Python for bioinformatics

どんぐり研究所　孫 建強

コンテンツ

基本文法

- 予約語
- 条件構文
- 繰り返し構文
- 関数
- パッケージ

基本文法

- ◯ 予約語
- ◯ 条件構文
- ◯ 繰り返し構文
- ◯ 関数
- ◯ パッケージ

予約語

条件判定		条件構文	繰り返し構文	関数		例外処理	いろいろ	
True	is	if	for	def	import	try	lambda	async
False	not	elif	while	return	from	except	del	await
None	in	else	pass	yield	as	finally	zip	global
	or		break		class	raise	assert	with
	and		continue				nonlocal	
			enumerate					

基本文法

- ○ 予約語
- ○ 条件構文
- ○ 繰り返し構文
- ○ 関数
- ○ パッケージ

条件構文

条件構文は、条件に応じて処理を切り分けたい場合に利用する構文の一つである。例えば、「晴れていれば公園に行くが、それ以外ならば家にこもる」、「パン屋さんでパンを購入して、イートインならば消費税 10%、持ち帰りならば消費税 8% にする」などのようなことを表現したりする際に利用する。これらのことをプログラミング言語らしく表現すると右のようになる 。

```
weather = 'sunny'


if weather is 'sunny',
    go_to_park()

otherwise,
    stay_home()
```

```
where_to_eat = 'takeout'
amount = 100

if where_to_eat is 'eatin',
    tax = 0.10

otherwise,
    tax = 0.08

amount = amount * (1 + tax)
```

イートインスペースのあるパン屋さんで、アップルパイを 1 つとメロンパンを 1 つ購入した。合計いくら支払えばよいか。ただし、店内で食べる場合の消費税率を 10% とし、持ち帰りの場合の消費税率を 8% とする。

アップルパイ	180
メロンパン	120
小計	300

takeout

300 × 1.08 = 324

eatin

300 × 1.10 = 330

if 構文

Python の条件構文は if、elif、else などの単語（予約語）を使う。条件構文は必ず if から始まる。条件構文の 1 行目には、if とともに条件を書く。条件判定後の処理は、2 行目以降に書く。ただし、条件判定後の処理は、条件構文の一部であることを明示するために、行の先頭にインデントを入れる。

もし持ち帰りならば ▶

もし店内で食事するならば ▶

```
a = 180
b = 120
s = 0


takeout = True

eatin = False


if takeout is True :
    s = (a + b) * 1.08


if eatin is True:
    s = (a + b) * 1.10


s
# 324
```

if 構文

```
a = 180
b = 120
s = 0
takeout = True

if takeout is True :
    s = (a + b) * 1.08

print(s)
```

判定条件

条件構文の開始 ▶

インデント

条件構文の終了 ▶

条件構文ブロック

インデントの数で、条件構文ブロックが終了しているかどうかを判定している。

インデント

インデントは、タブまたはスペースキーで入力する。

- タブ 1 個分
- スペース 4 個分 ✓
- スペース 2 個分

if 構文

```
a = 180
b = 120
s = 0
takeout = True

if takeout is True :
    s = (a + b) * 1.08

print(s)
```

判定条件

条件構文の開始 ▶

インデント

条件構文の終了 ▶

真判定の場合は、「is True」を省略するのが正しい書き方。

```
a = 180
b = 120
s = 0
takeout = True

if takeout:
    s = (a + b) * 1.08

print(s)
```

論理演算子

条件構文は、与えられた条件が真（True）であるかどうかを判定して、分岐処理を行う構文である。条件は数式などで与えるのが一般的である。

演算子	処理
a == b	a と b が等しいならば True
a != b	a と b が等しくなければ True
a > b	a が b よりも大きければ True
a >= b	a が b 以上ならば True
a < b	a が b よりも小さければ True
a <= b	a が b 以下ならば True

※表中の演算子のほか、is と is not も理論演算に使われる。"a is b" は a と b が同じ属性（かつ同じ値）を持ったオブジェクトである場合に True を返す演算子である。気になる方は、各自で調べてください。

```
a = 4
b = 3
c = 0
d = 0



if a > 3:
    c = 1
c
# 1



if b > 10:
    d = 1
d
# 0
```

問題 S1-1

5 min

次のプログラムを実行した時に、最後に出力される金額 n はいくらになるのかを答えよ。

```
apple = 200
melon = 100
takeout = True

n = apple + melon

if takeout:
    n = n * 1.08


n
```

```
apple = 200
melon = 100
takeout = True

n = apple + melon

if n > 1000:
    n = n * 0.95

if takeout:
    n = n * 1.08


n
```

問題 S1-2

5 min

次のプログラムを実行した時に、最後に出力される金額 n はいくらになるのかを答えよ。

```
apple = 200
melon = 100
takeout = True

n = apple * 4 + melon * 2


if n > 1000:
    n = n * 0.95

if takeout:
    n = n * 1.08


n
```

```
apple = 200
melon = 100
takeout = False

n = apple * 4 + melon * 2


if n > 1000:
    n = n * 0.95

if takeout:
    n = n * 1.08


n
```

問題 S1-3

10 min

イートインスペースのあるパン屋さんで、アップルパイを 1 つとメロンパンを 1 つ、購入して持ち帰る。商品購入時に、可能であればクーポン券を使う。このとき、合計金額を計算せよ。

ただし、

- アップルパイは 1 つ 180 円、メロンパンは 1 つ 120 円である。
- クーポン券は、合計金額が 1000 円を超えた場合のみに、5% OFF される。
- 店内で食べる場合の消費税率を 10% で、持ち帰りの場合の消費税率を 8% とする。

```
apple = 180
melon = 120
takeout = True
eatin = False
```

入れ子構造

複数の条件に対して条件判定を行うとき、条件構文を二つ重ねた入れ子構造にすることで実現できる。例えば、「毎月 20 日に 1000 円以上の買い物を行った時に 10% 値引きする」といった処理は、まず、会計日が 20 日かどうかを判定して、次に、会計日が 20 日であれば購入金額が 1000 円以上かどうかを判定する。これを if 構文で書くと右のようになる 。

```
apple = 200
melon = 100

date = 20


n = apple * 4 + melon* 2


if date == 20:
    if n >= 1000:
        n = n * 0.90
```

