2 times 3 equal 6” と表示  
print("{} times {} equal {}".format(2,3,6))
```

2. 基本的な文法(8/16)

2-6. 制御文(if文)

if 条件式1 : コロンが必要



インデント(字下げ)してコードを記述

- elifは同じif文中に何回も使える

2. 基本的な文法(9/16)


サンプルコード

```
n = 0
if n < 0:
    print('n<0')
elif n == 0:
    print('n=0')
else:
    print('n>0')
# 最初にn = 0と初期化しているので、n=0と表示される
```

2. 基本的な文法(10/16)

2-7. 制御文(for文)

for 変数 in オブジェクト : コロンが必要

 コードブロック

インデント(字下げ)してコードを記述

- in の後に書かれたオブジェクト内の各要素に対してループ処理を行う
- オブジェクトには rangeオブジェクトや listオブジェクトを指定する
- break文でループを抜けることができる

2. 基本的な文法(11/16)

サンプルコード

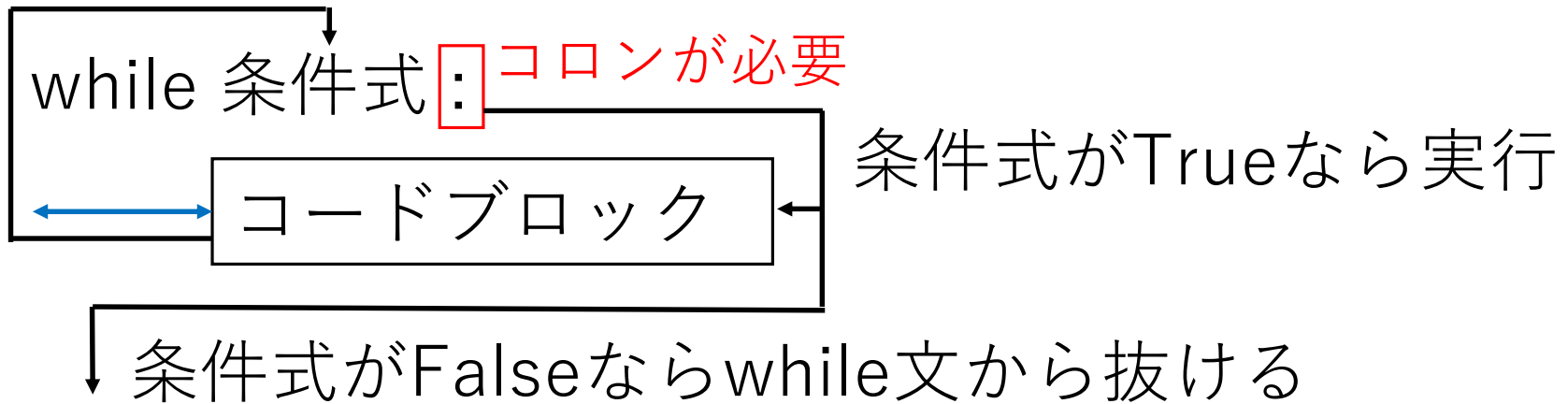
```
a = 0
for i in range(5): # for i in [0, 1, 2, 3, 4]: としても挙動は同じ
    a = a + i
print("a=", a)          # 10と表示
```

```
a = 0
for i in range(5):
    a = a + i
    if i == 3:
        break          # i == 3でfor文を抜ける
print("a=", a)          # 6と表示
```


2. 基本的な文法(12/16)

2-8. 制御文(while文)

実行し終わったらまた条件式を確認



インデント(字下げ)してコードを記述

- break文でループを抜けることができる

2. 基本的な文法(13/16)

サンプルコード

```
a = 4
b = 0
while a > 0:
    b = b + a
    a = a - 1
print("b=", b)          # 10と表示
```

```
a = 4
b = 0
while a > 0:
    b = b + a
    a = a - 1
    if a == 1:
        break

print("b=", b)  # 9と表示
```

2. 基本的な文法(14/16)

2-9. リスト

L = [1つまたは複数のオブジェクト]
(各オブジェクトはカンマで区切られる)

- 可変長の配列表現
- 任意の型のオブジェクトを格納
 - リストの入れ子も可能
- 要素へのアクセスは[]でインデックス指定
- リストのサイズはlen()関数を利用

2. 基本的な文法(15/16)

サンプルコード

