プログラミング初級（Python）

分岐処理

早稲田大学グローバルエデュケーションセンター

条件を定めて命令を分岐する

何かを学ぶためには、
自分で体験する以上に良い方法はない。

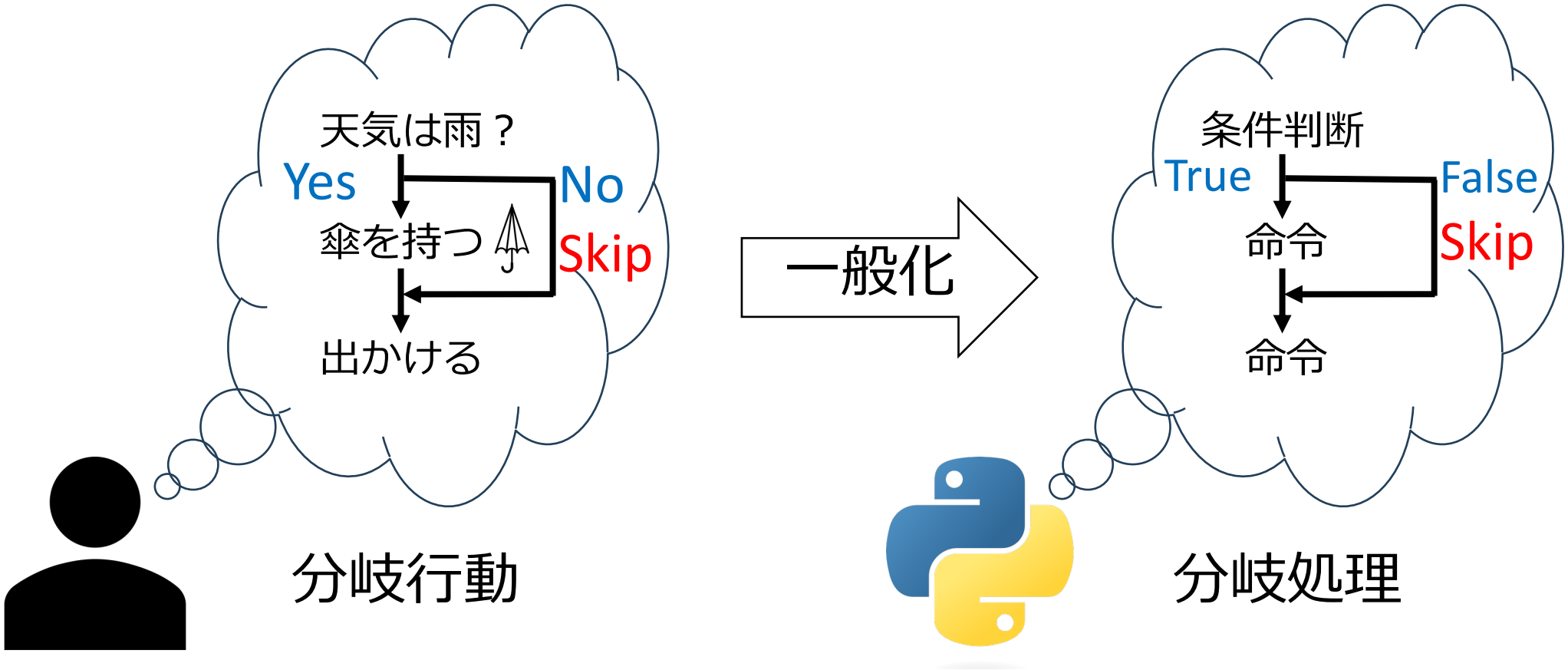
― アルベルト・アインシュタイン

プログラムの処理形態

- これまでは**命令のみ**が書かれたプログラムを扱ってきた。
- これまでは先に書いた命令から後に書いた命令の順にコンピュータが次々と命令を実行する処理を扱った。**書いた命令は必ず実行**される
- これを**順次処理**と呼ぶ。
- プログラムでは、命令だけではなく**コメント**を書くことができる。
- プログラムでは、命令をコメントに変えることにより、**機能を無効化**できる。その行為のことを**コメントアウト**と呼ぶ。
- プログラムでは、**書いた命令を実行せずにスキップ**させることが可能。
- これを**分岐処理**と呼ぶ。条件を指定して分岐を行うため**条件分岐**とも。
- **条件**には、**値**(変数やリテラル)や**式**(論理演算や比較演算)が利用可能。
  例 値が**真なら**命令を**実行**。**偽なら**命令を**スキップ**。
  　　値が**等しいなら**命令を**実行**。**等しくないなら**命令を**スキップ**。

