2021年度 東京成徳大学 特別講座 資料
東京工科大学コンピュータサイエンス学部
福西広晃

特別講習の内容

第01回 Python基礎（電卓機能の作成）

- 画面出力、変数、選択（条件文）、反復（繰り返し文）

第02回 Python基礎（データの扱い方）

- リスト、辞書

第03回 Pythonによるテーブルデータの操作と統計計算

- Pandasライブラリ

第04回 Pythonによるデータ可視化

- matplotlibライブラリ

第05回 Pythonによる実践的なデータ分析

- 株価分析

なんのためにプログラミングを行うのか

業務（作業）の自動化　→　業務（作業）効率化

プログラミング言語Pythonについて

- 本講義では、プログラミング言語として Python を用いる

Pythonの特徴

- プログラミング言語としての仕様がわかりやすい（文法がシンプル）
- オープンソース（無料で利用できる）
- 豊富な標準ライブラリと外部のパッケージ（便利機能をもとアプリを想像してください）
- コンパイル不要（スクリプト言語）

    - コンパイルとは、プログラミング言語で書かれた文字列（ソースコード）を、コンピュータ上で実行可能な形式（オブジェクトコード）に変換すること
- 汎用性が高い（以下に紹介）

Pythonは何に使われているか

- Webアプリ開発
    - Instagram, YoutubeなどはPythonで書かれている
    - Django, Flaskなどのフレームワークを用いれば、効率よく開発できる(フレームワークとは、アプリケーションを開発する際によく使われる機能がまとまったソフトウェアのこと)
- 機械学習プログラミング(AI開発)
    - AI分野において、Pythonはデファクトスタンダードとなっている
    - Scilit-learn, TensorFlow, PyTorch等のライブラリを用いれば、予測モデルが生成できる
- データサイエンス
    - Pandas, Matplotlib, Seabornn等のライブラリを用いれば、統計解析やデータの可視化ができる
- Webスクレイピング（Webサイトからのデータ取得）
    - htmlの知識があれば、BeautifulSoup, Seleniumのライブラリを用いてWebサイトから効率的にデータを取得できる
- Excelの自動処理
- ゲーム開発
- 株やFXの自動売買
- ブロックチェーン開発（暗号化技術）

Pythonの人気ランキング

- 2020年度のプログラミング言語人気ランキング第2位
- 日経XTEC

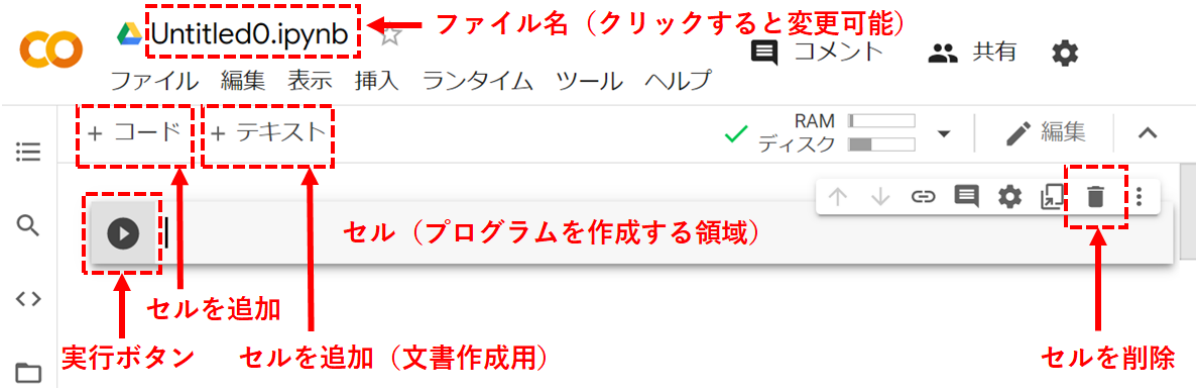
Pythonの開発環境

- オンライン(クラウドサービス)
  インターネットに繋がるブラウザがあればいつでもどこでも利用できる（スマートフォンでもOK）
    - **Google Colaboratory**：グーグル社が提供、対話型実行環境が利用できる、面倒な環境構築が不要で初学者の学習環境として最もメジャーな環境
    - AWS Cloud9：Amazon社が提供