```
# リストの作成
a = [] # 空リスト
b = ['RSA','AES','DES'] # 複数要素からなるリスト作成
c = ['RSA',['AES','DES','RC4']] # リストの入れ子
len(b) # リストの要素数 3
array = [1, 2, 3, 4]
# 任意の要素を取り出す
first = array.pop(0)      # first == 1, array == [2, 3, 4]
# 任意の要素を追加する
array.insert(0, 5)        # array == [5, 2, 3, 4]
# 末尾を取り出す
last = array.pop()        # last == 4, array = [5, 2, 3]
# 末尾に追加
array.append(9)           # array == [5, 2, 3, 9]
# 末尾にリストを追加
array.extend([0, 1])      # array == [5, 2, 3, 9, 0, 1]
# リストの 1 番目から 3 番目までの要素を取り出す (スライシング)
array[1:4]                # [2, 3, 9]と表示. array は変更されない
```

2. 基本的な文法(16/16)

2-10. 自作関数

def 関数名(引数1, 引数2, ..., 引数 n): コロンが必要

← コードブロック

return 戻り値

インデント(字下げ)してコードを記述

- “関数名(a, b, c, \dots)”は引数 a, b, c, \dots に対応する戻り値を返す
- サンプルコード

```
def ex(a, b):  
    t = a  
    a = b  
    b = t  
    return (a, b)
```

ex(1,3) # (3,1) と表示される

3. パッケージ関数(1/5)

3-1. パッケージ関数の利用方法

- パッケージをimportする
 - importは最初の一回で良い
- (パッケージ名).(関数名)で呼び出す

サンプルコード

```
import os          #OSパッケージを読み込む
os.urandom(16)     #OSパッケージに属するurandom関数の呼び出し
                  #16バイトの乱数を返す
```

3. パッケージ関数(2/5)

3-2. 乱数生成

- `os.urandom(size)`
 - 暗号用途に適したsizeバイトからなるランダムなバイト列を返す
- `random.randrange(a, b)`
 - $a \leq r < b$ となる擬似乱数rを生成する
 - 乱数シードは`random.seed()`で設定する

サンプルコード

```
import os
rand = os.urandom(16)      #16バイトの乱数生成
import random
random.seed(rand)          #乱数生成器の初期化
random.randrange(0, 100)   #0以上100未満の乱数を返す
```

3. パッケージ関数(3/5)

randomモジュールの関数は、乱数生成器として決定論的なメルセンヌツイスタを利用しており、シミュレーションには適しているが、暗号目的には向かない。暗号目的に利用する乱数には、secretsモジュールの関数を利用するとよい。

- secrets.choice(L) (L: リスト)
 - (空でない)Lから要素をランダムに選択.
- secrets.randbelow(n)
 - 0以上n未満のランダムな整数を返す.
- secrets.randbits(k)
 - ランダムな k ビット整数を返す.

3. パッケージ関数(4/5)

3-3. 時間計測

- time.perf_counter()を用いる
- 時間計測に本質的でない処理が入らないように注意

サンプルコード

```
import time, os, binascii
p = 184908779667576050572947262326548513689
a = []
for i in range(1000):
    a.append(int(binascii.hexlify(os.urandom(20)), 16))
t0 = time.perf_counter()
for i in range(1000):
    pow(2, a[i], p)
t1 = time.perf_counter() - t0 # 終了時刻 - 開始時刻
print(t1/1000)
```

この部分の時間を計測

3. パッケージ関数(5/5)

3-4. パッケージと関数の例

パッケージ名	関数名
binascii	hexlify, unhexlify
os	urandom
random	seed, randrange
secrets	choice, randbelow, randbits
sympy	gcd, gcdex, randprime
time	perf_counter

4. いろいろなPythonの関数(1/5)

サンプルコード