条件構文（if）の中の条件構文（if）の中の処理であるため、インデントは 2 つ文必要。

```
n
# 900.0
```

論理演算

複数の条件を同時に判断するときは、入れ子構造の条件構文を利用するほか、複数の条件を論理演算した結果を条件構文の判定条件として使うこともできる。論理演算でよく使われる演算として、論理積と論理和である。論理積は、英語の AND に相当するものである。論理和は、英語の OR に相当するものである。

論理演算子	演算
and	論理積
&	論理積
or	論理和
|	論理和
^	排他的論理和

```
apple = 200
melon = 100
date = 20

n = apple * 4 + melon* 2

if date == 20:
    if n >= 1000:
        n = n * 0.90

n
# 900.0

m = apple * 4 + melon* 2

if (date == 20) and (m >= 1000):
    m = m * 0.90

m
# 900.0
```

if 構文の入れ子構造

二つの判定条件を論理演算している

論理演算

条件 1	条件 2	論理積	論理和	排他的論理和
x	y	x and y	x or y	x ^ y
True	True	True	True	False
True	False	False	True	True
False	True	False	True	True
False	False	False	False	False

問題 S1-4

3 min

下に示した 2 種類の割引処理の違いを、論理演算子に注意しながら、説明してみてください。それぞれがどのような条件で割引されるのか。

```
coupon = True

s = 1100




if (coupon)  and  (s >= 1000) :

    s = (a + b) * 0.95

s
```

```
coupon = True

s = 1100




if (coupon)  or  (s >= 1000) :

    s = (a + b) * 0.95

s
```

問題 S1-5

3 min

赤色で書かれたオブジェクトが保持している値を答えよ。

```
a = 4
b = 3
c = 0

if (a > 0) and (b > 0):
    c = 1

c
```

```
a = 1
b = 4
c = 0

if (a > 0) or (b < 2):
    c = 1

c
```

if-else 構文

これまでに見てきた条件構文は「もし〜ならば・・・をする」のように、ある条件を満たせば、特定の 1 つの処理を行うものであった。これに対して、ある条件を満たしたときに処理 A を、満たさなかった時に処理 B を行いたい場合は、if 構文を続けて 2 つ書けば実現できる。

```
a = 180
b = 120
s = 0


takeout = True




if takeout:
    s = (a + b) * 1.08


if not takeout:
    s = (a + b) * 1.10

s
# 324.0
```

if-else 構文

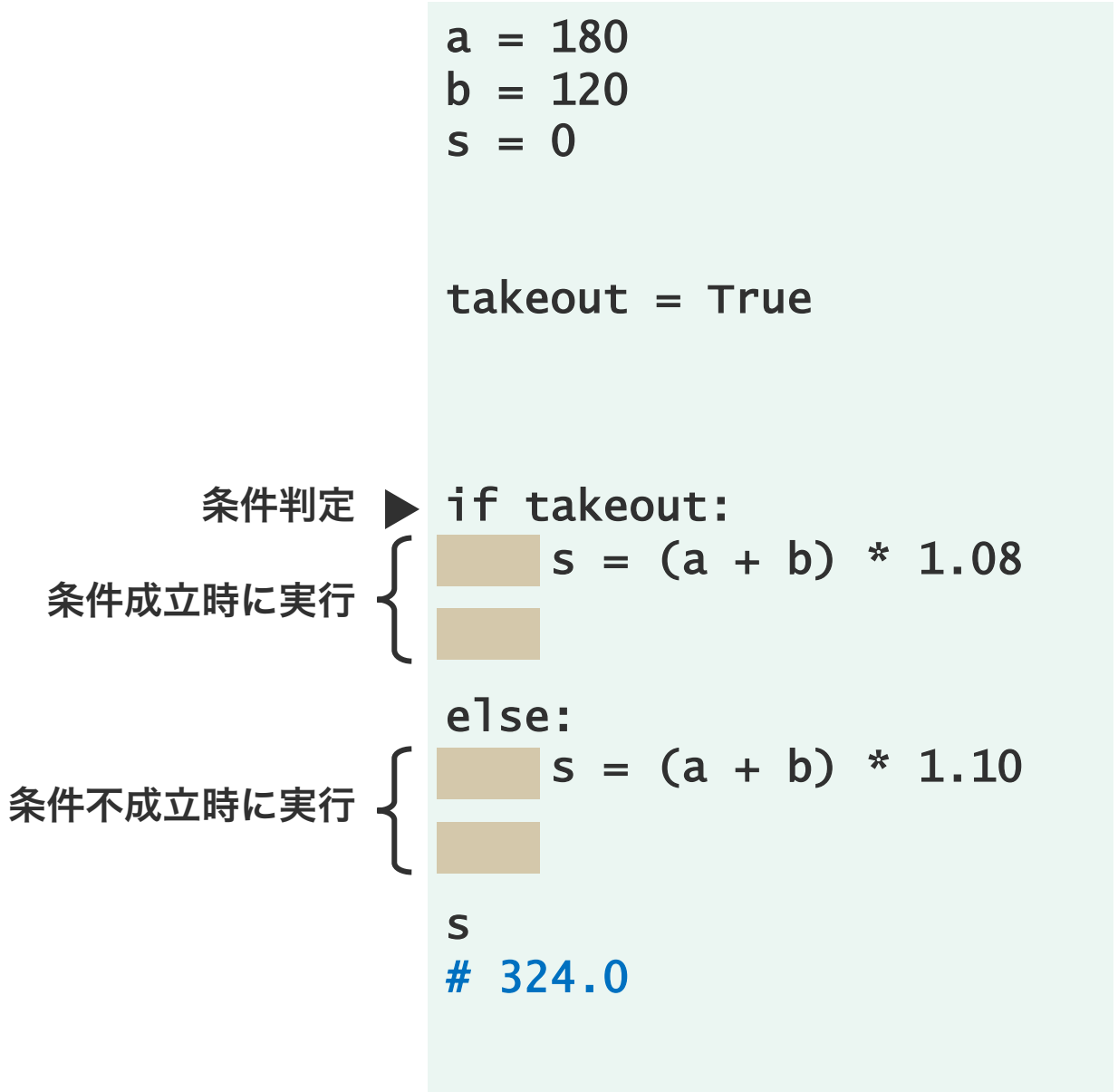
ある条件の真偽判定に基づいて処理を分けたい場合は、条件構文 if を 2 つ書くことで対応できる。しかし、if 構文を 2 つ続けて書くことで、同じ条件を 2 回判断していることになる。理論的にも、処理的にも煩雑である。このような場合は、if と else を用いて条件構文を作成すると、理論的にも処理的にもシンプルになる。

```
a = 180
b = 120
s = 0


takeout = True



if takeout:
    s = (a + b) * 1.08


if not takeout:
    s = (a + b) * 1.10


s
# 324
```

```
a = 180
b = 120
s = 0


takeout = True



if takeout:
    s = (a + b) * 1.08


else:
    s = (a + b) * 1.10


s
# 324
```