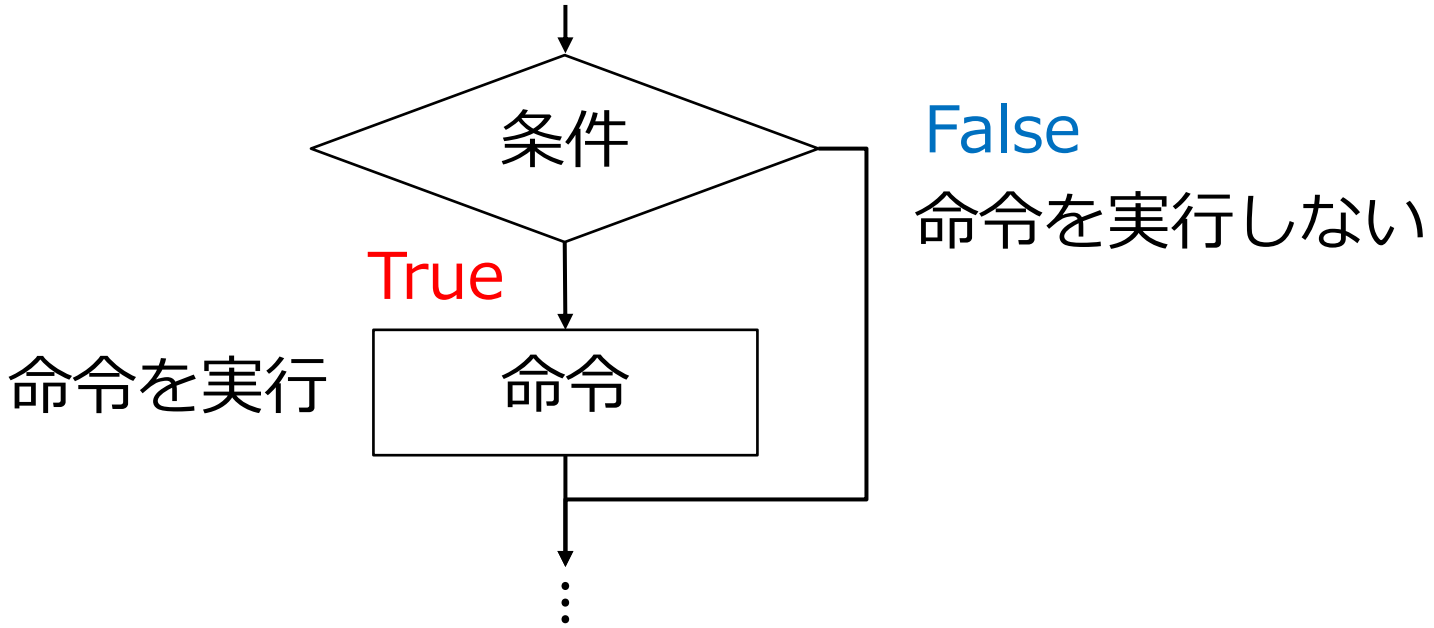
分岐処理の概念イメージ

例：天気が雨という条件が成り立つ(Yes)か否(No)か。
　　コンピュータ風にいうとその条件が**真(True)か偽 (False)か**。



分岐処理のフローチャート

- 条件によって処理する内容が分岐する。
- 下記のようなフローチャートでプログラムの流れをつかむ。



分岐処理(if文) のフローチャート

分岐処理のコードイメージ

```
1 # Below is a branch processing program.
2 rain = True
3 if rain:
4     take_umbrella()
5 go_out()
```

←このプログラムは架空のものでそのまま書いても動かない。

<模擬プログラム>

テキストエディタ等でPythonプログラム(branch.py)を作成

```
1 # Below is a branch processing program.
2 rain = True
3 if rain:
4     print('Taking my umbrella')
5 print('Going out')
```