```
# str(x) : x(整数値など) の 文字列"x" を返す.
```

```
a = str(4) # a = "4"
```

```
print(type(a)) # <class 'str'> と表示
```

```
# int(x) : x (整数値や実数など) の文字列を整数値に変換
```

```
print(int("4")) # 4 と表示 (4 は int型)
```

```
# 実数を小数点以下を切り捨てた整数値に変換することもできる
```

```
print(int(4.6)) # 4 と表示 (小数点以下は切り捨てられる)
```

```
# input() : キーボードからの入力を文字列として取得
```

```
x = input()
```

```
print(x) # 入力したものを表示
```

```
print(type(x)) # <class 'str'> と表示
```

```
x = int(input()) # キーボードで入力した整数値をint型で取得したい場合
```

```
# リストと似たものに集合 (set) がある. [] ではなく {} を用いる.
```

```
S = {1,2,3,4,5, 4, 5, 6}
```

```
print(S) # {1,2,3,4,5,6} と表示 (要素の重複は考慮されない)
```

```
L = list(S) # L = [1,2,3,4,5,6] となる (集合をリストに変換)
```

```
S2 = set(L) # S2 = S (リストを集合に変換)
```

4. いろいろなPythonの関数(2/5)

サンプルコード

リストと集合については、要素の削除や追加その他の操作について下記サイトが参考になる

リスト <https://note.nkmk.me/python-list-common/>

<https://note.nkmk.me/python-list-comprehension/>

<https://note.nkmk.me/python-enumerate-start/>

<https://note.nkmk.me/python-slice-usage/>

<https://note.nkmk.me/python-list-clear-pop-remove-del/>

集合 <https://note.nkmk.me/python-set/>

※古いバージョンのPythonを用いているので、現バージョンでの挙動と一致しない可能性あり

初等整数論関連

`sympy.gcd(a,b)` : 整数 a, b の最大公約数を返す

`sympy.gcdex(a, b)` : 整数 a, b の最大公約数 d と $ax+by=d$ を満たす整数 x, y を返す

`sympy.lcm(a,b)` : 整数 a, b の最小公倍数を返す

$ab = \text{gcd}(a, b)\text{lcm}(a, b)$ が成り立つ

`sympy.isprime(p)` : p が素数なら `True`, p が合成数なら `False` を返す

`sympy.randprime(a, b)` : a 以上 b 未満の素数をランダムに返す

素数判定, 素数生成については下記のサイトが参考になる

https://python.atelierkobato.com/prime_number/

`sympy.factorint(n)` : n の因数分解を返す

`abs(n)` : n (数値) の絶対値を返す

`divmod(a, b)` : $a = qb + r$ ($0 \leq r < b$) を満たす整数 q, r を返す

4. いろいろなPythonの関数(3/5)

サンプルコード

```
import sympy as sp
a = 14
b = 32
print(sympy.gcd(a, b)) # 2 と表示
print(sympy.gcdex(a, b)) # (7, -3, 2) と表示
print(sympy.lcm(a, b)) # 224 と表示
print(a*b // sympy.gcd(a,b)) # 224 と表示
print(a*b // sympy.lcm(a,b)) # 2 と表示
print(sympy.isprime(5)) # True と表示
print(sympy.isprime(9)) # False と表示
print(sympy.randprime(1,100)) # 1~99 の間の素数をランダムに返す
print(sympy.factorint(36)) # {2: 2, 3: 2} と表示 ( $36 = 2^2 \times 3^2$ )
print(abs(-9)) # 9 と表示
q, r = divmod(17, 5)
print(q, r) # 3 2 と表示 ( $17 = 3*5 + 2$ )
# import sympy as sp とすると, sp.gcd(a, b) 等のように省略して利用できる
```

4. いろいろなPythonの関数(4/5)

サンプルコード

```
# max(L) : リストL (または集合L) 中の数値で最大のものを返す
# min(L) : リストL (または集合L) 中の数値で最小のものを返す
# statistics.mean(L) : リストL (または集合L) 中の数値の平均を返す
# statistics.median(L) : リストL (または集合L) 中の数値の中央値を返す
# statistics.variance(L) : リストL (または集合L) 中の数値の分散値を返す
# statistics.stdev(L) : リストL (または集合L) 中の数値の標準偏差を返す
from statistics import mean, median, variance, stdev
data = [2,4,8,16,32,64,128,256]
print('dataの最大値: {}'.format(max(data))) # 256 と表示
print('dataの最小値: {}'.format(min(data))) # 2 と表示
print('dataの平均: {}'.format(mean(data))) # 63.75 と表示
print('dataの中央値: {}'.format(median(data))) # 24.0 と表示
print('dataの分散: {}'.format(variance(data))) # 7838.214285714285 と表示
print('dataの標準偏差: {}'.format(stdev(data))) # 88.5336901168944 と表示