if-else 構文

ある条件の真偽判定に基づいて処理を分けたい場合は、条件構文 if を 2 つ書くことで対応できる。しかし、if 構文を 2 つ続けて書くことで、同じ条件を 2 回判断していることになる。理論的にも、処理的にも煩雑である。このような場合は、if と else を用いて条件構文を作成すると、理論的にも処理的にもシンプルになる。



問題 S1-6

15 min

イートインスペースのあるパン屋さんで、アップルパイを 1 つとメロンパンを 1 つ、食パンを 2 袋購入した。このとき、合計金額を計算せよ。なお、商品購入時に、可能であればクーポン券を使用する。

ただし、

- アップルパイは 1 つ 180 円、メロンパンは 1 つ 120 円、食パンは 1 袋 400 円である。
- クーポン券は、合計金額が 1000 円を超えた場合のみに、5% OFF される。
- 店内で食べる場合の消費税率を 10% で、持ち帰りの場合の消費税率を 8% とする。

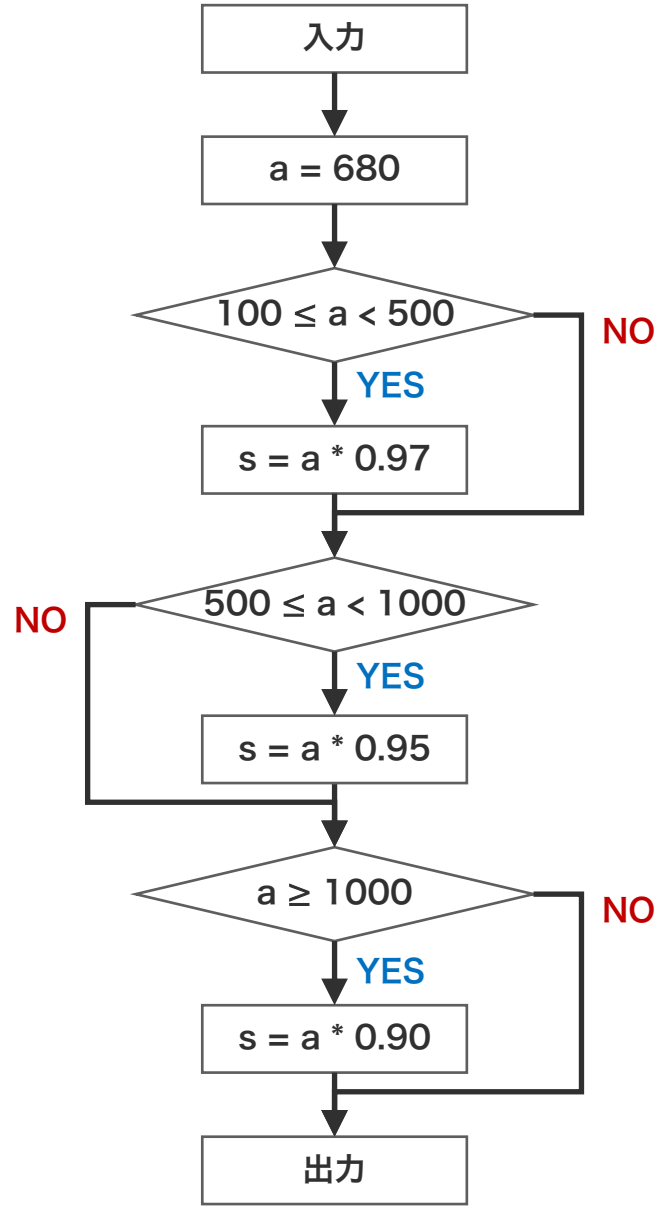
```
apple = 180 * 1
melon = 120 * 1
plain = 400 * 2
```

if-elif-else 構文

1 つの変数に対して複数の閾値を設けて条件判定したい場合は、複数の if 構文を使って書くことができる。しかし、この場合、ロジックも複雑になることに注意する必要がある。