ターミナル系ソフトでPythonプログラム(branch.py)を実行

```
python Desktop¥branch.py
```

```
python3 Desktop/branch.py
```

← コマンドは一例。真似ても変えても良い。

模擬プログラムを対話型インタプリタで書く場合

```
1  >>> # Below is the branch processing program.
2  >>> rain = True
3  >>> if rain:
   ...     print('Taking my umbrella')
   ...
   taking my umbrella
4  >>> print('Going out')
   going out
5  >>> rain = False
6  >>> if rain:
   ...     print('Taking my umbrella')
   ...
7  >>> print('Going out')
   Going out
8  >>>
```

インデントをスペースかタブつける必要あり

if文が終わったことを示すために改行

分岐処理は実行されない！

💡対話型インタプリタを使うとif文を書き直すのが手間だが、
Pythonの処理が逐次的なのでわかりやすい。

if文とは？

- **if文**とは条件が真となる場合に分岐処理の命令を実行する**構文**。

構文

```
if 条件:
    命令
```

- 条件は **True** (真) か **False** (偽) のいずれかの値をとる。
- 条件が True である場合、if文の分岐命令が実行される。
- 条件の後に`:`（**半角コロン**）**をつける**ルール。
- 分岐命令は何個でもよいがどれにも**字下げ(インデント)をつける**ルール。
- if文の範囲を示すために必ず**字下げ（インデント）をつけるルール**。
- インデントがないとif文の範囲外の命令と判断されたりエラーとなる。
- インデントのサイズは任意だがプログラム内で首尾一貫する必要あり。
- PEP8(https://bit.ly/pep-8)の**推奨スタイル**は**半角スペース4つ**。
- インデントに全角スペースを使用することはできない。

if文を用いたプログラムの例

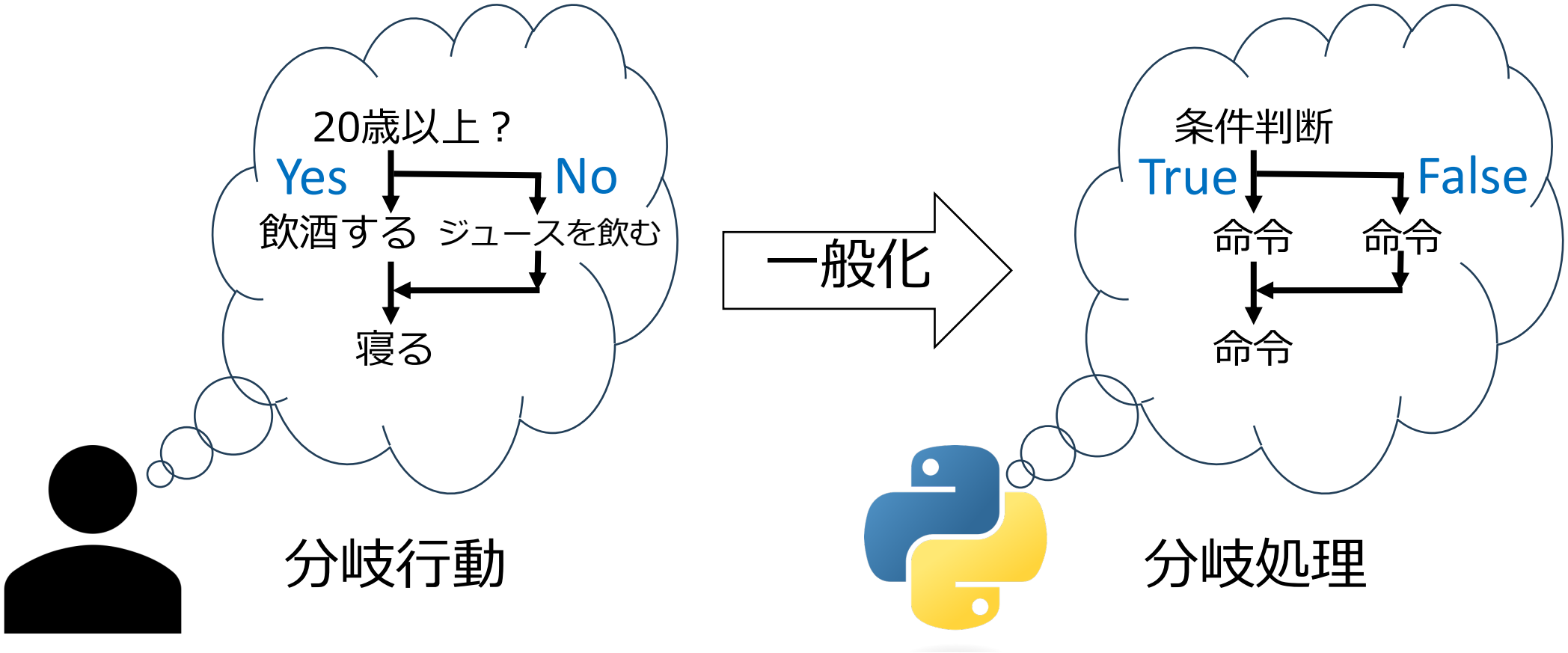
```
1 >>> if True:
  ...     print("It's True.")
  ...
  It's True.
2 >>> if False:
  ...     print("It's False.")
  ...
3 >>> if False
    File "<stdin>", line 1
      if False
              ^
  SyntaxError: expected ':'
4 >>> if False:
  ... print("It's False.")
    File "<stdin>", line 2
      print("It's False.")
      ^
  IndentationError: expected an indented block after 'if' statement on line 1
5 >>>
```