- オフライン(ローカルPCで開発)
  ローカルPCにインストールして利用するため利用場所が制限される
    - Python(Pure Python): コマンドラインで実行
    - **Anaconda**: Anaconda社が提供、Jupyter NotebookやJupyter Labの対話型実行環境が利用できる、分析用途で広く利用されている環境、
    - Visual Studio Code: マイクロソフト社が提供（Webアプリの開発で広く利用されている環境）

Google Colaboratoryの使い方

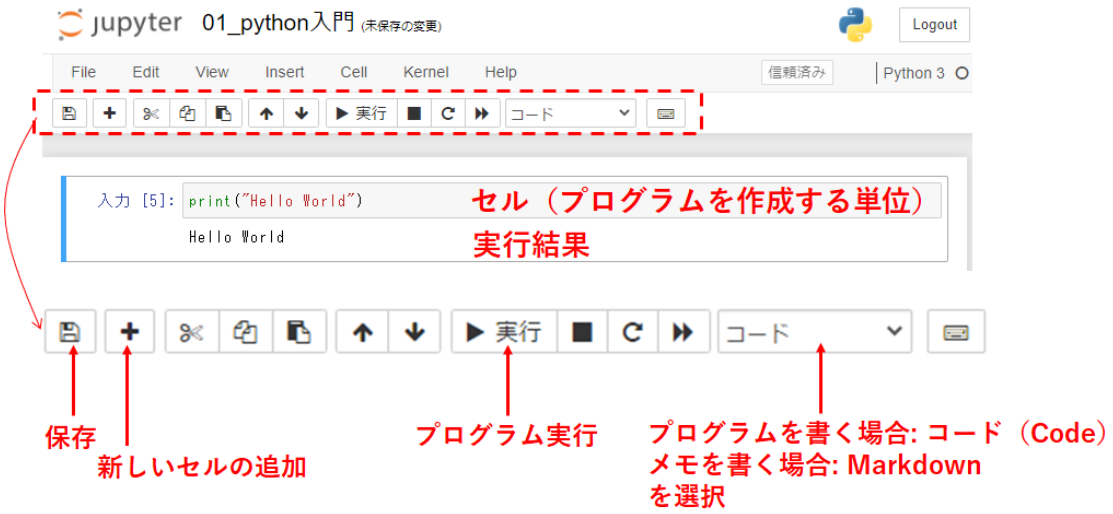
- Google ChromeやMicrosoft Edgeなどのブラウザを使ってプログラミングを行う
- Jupyter Notebook（読み方は「ジュパイター・ノートブック」または「ジュピター・ノートブック」）がベースとなっており、1セルごとにプログラムを実行・確認することができる（対話型実行環境とも呼ばれる）
- プログラムとその実行結果やその際のメモを簡単に作成でき、過去の作業内容の振り返りや、チームメンバーへ作業結果を共有する際に便利な他、授業や研修等の利用に適している
- 拡張機能（nbextensions）を導入すれば、利便性を向上できる



Jupyter Notebookの使い方[Anaconda用]

概要

- Jupyter Notebookの読み方は「ジュパイター・ノートブック」または「ジュピター・ノートブック」）
- Google ChromeやMicrosoft Edgeなどのブラウザを使ってプログラミングを行う
- 1セルごとにプログラムを実行・確認することができる（ 対話型実行環境 とも呼ばれる）
- プログラムとその実行結果やその際のメモを簡単に作成でき、過去の作業内容の振り返りや、チームメンバーへ作業結果を共有する際に便利な他、授業や研修等の利用に適している
- 拡張機能（nbextensions）を導入すれば、利便性を向上できる