小計	割引
100 円以上 500 円未満	3%
500 円以上 1000 円未満	5%
1000 円以上	10%

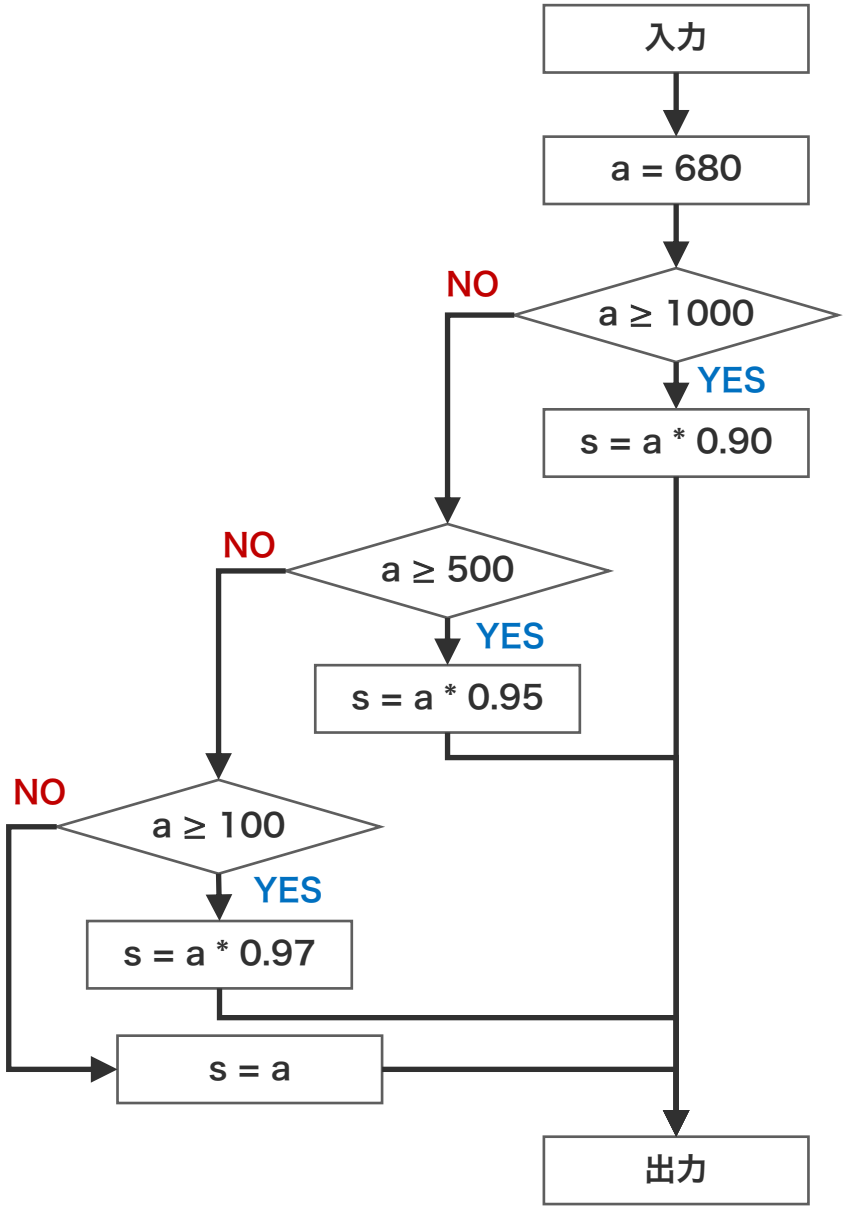
```
a = 680
s = 0


if 100 <= a and a < 500:
    s = a * 0.97

if 500 <= a and a < 1000:
    s = a * 0.95

if a >= 1000:
    s = a * 0.90




s
# 646.0
```



if-elif-else 構文

複数条件で処理を分岐させるときは、複数の if 構文を組み合わせるよりも、if-elif-else 構文を適切に使用した方がロジックもシンプルになる。また、if-elif-else 構文を使用することで、条件の取りこぼしが少なく、デバッグも行いやすい。

小計	割引
100 円以上 500 円未満	3%
500 円以上 1000 円未満	5%
1000 円以上	10%




```
a = 680
s = 0


if a >= 1000:
    s = a * 0.90

elif a >= 500:
    s = a * 0.95

elif a >= 100:
    s = a * 0.97

else:
    s = a


s
# 646.0
```

if-elif-else 構文

複数条件で処理を分岐させるときは、複数の if 構文を組み合わせるよりも、if-elif-else 構文を適切に使用した方がロジックもシンプルになる。また、if-elif-else 構文を使用することで、条件の取りこぼしが少なく、デバッグも行いやすい。

小計	割引
100 円以上 500 円未満	3%
500 円以上 1000 円未満	5%
1000 円以上	10%

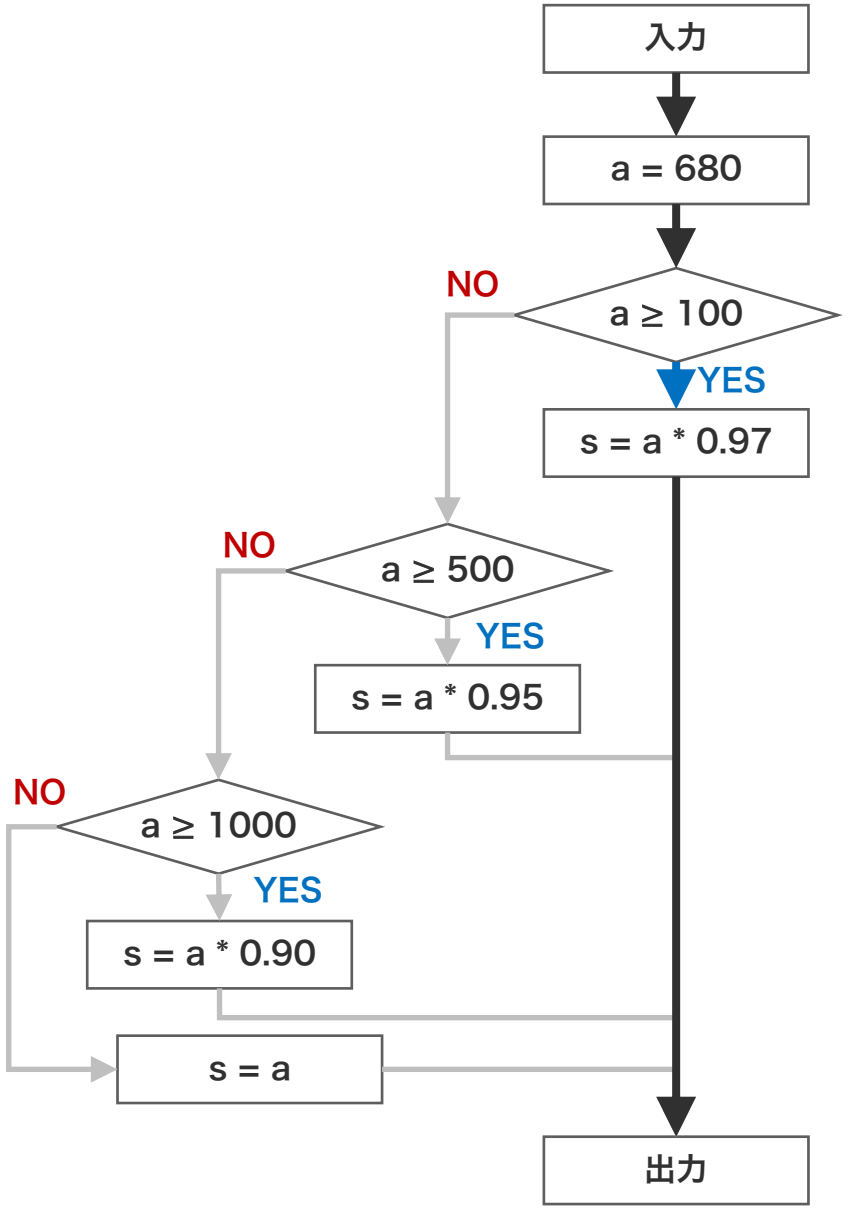
```
a = 680
s = 0


if 100 <= a and a < 500:
    s = a * 0.97

if 500 <= a and a < 1000:
    s = a * 0.95

if a >= 1000:
    s = a * 0.90




s
# 646.0
```

```
a = 680
s = 0


if a >= 1000:
    s = a * 0.90

elif a >= 500:
    s = a * 0.95

elif a >= 100:
    s = a * 0.97

else:
    s = a


s
# 646.0
```