- 条件には式ではなくブーリアンリテラルを直接的に記入することも可能。
- 処理は分岐しないが、プログラムが正しく動作しているか確認する際には有効。
- スペル(呪文ではない)を誤ったり、インデントを誤ったりするとエラーが発生する。

条件が偽の場合にも分岐命令を実行したい場合

条件１：20歳以上の場合、飲酒する。

条件２：20歳未満の場合、ジュースを飲む。



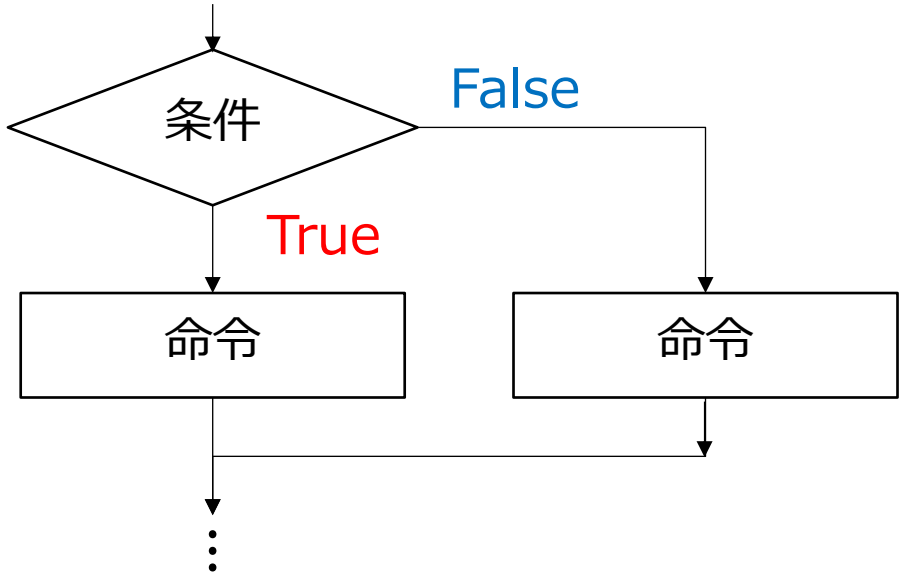
if-else文

- **if-else文**とは条件が真となる場合にif節の命令を実行し、偽となる場合にはelse節の命令を実行する**構文**。

構文

```
if 条件:
    命令
else:
    命令
```

if-else文のフローチャート →



- 条件は必ずTrueかFalseとなるため、if節もしくはelse節のブロック処理が必ず実行される。
- elseの後に条件は記述しないが、ifと同様に`:` の記述が必要。
- if節とelse節ともに命令のインデント数は揃えて記述する。

if-else文を用いたプログラムの例

1
```
>>> if True:
...     print("It's True.")
... else:
...     print("It's False.")
...
It's True.
```
2
```
>>> if False:
...     print("It's False.")
... else:
...     print("It's True.")
...
It's True.
```
3
```
>>> bool = True
```
4
```
>>> if bool:
...     print("It's True.")
... else:
...     print("It's False.")
...
It's True.
```