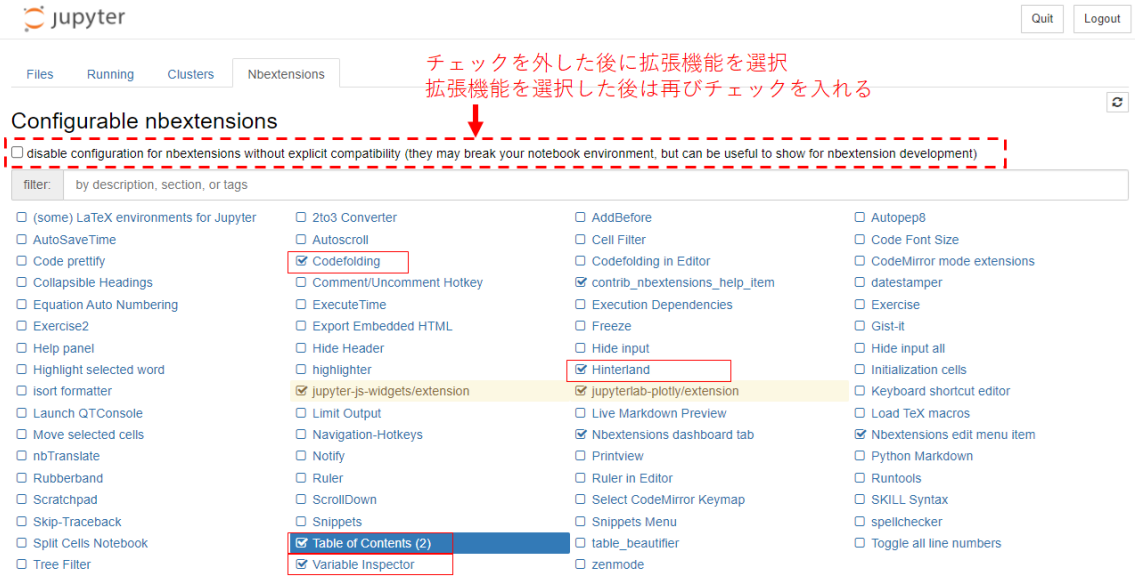


nbextensionsのインストール（Jupyter Notebookの拡張機能）

- Anaconda promptで以下を実行
  - `conda install -c conda-forge jupyter_contrib_nbextensions`
  - `jupyter nbextensions_configurator enable --user`
  - `conda install -c conda-forge/label/cf202003 jupyter_contrib_nbextensions`
  - 参考

拡張機能の追加

- Nbextensionsタブを開いて、「disable configuration・・・」のチェック外して必要な拡張機能にチェックを入れる
- 推奨の拡張機能
  - Table of Contents : 目次を追加する
  - Variable Inspector : 変数の一覧を表示する
  - Hinterland : 入力補完を使えるようにする
  - Codefolding : セル内のコードを折り畳めるようにする
- 拡張機能のチェックが終われば、再び「disable configuration・・・」のチェックを入れる



プログラムの実行結果の画面出力

print関数

- Pythonにおける画面への出力には `print関数` を使う
  `print(出力内容)`
- 文字列の出力："abc"（ダブルクォーテーション）または'abc'（シングルクォーテーション）で括る
- 数値の出力：ダブルクォーテーションやシングルクォーテーションで括る必要なし
- `print関数` で出力される文字列や数値の最後に改行（改行コード）が入る

In [ ]:
```
# 文字列を画面出力(カッコ内に画面出力した文字列)
print('Hello World') # 文字列はダブルクォーテーションで括る
```

上記の先頭の#はコメント（メモ書き）を表し、その行はコードとして認識されない

複数のprint関数

- 複数の値を出力する場合は，(カンマ)区切りで続けて記述できる
- print関数では最後に改行（改行コード）が入る

In [ ]:
```
print(10)
print(5.5)
print("ok")
print(10, "円")
print("時給",1000,"円")
```

最後に改行コードで入れない場合

- print(値, end="改行コードの代わりに用いる文字や記号")
- 複数のprint文が1行で表示される

In [ ]:
```
print(10, end=" ")
print(5.5, end=" ")
print("ok")
```

練習

(1) print関数を使って「I study Python.」と出力してみよう

In [ ]:
```

```

(2) print関数を2回使って「1111,9999」と出力してみよう。改行コードの代わりに , (カンマ)を付けることとする

In [ ]:
```

```

解答例

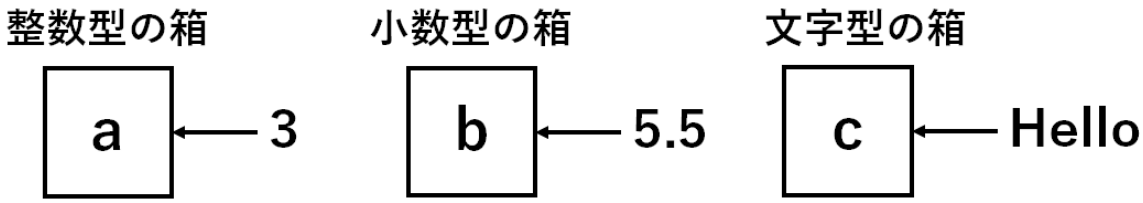
In [ ]:
```
#(1)
print("I study Python.")

#(2)
print(1111, end=',')
print(9999)
```