if-elif-else 構文

条件が複数あるとき、各条件の優先順序に注意を払う必要がある。条件の優先順序を間違えると、想定外の処理が行われることがある。

小計	割引
100 円以上 500 円未満	3%
500 円以上 1000 円未満	5%
1000 円以上	10%




```
a = 680
s = 0


if a >= 100:
    s = a * 0.97

elif a >= 500:
    s = a * 0.95

elif a >= 1000:
    s = a * 0.90

else:
    s = a


s
# 659.6
```

問題 S1-7

5 min

赤色で書かれたオブジェクトが保持している値を答えよ。

```
a = 11
s = ''
if (3 <= a) and (a <= 5):
    s = 'spring'

elif (6 <= a) and (a <= 8):
    s = 'summer'

elif (9 <= a) and (a <= 11):
    s = 'autumn'

else:
    s = 'winter'



s
```

```
a = 4
d = ''
if a == 1:
    d = 'Mon'
elif a == 2:
    d = 'Tue'
elif a = 3:
    d = 'Wed'
elif a == 4:
    d = 'Thu'
ellf a == 5:
    d = 'Fri'
elif a == 6:
    d = Sat'
elif a == 7:
    d = 'Sun'
d
```

問題 S1-8

15 min

コドンをアミノ酸に変換する処理を if-elif-else 構文で書け。

```
codon = 'CCG'
```

基本文法

- ○ 予約語
- ○ 条件構文
- ○ 繰り返し構文
- ○ 関数
- ○ パッケージ

繰り返し構文

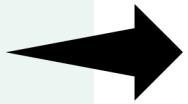
リスト a の各要素の合計を計算して、変数 n に代入する手順を考えよ。ただし、一度に 2 数値の足し算しかできないものとする。

```
a = [2, 3, 1, 5, 10]
n = 0









n
# 21
```

繰り返し構文

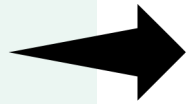
リスト a の各要素の合計を計算して、変数 n に代入する手順を考えよ。ただし、一度に 2 数値の足し算しかできないものとする。

```
a = [2, 3, 1, 5, 10]
n = 0



n = n + 2
n = n + 3
n = n + 1
n = n + 5
n = n + 10




n
# 21
```



```
a = [2, 3, 1, 5, 10]
n = 0



n = n + a[0]
n = n + a[1]
n = n + a[2]
n = n + a[3]
n = n + a[4]

n
# 21
```

i = 0, 1, ..., 4 のとき、n = n + a[i] の繰り返しである。



```
a = [2, 3, 1, 5, 10]
n = 0

for i in range(5):
    n = n + a[i]




n
# 21
```

i を 0, 1, ..., 4 に変化させながら、以下の処理を行う構文。

for 構文

繰り返し構文には while 構文と for 構文の 2 種類がある。このうち、for 構文は「n 回繰り返す」命令文である。for 文を使用するとき、繰り返し回数 n をはじめに指定する必要がある。

```
a = [2, 3, 1, 5, 10]
n = 0

for i in range(5):
    n = n + a[i]

n
# 21



m = 0

for i in range(len(a)):
    m = m + a[i]

m
# 21
```

for 構文

```
a = [2, 3, 1, 5, 10]
n = 0

繰り返し回数　繰り返す範囲
      ↓          ↓
for i in range(5) :
    n = n + a[i]


n
# 21
```

繰り返し構文の開始 ▶

インデント

繰り返し構文の終了 ▶

繰り返し構文ブロック

インデントの数で、構文ブロックが終了しているかどうかを判定している。

for 構文

```
a = [2, 3, 1, 5, 10]
n = 0

for x in a :
    n = n + x


n
# 21
```

一次変数　繰り返す対象

繰り返し構文の開始 ▶

インデント

繰り返し構文の終了 ▶

リストの各要素に対して繰り返し処理を行う場合、繰り返す対象としてリスト名をそのまま与えることが一般的である。

繰り返し構文ブロック

インデントの数で、構文ブロックが終了しているかどうかを判定している。

問題 S2-1

5 min

for 構文を使用して、リスト a の平均値を求めよ。

```
a = [5, 2, 6, 3, 7, 5]

mu = 0

for i in ??? :
    mu = ???




mu
```