# pow(a, k) : a の k乗を計算 ( a**k と等価)
# pow(a, k, n) : a の k乗 を n で割った余りを計算 ( $a^k \pmod{n}$ ) を計算)
# pow(a, k, n) は pow(a, k) % n よりも効率が良い
```

4. いろいろなPythonの関数(5/5)

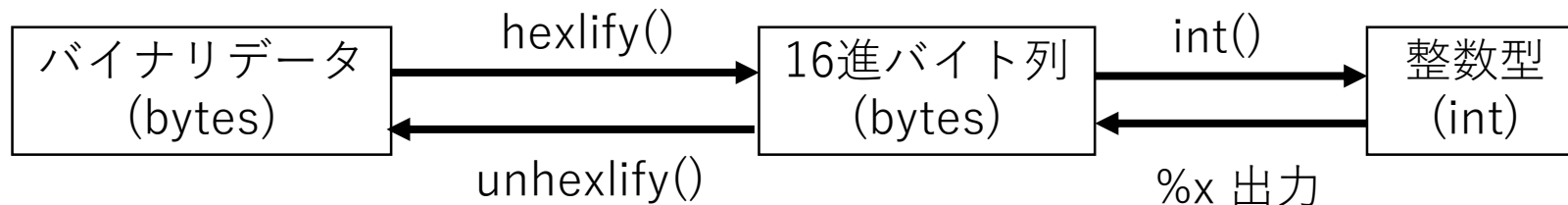
サンプルコード

```
a = 2
k = 14
n = 452
print(pow(a, k)) # 16384 と表示
print(pow(a, k, n)) # 112 と表示
# bin(k) : k の2進文字列を返す (先頭に '0b' がついている)
k = 2741
bin_k = bin(k)
print(bin_k) # '0b'+kの2進表現の文字列('0b101010110101')を表示
len_k = len(bin_k) - 2 # kのビット長を取得 (k.bit_length() と等価)
print(bin_k[2:len_k]) # bin_kの'0b'を除いた部分(kの2進表現)を表示
# ハッシュ関数の計算
import hashlib, random
m = 'Python'
m_b = m.encode() # mをバイト列に変換
h = hashlib.shake_256(m_b).digest(256) #shake256によるmのハッシュ値(256バイト) を計算
h_int = int.from_bytes(h, byteorder = 'big') # ハッシュ値を整数値に変換
print(h)
print(len(h))
print(h_int)
print(h_int.bit_length()//8)
```

5. データ表現の相互変換(1/2)

5-1. バイナリデータ/16進バイト列/整数型の相互変換

- 対応関係



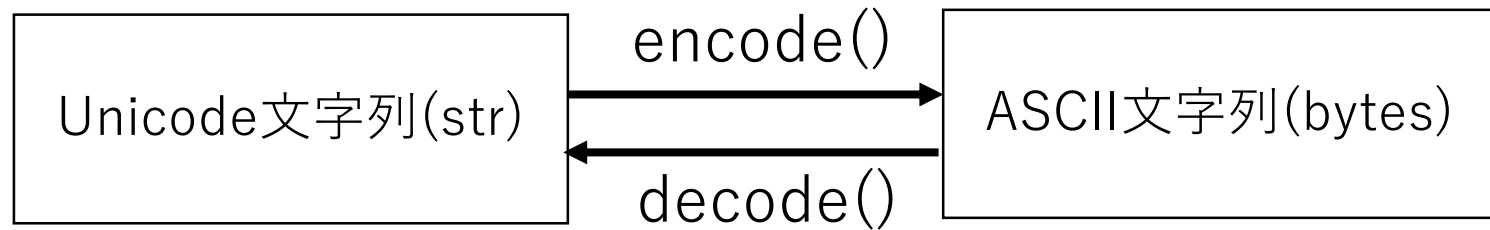
サンプルコード