変数と型

変数とは

- 変数とは箱のように値が出し入れできるもの
- 変数には一度に **1つ** の値だけを扱える
- 変数に出し入れする値とは **数値** や **文字列** など
- 変数に値を入れることを **代入** という（特に、初めて値を入れる場合は **初期化** という）
- 変数の値や文字列を何度でも入れ替えることができる（入れ替えの際、古い値は消える）

- 値の性質によって **データ型** という分類があり、それらに対して行える計算機能はそれぞれ異なる
  - 特に、 **整数** 、 **浮動小数点数（小数）** 、 **文字** を扱う型は **基本データ型** と呼ばれる
  - 代入される値の種類によって変数には型が設定される
    - 整数: `int`型 (例: 3, 10, 5)
    - 浮動小数点数（小数）: `float`型 (例: 3.14)
    - 文字: `str`型 (例: ABC)
    - Pythonは、変数に対して型の明示は不要（他言語では型を宣言して明示することがある）。ただし、変数の型を意識する必要はある。例えば、「3」が整数か文字なのかで、四則演算できるかできないかに影響する。



- 変数は以下の例外を除いて自由に名前を付けることができる
  - 変数名の一文字目には数字は使えない
  - +, -, /, %, (,), # のような演算子や記号は使えない
  - Pythonで予め用意されている予約語(and, as, breakなど)や、組み込みオブジェクト(while、if、sumなど)は使えない
  - 変数名は単純なものよりも、変数がどのような意味を持つかを考えて付けるのが望ましい

In [ ]:
```
# 予約語
__import__('keyword').kwlist
```

In [ ]:
```
# 組み込みオブジェクト
dir(__builtins__)
```

In [ ]:
```
# aという変数(箱)に999を代入
a = 999
```

変数の代入（初期化）

In [ ]:
```
num1  = 123             # 整数（int型）
num2  = 123.456         # 小数（float型）
text  = "Hello World!"  # 文字列（str型）
judge = True            # 論理型（bool型）, bool値とは、TrueやFalseのような論理値のこと
```

変数の画面出力

- 変数にどのような値が代入されているかは、 `print関数` で出力できる

In [ ]:
```
print(num1)
print(num2)
print(text)
print(judge)

# コンマで区切って並べることもできる
print(num1, num2, text, judge)
```

変数の値を変更

In [ ]:
```
num1  = 987             # 整数型（int）
num2  = 567.23          # 浮動小数点型（float）
text  = "See you."      # 文字列型（str）
judge = False           #  論理型（bool）, TrueかFalseか
```

```
print("num1:", num1)
print("num2:", num2)
print("text:", text)
print("judge:", judge)
```

練習

(1) num3 という名前の変数に「50」の整数を代入した後、print関数で num3 を出力してみよう

In [ ]:

(2) text1 という名前の変数に「Good morning」の文字列を代入した後、print関数で text1 を出力してみよう

In [ ]:

解答例

In [ ]:

```
#(1)
num3 = 50
print(num3)

#(2)
text1 = "Good morning"
print(text1)
```

数値と演算

算術演算子

算術演算子	+	足し算
	-	引き算
	*	かける
	/	割る（小数）
	//	割る（整数）
	%	余り
	**	べき乗

四則演算

In [ ]:

```
# 足し算
print("足し算", 10 + 3)

# 引き算
print("引き算", 10 - 3)
```

```
# 掛け算
print("掛け算", 10 * 3)

# 割り算
print("割り算", 10 / 3)

# 割り算の商
print("割り算の商", 10 // 3)

# 割り算の余り
print("割り算の余り", 10 % 3)

# べき乗
print("3乗", 10 ** 3)
```