問題 S2-2

10 min

リスト z の位置 i の要素が、リスト a の位置 i の要素とリスト b の位置 i の要素の和となるようなリスト z を作成せよ。

```
a = [1, 1, 2, 3, 4, 5]
b = [1, 3, 5, 7, 9, 10]
z = [0, 0, 0, 0, 0, 0]



# z[0] = a[0] + b[0]
# z[1] = a[1] + b[1]
# z[2] = a[2] + b[2]
#   :
#   :









z
# [2, 4, 7, 10, 13, 15]
```

問題 S2-3

5 min

次のプログラムを実行した時に、最後に出力される変数 n の値を答えよ。

```
a = [3, 2, 1, 9, 6, 8]

n = 0


for i in range(len(a)):

    if a[i] % 2 == 1:

        n = n + 2

     else:

        n = n - 1

n
```

```
a = [3, 2, 1, 9, 6, 8]

n = a[0]


for i in range(len(a)):

    if n < a[i]:

        n = a[i]




n
```

問題 S2-4

10 min

リスト a の中の偶数要素の個数を n に代入するプログラムを書け。

```
a = [1, 9, 2, 2, 4, 8, 7, 3]
n = 0




n
```

while 構文

繰り返し構文には while 構文と for 構文の二種類がある。このうち、while 構文は、「与えた条件を満たす限り同じ処理を繰り返す」命令文である。例えば、「リスト a の末端に達するまで、a の各要素を n に足す」処理に利用できる。for 構文で書ける処理は while 構文でも書ける。右は、リスト a の各要素の合計を計算する処理を、for 構文および while 構文で書いた例を示している。

```
a = [2, 3, 1, 5, 10]
n = 0

for i in range(5):
    n = n + a[i]

n
# 21



m = 0

i = 0
while i < 5:
    m = m + a[i]
    i = i + 1

m
# 21
```

while 構文

```
a = [2, 3, 1, 5, 10]
n = 0

i = 0
while i < 5 :
    n = n + a[i]
    i = i + 1

n
# 21
```

判定条件

繰り返し構文の開始 ▶

インデント

繰り返し構文の終了 ▶

繰り返し構文ブロック

インデントの数で、構文ブロックが終了しているかどうかを判定している。

問題 S2-5

10 min

while 構文を使用して、リスト a の平均値を求めよ。

```
a = [5, 2, 6, 3, 7, 5]
len_a = 6

s = 0

i = 0

while ??? :
    s = ???




s
```

問題 S2-6

10 min

while 構文を使用して、リスト a の最大値を m に代入せよ。

```
a = [3, 2, 1, 9, 6, 8]
m = a[0]



while ??? :
    if ??? :
        m = ???




m
```

基本文法

- ○ 予約語
- ○ 条件構文
- ○ 繰り返し構文
- ○ 関数
- ○ パッケージ

関数

同じ処理を繰り返すとき、その処理をまとめてパッケージ化することができる。このパッケージ化された処理群を関数という。自動販売機に例えるとわかりやすい。例えば、お金を入れて、お茶ボタンを押したときにお茶が出て、コーヒーボタンを押したときにコーヒーが出てくる。このとき、自動販売機は、お金というデータとお茶というオプション（引数）を受け取り、それに対応した処理を行って飲み物（戻り値）を出力する関数といえる。

関数

関数を使用しないで、支払い金額を計算するプログラムについて考えてみる。180 円のパンを 1 つ買ったときに支払うべき金額を、持ち帰りの有無・クーポンの使用の有無を考慮して、計算すると、右のようになる。

＊クーポン券使用で、持ち帰りの場合

```
a = 180
takeout = True
use_coupon = True

s = 0

if use_coupon:
    s = a * 0.95

if takeout:
    s = s * 1.08
else:
    s = s * 1.10

s
```

関数

関数を使用しないで、支払い金額を計算するプログラムについて考えてみる。180 円のパンを 1 つ買ったときに支払うべき金額を、持ち帰りの有無・クーポンの使用の有無を考慮して、計算すると、右のようになる。

＊クーポン券使用で、店内で食べる場合

```
a = 180
takeout = False
use_coupon = True

s = 0

if use_coupon:
    s = a * 0.95

if takeout:
    s = s * 1.08
else:
    s = s * 1.10

s
```

関数

関数を使用しないで、支払い金額を計算するプログラムについて考えてみる。180 円のパンを 1 つ買ったときに支払うべき金額を、持ち帰りの有無・クーポンの使用の有無を考慮して、計算すると、右のようになる。

＊クーポン券使用しないで、店内で食べる場合

```
a = 180
takeout = False
use_coupon = False

s = 0

if use_coupon:
    s = a * 0.95

if takeout:
    s = s * 1.08
else:
    s = s * 1.10

s
```

関数

関数を使用しないで、支払い金額を計算するプログラムについて考えてみる。180 円のパンを 1 つ買ったときに支払うべき金額を、持ち帰りの有無・クーポンの使用の有無を考慮して、計算すると、右のようになる。

＊クーポン券使用しないで、持ち帰りの場合

```
a = 180
takeout = True
use_coupon = False

s = 0

if use_coupon:
    s = a * 0.95

if takeout:
    s = s * 1.08
else:
    s = s * 1.10

s
```

関数

関数を使用しないで、支払い金額を計算するプログラムについて考えてみる。140 円のパンを 1 つ買ったときに支払うべき金額を、持ち帰りの有無・クーポンの使用の有無を考慮して、計算すると、右のようになる。

＊クーポン券使用しないで、持ち帰りの場合

```
a = 140
takeout = True
use_coupon = False

s = 0

if use_coupon:
    s = a * 0.95

if takeout:
    s = s * 1.08
else:
    s = s * 1.10

s
```

関数

関数を使用しないで、支払い金額を計算するプログラムについて考えてみる。120 円のパンを 1 つ買ったときに支払うべき金額を、持ち帰りの有無・クーポンの使用の有無を考慮して、計算すると、右のようになる。

＊クーポン券使用で、持ち帰りの場合

```
a = 120
takeout = True
use_coupon = True

s = 0

if use_coupon:
    s = a * 0.95

if takeout:
    s = s * 1.08
else:
    s = s * 1.10

s
```