```
import os, binascii
a0 = os.urandom(10) # バイナリデータ
a1 = binascii.hexlify(a0) # バイナリデータ → 16 進バイト列
a2 = int(a1, 16) # 16 進バイト列 → 整数型
a3 = b"%x" % a2 # 整数型 → 16 進バイト列
a4 = binascii.unhexlify(a3) # 16 進バイト列 → バイナリデータ
```


5. データ表現の相互変換(2/2)

5-2. バイト列と文字列の相互変換

- 対応関係



サンプルコード

```
a = 'あいうえお'
print("a=", a) # 「あいうえお」と表示
b = a.encode() # Unicode 文字列 → バイト列
print("b=", b) # b'\xe3\x81\xe2\xe3\x81\xe4\xe3\x81\xe6\xe3\x81\xe8\xe3\x81\xa'と表示
c = b.decode() # バイト列 → Unicode 文字列
print("c=", c) # 「あいうえお」と表示
```

6. 整数値と文字列の相互変換(1/4)

6-1. 文字列->整数値

- `int.from_bytes()`関数
 - `int.from_bytes(バイト列, byteorder)`
 - 与えられたバイト列の整数表現を返す
 - 整数を表すバイトオーダー(big or little)を指定する
 - > big : 最上位バイトがバイト配列の最初
 - > little : 最上位バイトがバイト配列の最後
- 文字列から整数値への変換方法
 1. 文字列をASCII文字列(バイト列)に変換
 - > 文字列.encode()
 2. 1のバイト列を整数値に変換
 - > `int.from_bytes(バイト列, byteorder)`

6. 整数値と文字列の相互変換(2/4)

サンプルコード

```
a = int.from_bytes(b'¥x00¥x10', byteorder='big')
b = int.from_bytes(b'¥x00¥x10', byteorder='little')
print("a=", a) #16と表示
print("b=", b) #4096と表示
m = 'BasicSecCap'
m_byte = m.encode()
m_int = int.from_bytes(m_byte, byteorder = 'big')
print("m_int=", m_int) # 80249302612977449313722736
```

6. 整数値と文字列の相互変換(3/4)

6-2. 整数値->文字列

- 整数値.to_bytes()関数
 - 整数値.to_bytes(length, byteorder)
 - 与えられた整数を表すバイト列(lengthバイト)を返す
 - byteorderについては、int.from_bytes()と同様

サンプルコード

```
x = (1024).to_bytes(2, byteorder='big')
y = (1024).to_bytes(2, byteorder='little')
print("x=", x) # b'¥x04¥x00' と表示
print("y=", y) # b'¥x00¥x04' と表示
m_int = 80249302612977449313722736
m_byte = m_int.to_bytes(32, byteorder = 'big')
m = m_byte.decode()
print("m=", m) # BasicSecCap と表示
```

6. 整数値と文字列の相互変換(4/4)

下記のように別の方法もある.

サンプルコード

```
import binascii

# 文字列->整数値
M = "BasicSecCap"
Mb = binascii.hexlify(M.encode())
M_int = int(Mb, 16)
print("M_int=", M_int) # 80249302612977449313722736

# 整数値->文字列
Mh = hex(M_int)[2:].encode()
M = binascii.unhexlify(Mh).decode()
print("M=", M) # BasicSecCapと表示
```

7. ファイルの入出力(1/6)

7-1. ファイルを開く

`fileobject = open(filename, mode)`

`filename` : ファイル名を指定(文字列)

`mode` : ファイルをどのモードで開くかを指定

- 2文字の文字列

- > 1文字目はファイル操作の種類を指す

- > 2文字目はファイルのタイプを指す

`mode`の例(1文字目) :

1. 'w' : - 書き込み用にファイルを開く

- 指定のファイル名のファイルがなければ新たに生成する

- ファイルが存在する場合は上書きされる

7. ファイルの入出力(2/6)

- 2. 'a' : ファイルが存在する場合に上書きではなく
末尾の後ろに追加すること以外は、'w'と同様
- 3. 'r' : ファイルを読み出し用を開く

・ modeの例(2文字目)

- 1. 't' (またはなし) : テキストファイルを開く
- 2. 'b' : バイナリファイルを開く

※ファイルを開いた後は閉じる必要がある

※with as 構文を使うと、最後に自動的にファイルを
閉じてくれる

7. ファイルの入出力(3/6)

7-2. ファイル入出力

- write(), dump()関数(書き込み)とread(), readlines(), load()関数(読み出し)を利用する
- read(): ファイル全体を文字列として読み込む
- readlines(): ファイル全体を一行ごとに分割したリストとして読み込む
- write(): ファイルに()内の文字列を書き込む
- dump()/load(): データ型を意識することなくファイルの入出力が可能(pickleパッケージが必要)

7. ファイルの入出力(4/6)

サンプルコード1