- 条件にはブーリアンリテラルも指定可能。
- やはりスペルを誤ったり、インデントを誤ったりするとエラーが発生する。

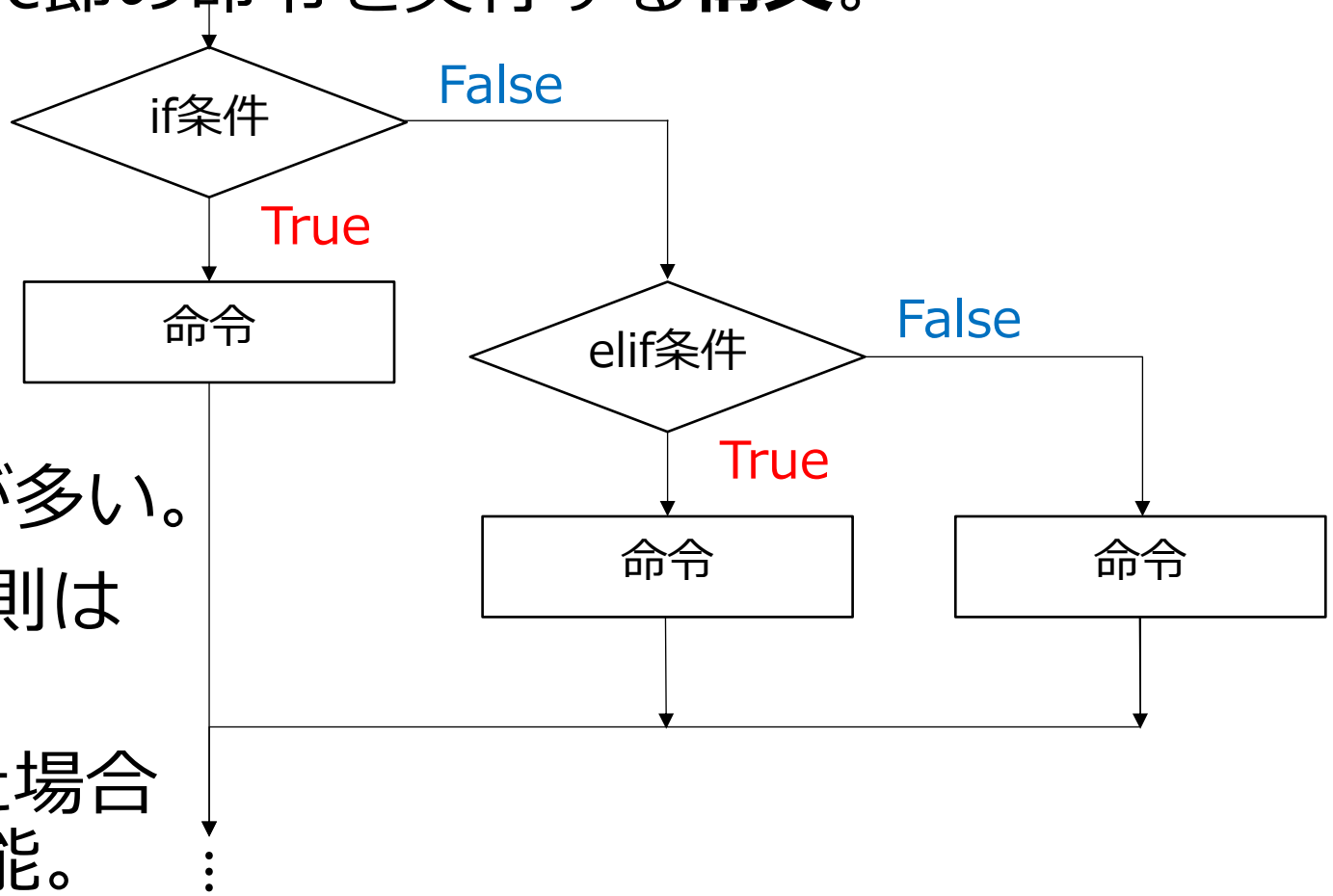
5
```
>>> bool = False
```
6
```
>>> if bool:
...     print("It's False.")
... else:
...     print("It's True.")
...
It's True.
```
7
```
>>> if bool:
...     print("It's False.")
... elsee:
  File "<stdin>", line 3
    elsee:
    ^^^^^
SyntaxError: invalid syntax
```
8
```
>>> if bool:
...     print("It's False.")
... else:
... print("It's True.")
  File "<stdin>", line 4
    print("It's True.")
    ^
IndentationError: expected an indented
block after 'else' statement on line 3
```
9
```
>>>
```

