The Kyoto College of Graduate Studies for Informatics

kcg.edu

コンピュータプログラミング概論

秋期第７回

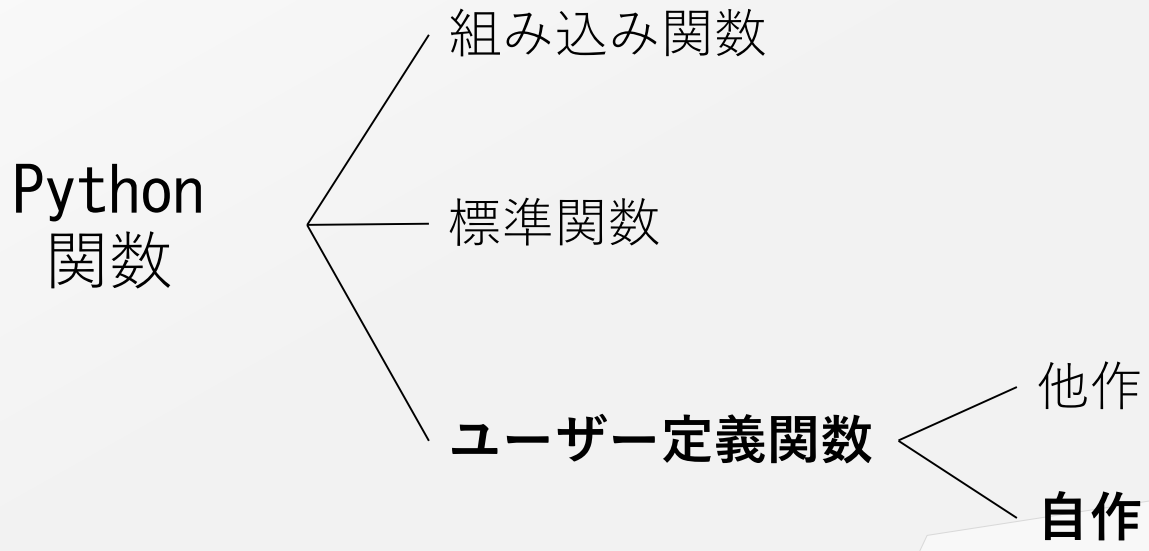
安　平勲

h_an@kcg.ac.jp

ユーザー定義関数

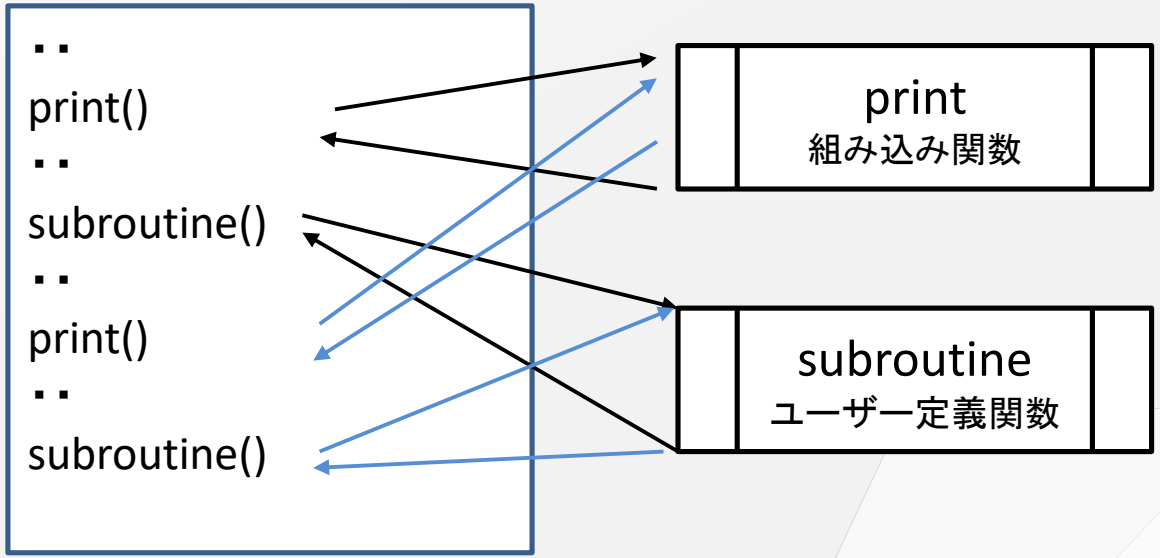
関数

- Pythonの関数には色々な種類ある



関数の目的

- よく使う処理コードを利用できる（組込み・標準関数）
- **自プログラムで繰返し利用するコードをまとめる**⇒一回のコーデイングで済む　　※”サブルーチン”と呼ぶ言語がある