関数

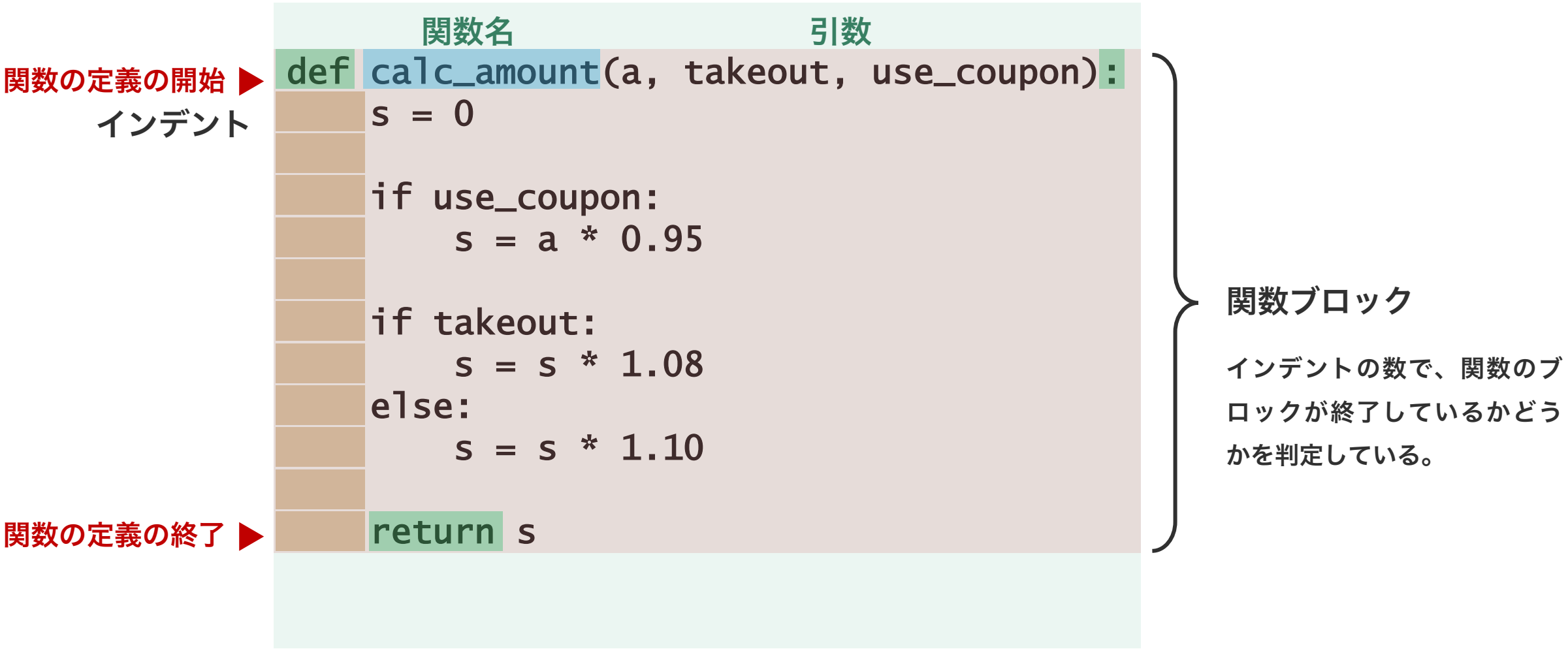
```
a = 120
takeout = True
use_coupon = True

s = 0

if use_coupon:
    s = a * 0.95

if takeout:
    s = s * 1.08
else:
    s = s * 1.10

s
```

実行するたびに変化する値（変数）を入力値として定義。

共通する処理をそのまま移植。

```
def calc_amount(a, takeout, use_coupon):
    s = 0

    if use_coupon:
        s = a * 0.95

    if takeout:
        s = s * 1.08
    else:
        s = s * 1.10

    return s
```

関数


関数名
引数
関数の定義の開始
インデント
def calc_amount(a, takeout, use_coupon):
s = 0
if use_coupon:
s = a * 0.95
if takeout:
s = s * 1.08
else:
s = s * 1.10
return s
関数の定義の終了
関数ブロック
インデントの数で、関数のブロックが終了しているかどうかを判定している。

関数

関数を使うとき、その関数の名前を呼び出せばよい。その際に、関数に入力したい値を、定義した順番通りにカンマ区切りで並べて与える。関数での処理結果を変数に代入したい場合は、代入演算子を使用する。

```
def calc_amount(a, takeout, use_coupon):
    s = a
    if use_coupon:
        s = s * 0.95
    if takeout:
        s = s * 1.08
    else:
        s = s * 1.10
    return s


total_1 = calc_amount(120, True, True)
print(total_1)

total_2 = calc_amount(140, True, False)
print(total_2)

total_3 = calc_amount(160, False, True)
print(total_3)
```

問題 S3-1

15 min

リスト中の要素のうち、奇数の要素の個数を求める関数を作成せよ。

```
a = [1, 7, 4, 8, 3]

def getnodd(x):




getnodd(a)
# 3
```