if-elif-else文

- **if-elif-else文**とはif条件が真となる場合にif節の命令を実行し、偽となる場合にはelif節に移動して、elif条件が真の場合にはelif節の命令を実行し、偽となる場合にはelse節の命令を実行する**構文**。

構文

```
if 条件:
    命令
elif 条件:
    命令
else:
    命令
```



if-elif-else文のフローチャート

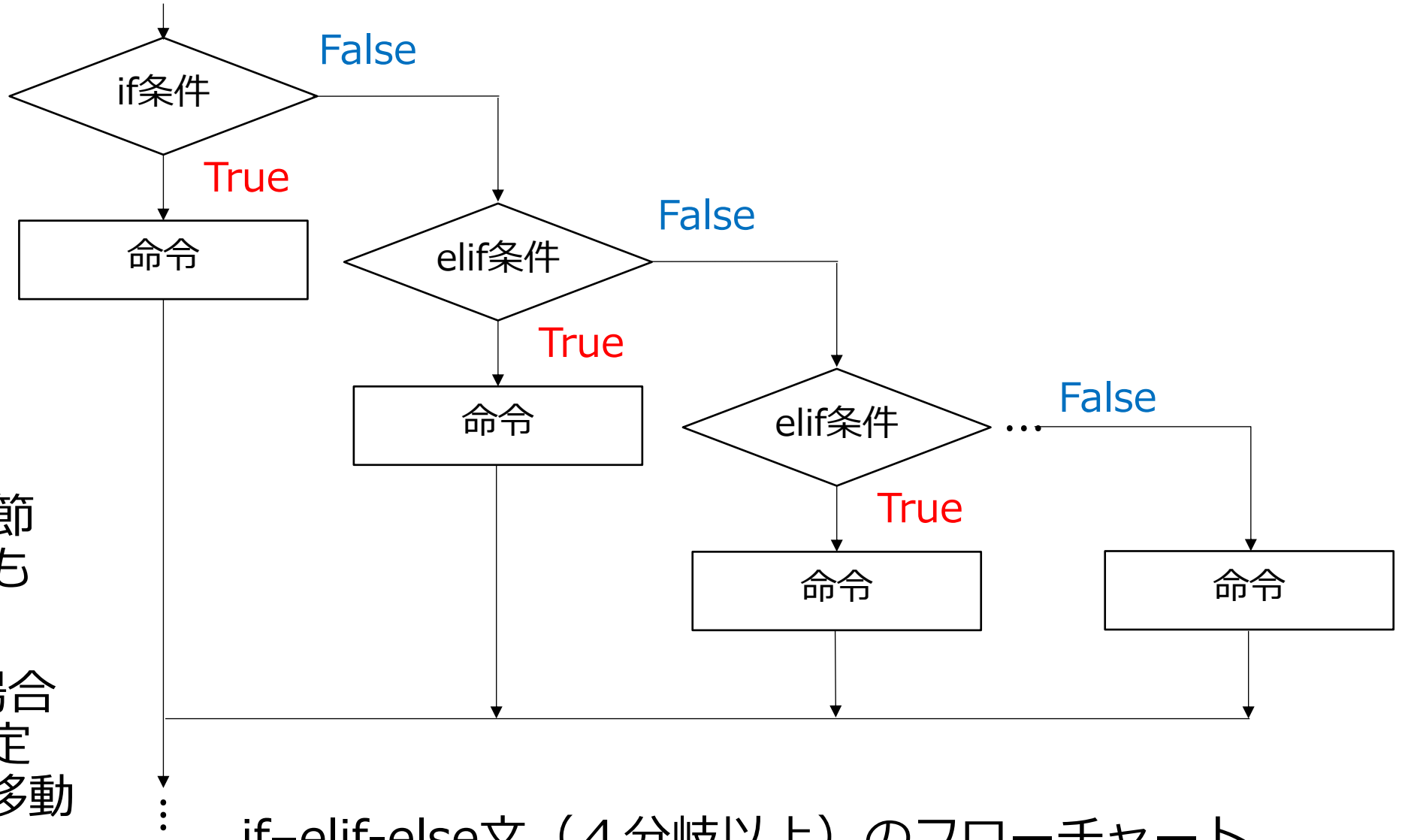
- elifはエルイフと発音する人が多い。
- コロンやインデントの指定規則はif文、if-else文と同様。
- elif節で全ての事象を指定した場合はelse節を省略することも可能。
- elif-else文は存在しない。