ユーザー定義関数（自作）

- **ユーザー定義関数を定義**defineする

```
def 関数名（仮引数, 仮引数, ・・）：
    # インデントを忘れない
     命令文１
     ・・・
     命令文ｎ
     return 戻り値
```

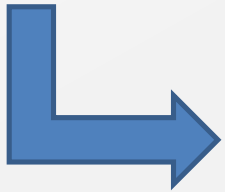
- **自作した関数**を呼び出す

```
関数名（引数, 引数, ・・）
```

※pythonでは**必ず定義の後に呼ぶ**（関数定義の前に呼ぶとエラー）

ユーザー定義関数（自作）

```
# 高さ5，半径10の円柱の体積
height = 5
radius = 10
#円柱の体積を求める公式
cylinder_volume = (radius**2) *3.14 *height

print(cylinder_volume)
```



```
#円柱の体積を求める関数
def cylinder_volume(height,radius) :
    volume = (radius ** 2) * 3.14  * height
    return volume

# 高さ5，半径10の円柱の体積は
print(cylinder_volume(5,10))
```

ユーザー定義関数（自作）

```
def bmi(height,weight):
    """BMIを計算・表示する関数
      入力は身長と体重"""
    bmi = weight / height**2 * 10000
    bmi = round(bmi, 1)
    if bmi < 18.5:
        message = "痩身"
    elif 25.0 <= bmi:
        message = "肥満"
    else:
        message = "標準体重"
    print("あなたの肥満度は",bmi,"で",message,"です")

print("あなたの肥満度を計算します")
w = int(input("体重をkgで入力："))
h = int(input("身長をcmで入力："))
bmi(h,w)  # 関数呼び出し
```

- 関数を呼ぶ前に**定義def**
- **関数名**はbmi
- **仮引数**はheightとweight
- 定義範囲は**インデント**で示す
- **def**文の直後のコメントは**docstring。help**関数で読める

✓ **戻り値がない**タイプ
※関数内で処理が完結
※return文は省略可能

ユーザー定義関数（自作）

```
def calc(num):
    unit_price = 98          # 単価を設定
    print('単価は', unit_price)
    result = int(num) * unit_price
    return result

# キーボードから個数を入力し、価格を表示する
while True :
    input_num = input("個数を入れてください。(qで終了)>> ")
    if input_num == "q" :
        break
    result = calc(input_num)# calc()を呼び、価格を計算する
    print('価格は', result)
print('処理終了')
```

- **戻り値を返す**タイプ
  ※returnの後に値か式を書く
- この例では価格が返るので,
  変数resultに戻り値を代入

関数と戻り値

再掲

- 関数には**戻り値return value**があり／なしの2タイプがある

戻り値	関数の例	説明
あり	strings = input(”*文字列*”)	コンソールからの入力文字列が**戻り値として返る**。戻り値を変数で受け取る（代入できる）
なし	print(*値*)	引数の値（文字列または数値など）をコンソールに出力する ＊戻り値はない

ユーザー定義関数；キーワード引数

```
def bmi(height,weight):
    """BMIを計算・表示する関数
      入力は身長と体重"""
    bmi = weight / height**2 * 10000
    bmi = round(bmi, 1)
    if bmi < 18.5:
        message = "痩身"
    elif 25.0 <= bmi:
        message = "肥満"
    else:
        message = "標準体重"
    print("あなたの肥満度は",bmi,"で",message,"です")

print("あなたの肥満度を計算します")
w = int(input("体重をkgで入力："))
h = int(input("身長をcmで入力："))
bmi(weight=w,height=h)  # 関数呼び出し
```

キーワード引数

- 実引数に"仮引数**=**"を指定
  ※引数の順序が変わってもOK
- これまでのものは**位置引数**と呼ぶ（引数の順序が従業）

ユーザー定義関数；キーワード引数

```
def calc(num, size = "M") :     # キーワード引数に初期値を設定
    unit_price = {"S": 100, "M":150, "L":180} # サイズ別単価表
    price = unit_price[size] * int(num)
    return price

# キーボードから個数とサイズを入力し、価格を表示する
while True :
    input_num = input("個数を入れてください。(qで終了)>>")
    if input_num == "q" :
        break
    input_size = input("SかMかLを入れてください>>").upper()
    if input_size !="":
        price = calc(input_num,size = input_size)
    else:
        price = calc(input_num)
    print('サイズ',input_size ,'10価格は', price)
```