```
s1 = 'This is an example'
s2 = 'end'
f1 = open('example.txt', 'wt')
f1.write(s1+'¥n') # s1を書き込んで改行
f1.write(s2)      # s2を書き込む
f1.close()
f2 = open('example.txt', 'r')
s3 = f2.read()
print("s3=",s3)
'''
This is an example
end
'''
f2.close()
with open('example.txt', 'r') as f3: #with as 構文でファイルを開く
    s4 = f3.readlines()
    print("s4=", s4)                #['This is an example¥n', 'end']と表示
                                    # f3.close()はなくてよい
```

7. ファイルの入出力(5/6)

サンプルコード2

```
x = ['aaa','bbb','ccc']
with open('example2.txt','w') as f:
    f.write('¥n'.join(x)) # joinで文字列のリストを間に '¥n'(改行)をはさみながら連結
with open('example2.txt','r') as f:
    y = [row.strip() for row in f.readlines()] # strip()で '¥n'を削除
print("y=", y) # ['aaa','bbb','ccc']と表示
```

サンプルコード3

```
import pickle
x = [1, 2]
with open('data.txt','wb') as f:
    pickle.dump(x, f)
with open('data.txt','rb') as f:
    y = pickle.load(f)
print("y=",y) # [1, 2]と表示
```

7. ファイルの入出力(6/6)

CSVファイルへの入出力は下記のようにできる

サンプルコード

```
import csv
with open('file.csv', 'w', newline = '') as f: # 書き込み
    writer = csv.writer(f)
    writer.writerow([1, 'RSA', 'A'])
    writer.writerow([2, 'AES', 'S'])
    writer.writerow([3, 'RC4', 'S'])
with open('file.csv', 'r', newline = '') as f: # 読み込み
    reader = csv.reader(f)
    a = [row for row in reader]

print("a=",a) # [['1', 'RSA', 'A'], ['2', 'AES', 'S'], ['3', 'RC4', 'S']]と表示
```

8. グラフの描画

数値データのグラフを描くときは、`matplotlib.pyplot`を利用する

サンプルコード