if-elif-else文（4分岐以上）

構文

```
if 条件:
    命令
elif 条件:
    命令
elif 条件:
    命令
  ⋮
else:
    命令
```

- elif節はif節とelse節の間に何個書いても良い。
- elif節が複数ある場合は全ての条件を判定してからelse節に移動する。



if-elif-else文（４分岐以上）のフローチャート

if-elif-else文を用いたプログラムの例

```
1  >>> color = "mauve"
2  >>> if color == "red":
   ...     print("It's a tomato.")
   ... elif color == "green":
   ...     print("It's a green peper.")
   ... elif color == "bee purple":
   ...     print("I don't know what it is, but only bees can see it.")
   ... else:
   ...     print(f"I've never heard of the color: {color}.")
   ...
   I've never heard of the color: mauve.
3  >>>
```

- `mauve`は薄紫、`bee purple`は紫外線の範囲の色で人間には見えない。
- 条件として比較演算を利用：「==」は「等しい」という意味。
- 各条件では変数colorが指定した文字リテラルと等しいかどうかを判定している。
- 分岐命令がたくさん存在しても、結局、実行されるのはどれかの１つの分岐のみ。
- 分岐命令は**上からチェック**されていき、条件が真となる分岐があればそこの命令を実行し、**以降の分岐命令はすべてスキップされる**。

典型的な間違い

```
1  >>> color = "mauve"
2  >>> if color = "red":
     File "<stdin>", line 1
       if color = "red":
          ^^^^^^^^^^^^^
   SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
3  >>>
```

比較演算の式「color == "red"」を書くべきところに代入演算の式「color = "red"」を書いた例。上記のようなメッセージでPythonがエラーを教えてくれる。

比較演算子（復習）

演算子	意味	例
==	等しい	a == True
!=	等しくない	a! = True
>	大なり	a > 1
<	小なり	a < 1
>=	大なりイコール	a >= 1
<=	小なりイコール	a <= 1

条件の真偽値

- Pythonは以下の値を偽（False）と判定する。
  - ブール値：False
  - 値の非存在：None
  - 整数のゼロ：0
  - 浮動小数点数のゼロ：0.0
  - 空文字列：''、""など
  - 空タプル：() 、空リスト：[]、空集合：set()、空辞書：{}
    ※タプル、リスト、集合、辞書は後の講義で解説する。

- 上記以外のものは全て真（True）と判定する。
- 算術演算：例　3+1　上記に従い、True。
- 論理演算：ブール代数と上記に従う。
- 比較演算：color == "red"　上記に従い変数の値に応じてTrueもしくはFalse。
- 代入演算：上記に従うが、最終的にPythonがエラーとして処理してくれる。

比較演算結果を確認 左辺と右辺の2つの関係が真 (True) か偽 (False) かを比較して評価する演算。

```
>>> a = 20
>>> b = 10
>>> a < b
False
>>> a <= b
False
>>> a > b
True
>>> a >= b
True
>>> a == b
False
>>> a != b
True
>>> a = 5
>>> b = 30
>>> a < b
True
>>> a <= b
True
>>> a > b
False
>>> a >= b
False
```

```
>>> a == b
False
>>> a != b
True
>>> a = 15
>>> b = 15
>>> a < b
False
>>> a <= b
True
>>> a > b
False
>>> a >= b
True
>>> a == b
True
>>> a != b
False
>>> 30 < a
False
>>> 3 + 1 < 3 + 2
True
```