キーワード引数

- **仮引数に初期値**を設定できる。初期値は省略値として扱われる

ユーザー定義関数；可変の位置引数

```
def fruit(*args): #  引数が *変数
    print(args)   #  引数が可変のタプルとして関数に渡る
    return

fruit('みかん','りんご') # 二つの引数
```

```
('みかん', 'りんご')
```

```
fruit() #  空の引数
```

```
()
```

```
def route(start, end, *args):
    # 引数からルートのリストを作る
    route_list = [start]    # スタート地点
    route_list += list(args)  # 経由地点は可変のタプル
    route_list += [end]  # ゴール地点
    print(route_list)
    return

route( "京都", "鹿児島", "神戸", "福岡", "熊本")
```

```
['京都', '神戸', '福岡', '熊本', '鹿児島']
```

***args**

- **可変**の**位置引数**をまとめてタプルで受け取れる
  ※*が重要。argsは慣習の変数名

ユーザー定義関数；可変のキーワード引数

```
def entry(name, gender, **kwargs) :
    """ **kwargsで可変のキーワード引数を受けると
        辞書として関数に渡る                    """
    print(name, gender, kwargs)
    # 位置引数から辞書作成
    students = {"name" : name, "gender" : gender}
    # この辞書に可変のキーワード引数をメソッドで追加
    students.update(kwargs)
    return students

# 必須の氏名と性別。オプションの年齢、講座名で呼び出し
result=entry( "安平勲", "男性", age=22, lecture="Python",semester='秋')
print(result)
```

```
安平勲 男性 {'age': 22, 'lecture': 'Python', 'semester': '秋'}
{'name': '安平勲', 'gender': '男性', 'age': 22, 'lecture': 'Python', 'semester': '秋'}
```

**kwargs

- **可変**の**キーワード引数**をまとめてディクショナリで受け取れる
  ※**が重要。**kwargs**は慣習名

ユーザー定義関数；変数のスコープ

```
def calc() :
    v = 10  # vはcalc関数のローカル変数
    answer = 3 * v
    print(answer)

# calc()を実行する
calc()
print(v)   # 関数の外からローカル変数にアクセスできない
```

```
6

---------------------------------------------------------
NameError                                 Traceback (mo
<ipython-input-2-de236229fd28> in <module>
      6 # calc()を実行する
      7 calc()
----> 8 print(v)
      9

NameError: name 'v' is not defined
```

ローカル変数と**グローバル変数**

- ローカル変数は関数内で定義した変数のこと
- 関数の外からローカル変数を**アクセスできない**

kcg.edu

ユーザー定義関数；変数のスコープ

```
v = 10    # vはグローバル変数
def calc() :
    ans = 3 * v    # 変数vを利用する
    print(ans)

calc()
```

```
30
```

```
v = 10    # vはグローバル変数
def calc() :
    v = v * 10    # グローバル変数vを10倍にする
    ans = 3 * v
    print(ans)

calc()
```

```
---------------------------------------------------------------
UnboundLocalError                         Traceback (most recent
<ipython-input-7-ba799158b81c> in <module>
      5     print(ans)
      6
----> 7 calc()

<ipython-input-7-ba799158b81c> in calc()
      1 v = 10    # vはグローバル変数
      2 def calc() :
----> 3     v = v * 10    # グローバル変数vを10倍にする
```

ローカル変数と**グローバル変数**

- グローバル変数は呼出し元で定義した変数のこと
- 関数内ではグローバル変数を**更新できない**。但し，参照は可能
  ※Javaでは参照も不可

ユーザー定義関数；変数のスコープ

```
v = 10     # vはグローバル変数
def calc() :
    global v        # 関数内でも v を定義する宣言
    v = v * 10      # グローバル変数vを10倍にする
    ans = 3 * v
    print(ans)

calc()
```

```
300
```