リスト中の要素のうち、奇数の要素の和を求める関数を作成せよ。

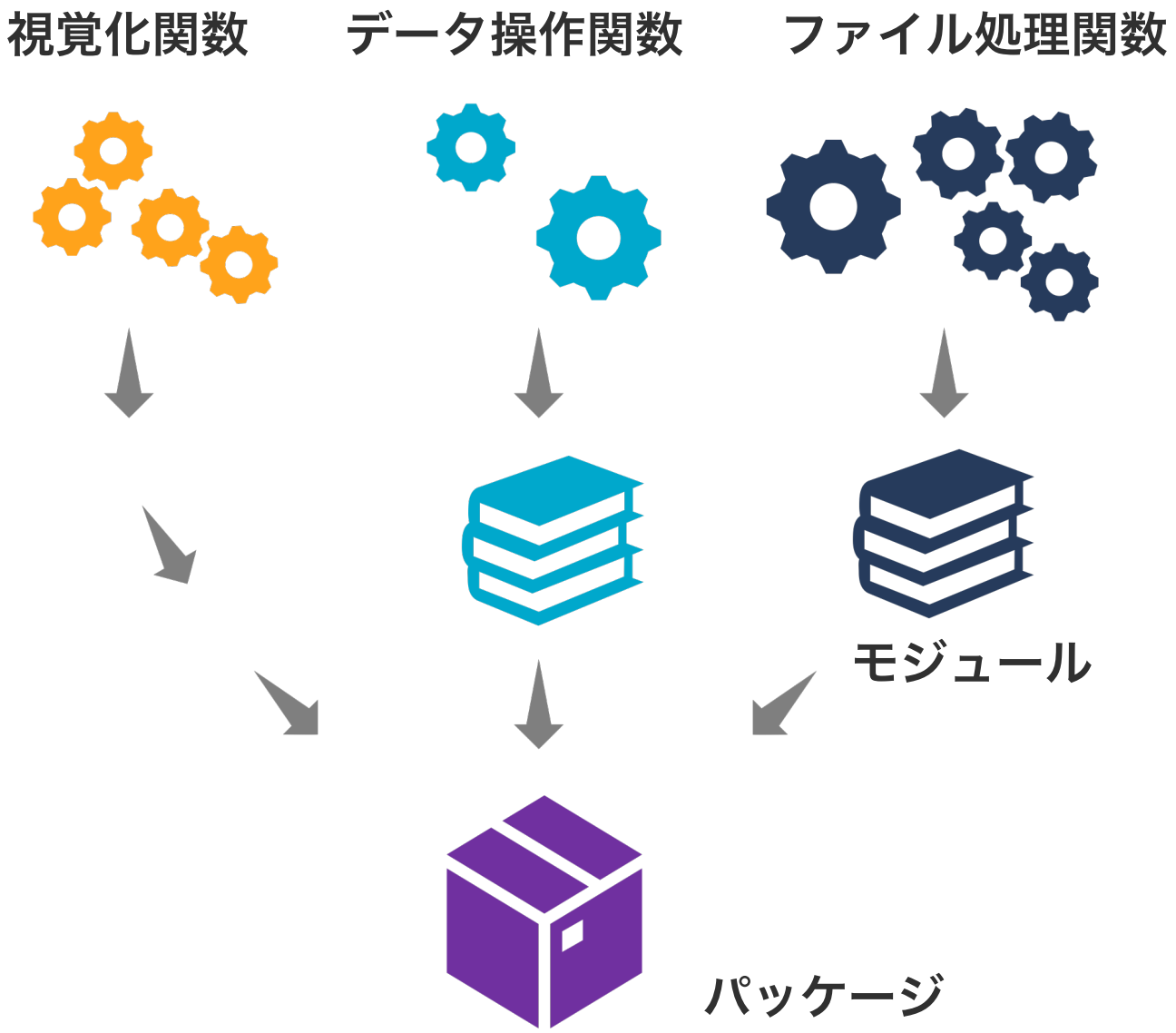
```
a = [1, 7, 4, 8, 3]

def sumodd(x):




sumodd(a)
# 11
```

基本文法

- ○ 予約語
- ○ 条件構文
- ○ 繰り返し構文
- ○ 関数
- ○ パッケージ

パッケージ

いくつかの機能の似た関数をまとめたものをライブラリー（モジュール）と呼ぶ。さらに複数のモジュールをまとめたものがパッケージと呼ぶ。



パッケージ

パッケージは、各開発者や開発団体の公式ウェブサイトあるいは GitHub や PyPI などのレポジトリで配布されている。例えば、PyPI には約 17 万のパッケージが登録されている。PyPI への登録は個人でも簡単に行えるので、研究目的の場合は、あまりマイナーなパッケージを使わないようにすること。

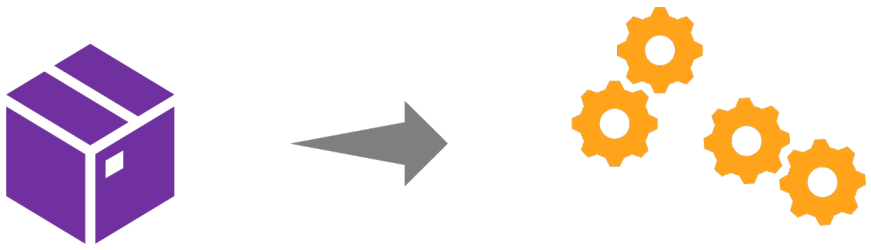
パッケージ

- NumPy
  - ベクトルの高速演算
  - 線形代数
- Pandas
  - データ操作
- SciPy
  - 統計処理
  - 微分方程式・積分
  - パラメーター最適化
- matplotlib
  - 可視化
- Seaborn
  - 可視化
- OpenCV
  - 画像解析
  - 動画解析
  - 機械学習
- scikit-image
  - 画像処理
  - 機械学習
- PIL / Pillow
  - 簡易画像処理
- scikit-learn
  - モデリング
  - 統計的機械学習
- tensorflow
  - 深層学習
- PyTorch
  - 深層学習

パッケージ

パッケージを使うとき、import を使ってパッケージの全機能を読み込むことをできる。例えば、データ処理用ライブラリーの NumPy の全機能を読み込むとき、"import numpy" のようにする。パッケージ中の関数（メソッド）を使うとき、パッケージ名の後に . をつけて、必要な関数を呼び出す。例えば、NumPy 中の array 関数を使いたい場合は、右のようにする。

```
import numpy

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



パッケージ

パッケージ名は、読み込み後に略名を与えることができる。略名は、右のように予約語 as で定義する。略名を定義した後、略名を用いて関数などの呼び出しが可能になる。長いパッケージ名に略名を付けることで、コードが見やすくなる。

```
import numpy as np

a = np.array([1, 3, 5])
b = np.array([2, 4, 6])



import matplotlib.pyplot as plt

fig = plt.figure()
```