```
import numpy as np
import matplotlib.pyplot as plt
plt.axis([0.0, 5.0, -1.5, 1.5]) #x軸は0.0 から 5.0, y軸は-1.5から1.5
plt.grid(True)
x = np.arange(0, 5, 0.01) # x軸の値は0.01刻み
y = np.sin(x) # y = sin(x)(リスト)
plt.title('y = sin(x)') # グラフのタイトル
plt.xlabel('x') # x軸のラベル名
plt.ylabel('y') # y軸のラベル名
plt.plot(x, y) # x = [x1, ..., xn], y = [y1, ..., yn]->(xi,yi)を描写
plt.show() # y = sin(x)のグラフを表示
```

9. 参考文献

Python 3.10.11 ドキュメント

<https://docs.python.org/3.10/>

言語リファレンス

<https://docs.python.jp/3/reference/index.html>

ライブラリリファレンス

<https://docs.python.jp/3/library/index.html>

10. 事前課題

(1) \mathbb{N} を自然数全体の集合 ($0 \notin \mathbb{N}$) とする.

オイラーの (トーシェント) 関数 φ は

$$\varphi : \mathbb{N} \rightarrow \mathbb{N}; n \mapsto \#\{x \in \mathbb{N} \mid 1 \leq x < n, \gcd(x, n) = 1\}$$

で定義される (集合 A に対して $\#A$ は A の要素数) .

自然数の入力 n について $\varphi(n)$ を求める関数

`Euler_phi(n)` を作成せよ. また, $n = 12345$ のときの出力を求めよ.

ヒント : 整数 x, n の最大公約数 $\gcd(x, n)$ は `sympy.gcd(x, n)` で求めることができる.

(2) 整数 a, b について,

$$ax + by = \gcd(a, b)$$

を満たす整数の組 $[x, y]$ (リスト)のうち, 可能な限り絶対値 $|x|, |y|$ が小さいものを k 個出力する関数 `solve_diophantus(a, b, k)` を作成せよ.

$a = 525327, b = 35847, k = 100$ の時の出力を求めよ.

ヒント: まず, $ax_0 + by_0 = \gcd(a, b)$ を満たす整数 x_0, y_0 を求める (`sympy.gcdex()`関数を利用する).

この時, $a' = \frac{a}{\gcd(a, b)}, b' = \frac{b}{\gcd(a, b)}$ とすると

$$x_i = x_0 + b'i, y_i = y_0 - a'i \quad (i \geq 1) \text{ は} \\ ax_i + by_i = \gcd(a, b)$$

を満たす.

(3) 素数 p について, $1 \leq a < p$ を満たす整数 a で $a^{p-1} \equiv 1 \pmod{p}$, $a^k \not\equiv 1 \pmod{p}$ ($1 \leq k < p-1$) を満たすものが存在する. このような a のうち, 最小のものを出力する関数 `min_primitive_root(p)` を作成せよ. ここで, 整数 x, y, n ($n > 1$) について, n が $x - y$ を割り切る時, $x \equiv y \pmod{n}$ と書く. また, $p = 10007$ の時の出力を求めよ. ヒント: $a^k \equiv g \pmod{p}$ を満たす g ($0 \leq g < p$) を求めるには `pow(a, k, p)` を利用する.

(4) 自然数 x, a, b ($\gcd(a, b) = 1$) について, x 以下の素数の個数を $\pi(x)$, x 以下かつ $p \equiv a \pmod{b}$ を満たす素数 p の個数を $\pi_{a,b}(x)$ と書く.

$2 \leq x \leq 10000000$ の各自然数 x について, $\pi(x)/\pi_{1,3}(x)$ の値を prime_thm1.csv に, $\pi(x)/\pi_{2,3}(x)$ の値を prime_thm2.csv に出力せよ.

ヒント: 自然数 n が素数かどうかを判定するには `sympy.isprime(n)` を利用する.

※それなりに時間がかかるので注意してください。
PCのスペックによっては計算できない可能性があります. その場合は, $2 \leq x \leq 1000000$ の範囲で構いません.

(5) (4)で作成したcsvファイルの値を読み込み,
 $\pi(x)/\pi_{1,3}(x)$, $\pi(x)/\pi_{2,3}(x)$ を同じグラフに図示せよ.
グラフのタイトルは,
"Theorem on arithmetic progressions"
とすること.

事前課題の提出について

下記は課題提出に関する注意事項となります。

- ・ (1) - (5)の問題の回答となるPythonのプログラムを提出してください。
- ・ Pythonのjupyter notebookのファイル(ipynbファイル)で提出してください。
(その際、プログラムの実行し出力が表示されたままでファイルを保存し提出してください)
- ・ (1) - (5)のプログラムをひとまとめにして1つのipynbファイルを提出してください。
(4)のcsvファイルは提出不要です。
- ・ ipynbファイルの提出方法は下記の通りです。
 1. jupyter notebookにより課題の解答を作成(出力も表示させる)
 2. 出力を表示させたまま、画面上の「File」->「Rename」を選択しファイル名指定して保存
 3. jupyter notebookの起動時に開かれるHomeに保存した課題のファイルを確認
 4. そのファイルの□をクリックしてチェックマークを入れると、画面上に「download」が表示されるのでそれをクリック(ファイルがRunningの状態なら画面上に「shutdown」と表示されるので、それをクリックして再び✔を入れると「download」と表示される)
 5. ダウンロードしたファイルを提出(csvファイルも同様の方法で提出)

〆切：6/23(金), 23:59

提出先：Moodleから提出