※前頁のエラーの解決法

- 関数内でグローバルglobal宣言する

✓ このやり方は禁じ手（Javaではできない）
✓ 一般に呼出し元と関数間では，引数以外の変数をやり取りしない（保守性が悪くなる）

ユーザー定義関数；docstring

```
def route(start, end, *args):
    '''引数からルートのリストを作る関数
       第1引数に起点、第2引数に終点を設定
       第3引数以降に複数の経由地を設定'''
    route_list = [start]     # スタート地点
    route_list += list(args)  # 経由地点は可変のタプル
    route_list += [end]  # ゴール地点
    print(route_list)

#  route関数のdocstring を確認する
help(route)
```

```
Help on function route in module __main__:

route(start, end, *args)
    引数からルートのリストを作る関数
    第1引数に起点、第2引数に終点を設定
    第3引数以降に複数の経由地を設定
```

- 関数の一行目の引用符による文字列（コメント）が **docstring**
- 他の人がhelp(*関数名*) 関数でdocstringを読める

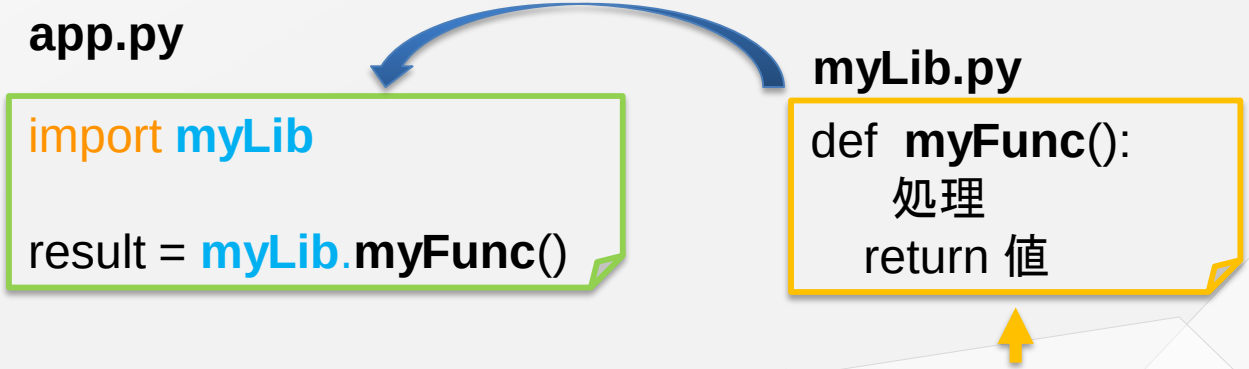
- ✓ 他の人に関数の仕様を読んでもらうために，docstringを書く習慣をつける

コーディングスタイル；Python チュートリアル抜粋

- 関数やクラスや関数内の大きめのコードブロックの区切りに空行を使うこと
- 可能なら，コメントは行に独立で書くこと
- docstring を使うこと
- 演算子の前後とコンマの後には空白を入れ，括弧類のすぐ内側には空白を入れないこと: a = f(1, 2) + g(3, 4)
- クラスや関数に一貫性のある名前を付けること。慣習では UpperCamelCase をクラス名に使い， lowercase_with_underscores を関数名やメソッド名に使います。常に self をメソッドの第 1 引数として使うこと（※別の回で説明）
- あなたのコードを世界中で使ってもらうつもりなら，風変りなエンコーディングは使わないこと。どんな場合でも，Python のデフォルト UTF-8 またはプレーン ASCII が最も上手くいきます

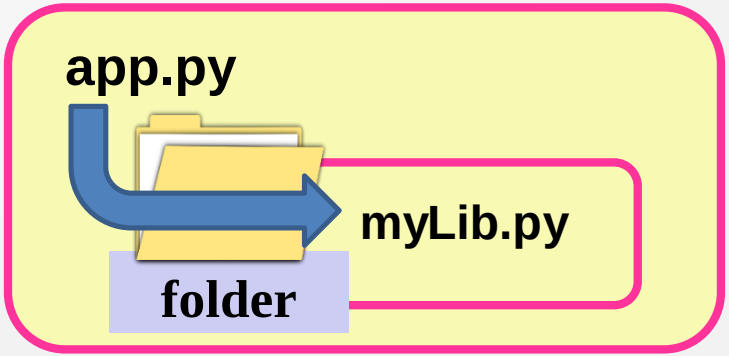
関数モジュール（このコードはspyderで！）

- 関数定義を実行プログラムとは別のファイルに作成できる
- 関数定義を別ファイルにする場合，実行側のプログラムでは，標準関数と同様に，**import**文を使う