変数を使った四則演算

In [ ]:

```
num1 = 10
num2 = 3

# 足し算
print("足し算", num1 + num2)

# 引き算
print("引き算", num1 - num2)

# 掛け算
print("掛け算", num1 * num2)

# 割り算
print("割り算", num1 / num2)

# 割り算の商
print("割り算の商", num1 // num2)

# 割り算の余り
print("割り算の余り", num1 % num2)

# べき乗
print("3乗", num1 ** num2)
```

演算結果を新たな変数に代入して保持

In [ ]:

```
num1 = 10
num2 = 3

# 足し算
tasu = num1 + num2
print("足し算", tasu)

# 引き算
hiku = num1 - num2
print("引き算", hiku)

# 掛け算
kake = num1 * num2
print("掛け算", kake)

# 割り算
waru = num1 / num2
```

```
print("割り算", waru)

# 割り算の小
shou = num1 // num2
print("割り算の余り", shou)

# 割り算の余り
amari = num1 % num2
print("割り算の余り", amari)

# べき乗
beki = num1 ** num2
print("3乗", beki)
```

変数の値を１増やす

- プログラムの特有の記述方法

パターン1

In [ ]:
```
num1 = 10

num1 = num1 + 1
print(num1)
```

パターン2

In [ ]:
```
num1 = 10

num1 += 1
print(num1)
```

文字列の足し算

- 文字列同士を足し算すると結合される

In [ ]:
```
text1 = 'Hello'
text2 = 'World'
print(text1 + text2)
```

変数の型が異なる場合の演算

- 文字型と整数型の足し算はできない

In [ ]:
```
num11 = "3" # 文字
num12 =  5  # 整数

print(num11 + num12)
```

数値として計算する場合

- int関数 を用いて数字（文字）を数値に変換: int(数字)

In [ ]:
```
num11 = "3" # 文字型
num12 =  5  # 整数型
```

```
num11 = int(num11)
print(num11 + num12)
```

文字として計算する場合

- str関数 を用いて数値を数字（文字）に変換: str(数値)

In [ ]:
```
num11 = "3" # 文字型
num12 =  5  # 整数型

num12 = str(num12)
print(num11 + num12)
```

練習

(1) 変数num3に13、変数num4に2を代入した後、 2 つの変数の割り算の余りを計算し、出力してみよう。
この問題はプログラミングにおいて重要です。2で割った余りが0かそれ以外で、奇数か偶数かを判定することに用いられます。

In [ ]:
```

```

(2) 底辺が13、高さが2の三角形の面積の値をsankakuという変数に代入した後、出力してみよう。

In [ ]:
```

```

(3) 半径13の球の体積をkyuuという変数に代入した後、出力してみよう。球の公式は円周率は3.14とする。

In [ ]:
```

```

解答例

In [ ]:
```
#(1)
num3 = 13
num4 = 2
print(num3%num4)

#(2)
sankaku = 13*2/2 # 三角形の面積
print(sankaku)

#(3)
kyuu = 4*13*13*3.14/3  # 球の体積
print(kyuu)
```

比較演算子

- 等しい(==)、〇〇以上(>=)、〇〇未満(<)などの左辺と右辺の比較に使う演算記号を、 比較演算子 と言う
- 出力は True （真）または False （偽）

- 左辺の変数に右辺の値を代入する場合は = であったが、左辺と右辺の値を比較する場合は == を使うことに注意

比較演算子	<	小さい
	>	大きい
	<=	以上
	>=	以下
	==	等しい
	!=	等しくない

In [ ]:
```
num1 = 10

# 等しい
print("判定：", num1 == 10)

# 等しくない
print("判定：", num1 != 10)

# 大きい
print("判定：", num1 > 10)

# 小さい
print("判定：", num1 < 10)

# 以上
print("判定：", num1 >= 10)

# 以下
print("判定：", num1 <= 10)
```

練習

- 半径5cmの円の面積をenとう変数に代入した後、100cm^{2}を超えるかを判定し、TrueまたはFalseを出力してみよう。円周率は3.14とする。

In [ ]:
```

```

解答例

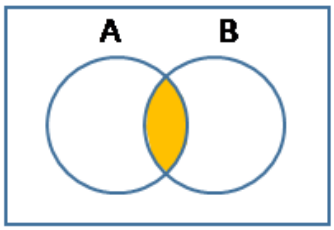
In [ ]:
```
en = 5*5*3.14 # 円の面積
print("判定:", en > 100)
```