- 記法：通常は**左辺に変数を右辺にリテラル**を書く。
- 優先順位：比較演算は、算術演算よりも低い。

input関数とは？

- input関数はユーザからのキーボード入力を受け取るために利用する。

書式 `input(prompt)`

- input関数はプログラムの実行を一時停止し、ユーザがキーボードから入力したテキストを文字列として取得する。
- promptは省略可能。
- 変数をinput関数に関連づけるとユーザの入力した値を保存できる。

```
1  >>> name = input("Please enter your name: ")
   Please enter your name: Python
2  >>> print("Hello, " + name + "!")
   Hello, Python!
3  >>> name
   'Python'
4  >>>
```

「Please ～ name： 」がプロンプト。**「Python」がユーザの入力した値。**
入力を終えたら **Enter キー**を押す。すると処理が先に進む。

値の**型は文字列**として保存される。

input関数と型変換（int型とfloat型）

```
>>> # Prompt for an integer and convert it to int type
>>> user_input = input("Please enter an integer: ")
Please enter an integer: 3
>>> integer_value = int(user_input)
>>> print(1 + integer_value)
4
>>> integer_value = user_input
>>> print(1 + integer_value)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> # Prompt for a floating-point number and convert it to float type
>>> user_input = input("Please enter a floating-point n
umber: ")
Please enter a floating-point number: 3.14
>>> float_value = float(user_input)
>>> print(3 * 3 * float_value)
28.26
>>> float_value = user_input
>>> print(3 * 3 * float_value)
3.143.143.143.143.143.143.143.14
```

input関数で入力した値を算術演算に使う場合は型変換が必要。数値を入力したつもりでも数字として保存される。

半径 * 半径 * 円周率で円の面積を求める。

この例はエラーにならないが、意図しない動作となったことを示す。

input関数の実行結果をワンライナーで型変換

```
13 >>> integer_value = int(input())
   3
14 >>> integer_value
   3
15 >>> float_value = float(input())
   3.14
16 >>> float_value
   3.14
17 >>>
```

一時変数を省略して、１行で書いても良い。

一時変数を省略して、１行で書いても良い。

分岐処理を並列に記述

```
1  >>> a = int(input())
   10
2  >>> if a > 0:
   ...     judge = True
   ... else:
   ...     judge = False
   ...
3  >>> if judge:
   ...     print('It is a positive value.')
   ... else:
   ...     print('It is either 0 or a negative value.')
   ...
   It is a positive value.
4  >>> print('The program will exit.')
   The program will exit.
5  >>>
```

1つ目のif-else文。

2つ目のif-else文。

分岐処理と順次処理の区別を常々意識

```
1  >>> a = int(input())
   10
2  >>> if a > 0:
   ...     judge = True
   ... else:
   ...     judge = False
   ...
3  >>> if judge:
   ...     print('It is a positive value.')
   ... else:
   ...     print('It is either 0 or a negative value.')
   ...     print('The program will exit.')
   ...
   It is a positive value.
4  >>>
```

else節の命令となっている例。

プログラムの終了を告げるためにprint関数を利用したつもりが表示されていない。

まとめ

- プログラムが**順次処理**される場合は書いた順番に命令が全て実行される。
- プログラムが**分岐処理**される場合は条件に応じて命令をスキップし、実行する命令を選択することができる。
- 分岐処理の**条件は値や式で指定**することができる。
- 分岐処理の具体的な構文には、**if文**、**if-else文**、**if-elif-else文**がある。
- if-else文やif-elif-else文においてelse節は省略可能できる。
- Pythonでは、これらの文のブロック（文の範囲）は**インデント**で示す。
- Pythonを含めて多くの言語では**代入**を = で示し、**比較**を == で示す。
- 条件判定において**偽(False)と判定される値以外は全て真(True)**となる。
- 比較演算は算術演算よりも処理の**優先順位が低い**。
- 比較演算では**左辺に変数、右辺にリテラル**を書く記法が一般的。
  ただし、逆に書いても動作する。

次回予告　**多重分岐**