関数モジュールと相対パス

- 関数モジュールの場所が**実行プログラムと異なる**場合，**関数モジュールの相対パス**をfromで記述する必要がある
  - **from** *フォルダの相対パス* **import** *モジュールファイル名*
    - 実行プログラムの下位フォルダに関数モジュールを置くと・・



app.py

```
from folder import myLib

result = myLib.myFunc()
```

```
from folder.myLib import myFunc

result = myFunc()
```

第７回演習課題（宿題）

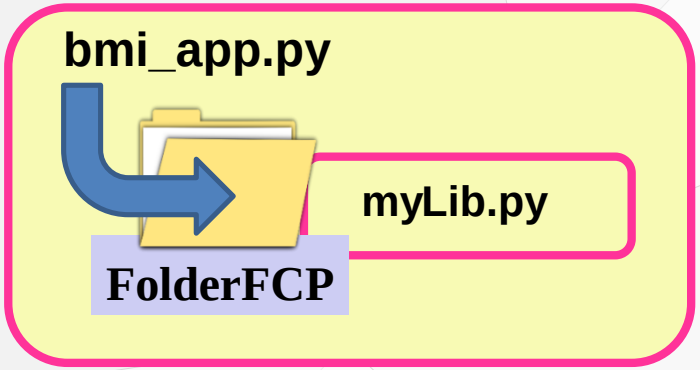
５．BMIの関数定義プログラムを関数モジュールに分離し，下記のフォルダ構成で動作するように修正・確認しよう

```
def bmi(height,weight):
    """BMIを計算･表示する関数
     入力は身長と体重"""
    bmi = weight / height**2 * 10000
    bmi = round(bmi, 1)
    if bmi < 18.5:
        message = "痩身"
    elif 25.0 <= bmi:
        message = "肥満"
    else:
        message = "標準体重"
    print("あなたの肥満度は",bmi,"で",message,"です")

print("あなたの肥満度を計算します")
weight = int(input("体重をkgで入力:"))
height = int(input("身長をcmで入力:"))
bmi(height,weight)  # 関数呼び出し
```

関数定義のファイル名を **myLib.py** とする。そして **FolderFCP** の下に置く

呼出し元のファイル名を **bmi_app.py** とする

フォルダ名：*学籍番号*_task7



ラムダlabmda式，無名関数

```
def cylinder_volume(height,radius) :
    pi = 3.14
    volume = (radius ** 2) * pi  * height
    return volume

# 高さ５，半径10の円柱の体積は
cylinder_volume(5,10)
```

```
1570.0
```

- ラムダ式（無名関数）
  ※短い関数を定義(**def**)なしの１文で書ける
- **Lambda *引数1,引数2 : 命令文***

```
# lambda関数
cylinder_volume =lambda height,radius : (radius ** 2) * 3.14  * height

# 高さ５，半径10の円柱の体積は
cylinder_volume(5,10)
```

```
1570.0
```

関数オブジェクト

```
def hello() :
    print("ハロー！ハロー！")

msg = hello     # 変数に関数名を代入する
msg()           # 変数が示す関数を実行する
msg
```

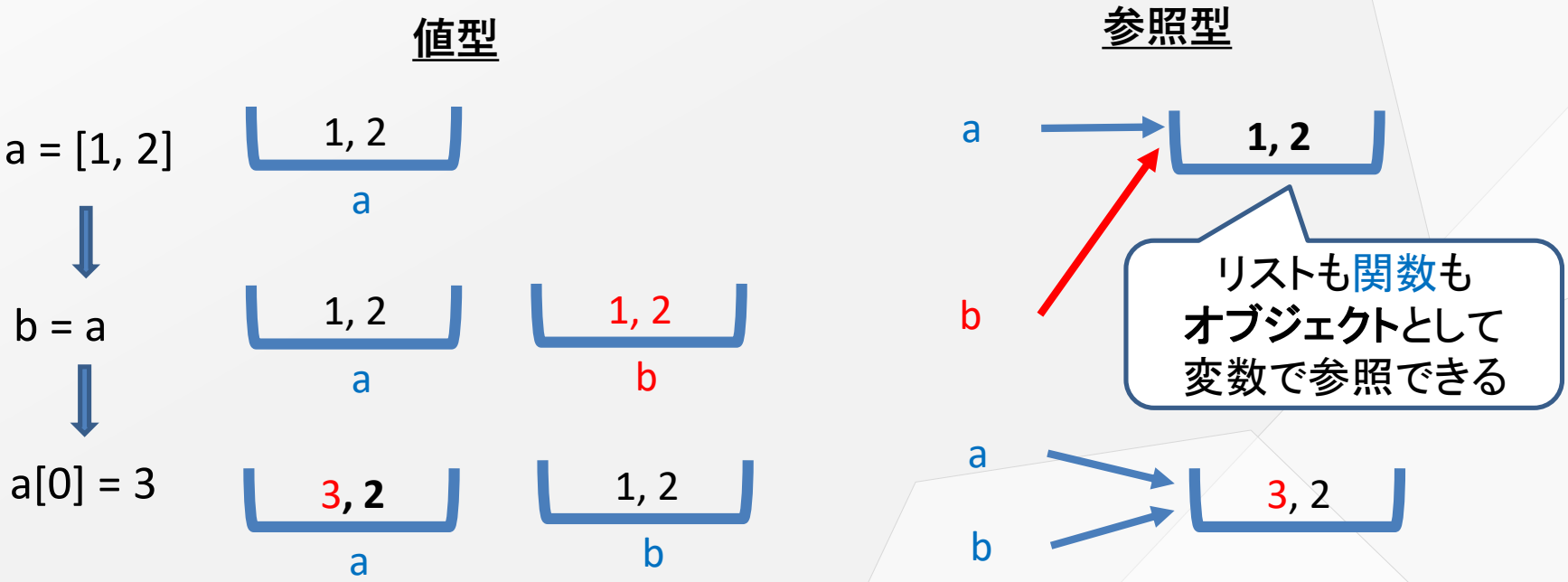
```
ハロー！ハロー！

<function __main__.hello()>
```

- **関数名を変数に代入←オブジェクト参照**
- 関数もオブジェクト（箱）なので変数（箱の名前）で関数を呼べる

✓ 変数に（）つけると関数呼び出し。（）を省略すると元の関数” hello“オブジェクトを示す

（参考）値型と参照型

- 一般的に，変数には値型と参照型の2種類がある。Pythonのlist変数は参照型



関数を呼び出す関数

```
# 関数を呼び出す関数
def do(func) :
    func()    # 引数で受け取った関数を実行する

def hello() :
    print("ハロー！")

def thanks() :
    print("ありがとう")

def hi() :
    print("やあ！")

do(hello)
do(thanks)
do(hi)
```

```
ハロー！
ありがとう
やあ！
```

- 関数(名)を他の関数の引数（関数オブジェクト）として渡せる
  ※do()は引数の関数オブジェクトを実行する

- ✓ （）の有無に注意
- ✓ Pythonは；
  - *関数名*()：関数を呼び出し（実行）
  - *関数名*のみ：オブジェクト参照

- ✓ 関数オブジェクトは、クロージャーなどスマホアプリでのGUI作成のための高度＆必須なテクニックに使われている

（参考）関数を呼び出す関数その２

```
# 関数を呼び出す関数 2
def calc(func, arg) :
    price = func(arg) # 渡された関数名と引数で関数を実行
    return price

def child(num) :        # 子供料金を計算する関数
    return 600 * num

def adult(num) :        # 大人料金を計算する関数
    print("身分証明書を提示してください")
    return 1200 * num

# 年齢によって料金計算の関数を変える
age = int(input('年齢入力：'))
number = int(input('人数入力：'))
if age<16 :
    price = calc(child, number)
else :
    price = calc(adult, number)
print(f"{age}歳、{number}人は{price}円です。")
```

```
年齢入力：20
人数入力：4
身分証明書を提示してください
20歳、4人は4800円です。
```

（参考）クロージャー（関数内関数）

```
# クロージャの定義
def charge(price) :
    # 関数内関数
    def calc(num) :
        return price * num # 料金と個数の掛け算
    return calc

# クロージャ（関数オブジェクト）を2種類作る
child = charge(600)     # 子供料金600円
adult = charge(1200)    # 大人料金1200円

# 料金を計算 ⇒　クロージャ呼び出し
a_number = int(input('大人の人数入力：'))
print("大人",a_number,"人は",adult(a_number),"円")

c_number = int(input('子供の人数入力：'))
print("子供",c_number,"人は",child(c_number),"円")
```

```
大人の人数入力：2
大人 2 人は 2400 円
子供の人数入力：4
子供 4 人は 2400 円
```

- 関数内関数の外側の関数オブジェクトが**クロージャー**
- クロージャーに（）をつけると内側の関数が実行される