論理演算子

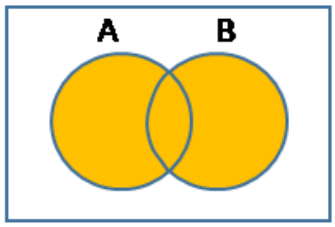
- 2つ以上の比較条件を組み合わせを判定する場合に使う
- 出力は `True` （真）または `False` （偽）

論理演算子	and	両者を満たす
	or	どちらか片方を満たす
	not	満たさない

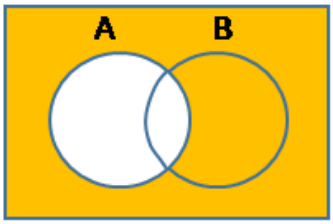
A	B	A or B	A and B	not A	not B
True	True	True	True	False	False
True	False	True	False	False	True
False	True	True	False	True	False
False	False	False	False	True	True



A and B
A かつ **B**
論理積



A or B
A または **B**
論理和



not A
A ではない
否定

2つの比較条件の判定

In [ ]:
```
num1 = 10
num2 = 20

# 条件1と条件2の両方を満たすか
print("判定: ", num1 >= 10 and num2 <= 10)

# 条件1と条件2のどちらかを満たすか
print("判定: ", num1 >= 10 or num2 <= 10)

# 条件を満たさないか
print("判定: ", not num1 >= 10)

# 条件を満たさないか
print("判定: ", not num2 <= 10)
```

3つの比較条件の判定

In [ ]:
```
num1 = 10
num2 = 20
num3 = 30

# 条件1と条件2と条件3を同時に満たすか
print("判定: ", num1 >= 10 and num2 <= 10 and num3 == 10)

# 条件1と条件2と条件3のいずれかを満たすか
print("判定: ", num1 >= 10 or num2 <= 10 or num3 == 10)
```

練習

縦3.5cm、横4.3cmの四角形の面積をareaという変数に代入し、15<area<16であるかを判定せよ

In [ ]:

```
```

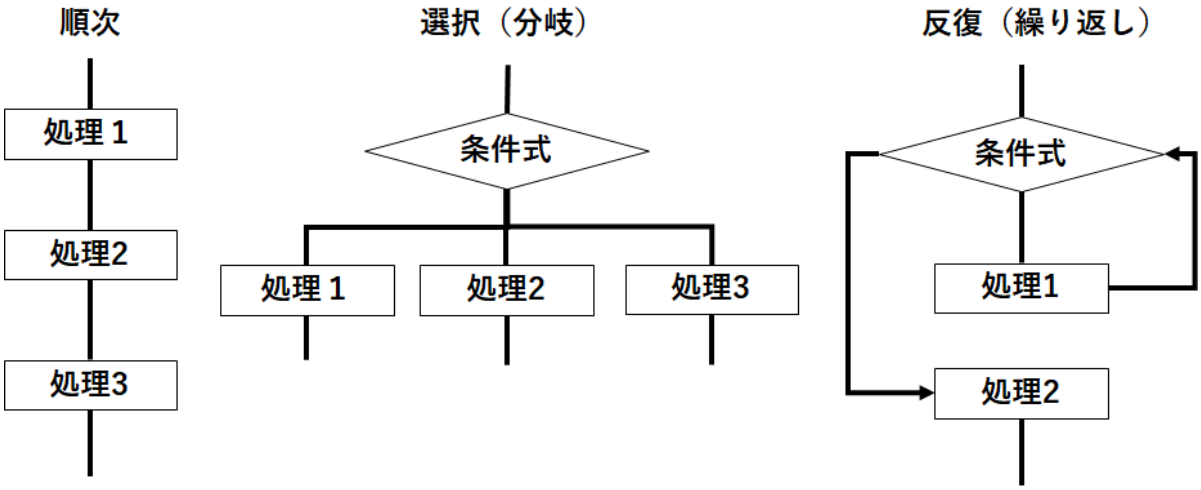
解答例

In [ ]:

```
area = 3.5 * 4.3

print(area > 15 and area< 16)

#または以下の書き方でもOK
print(15 < area < 16)
```

制御構文

- プログラムに高度な機能（電卓以上）を持たせるためには制御構文が必要
- 制御構文とは、プログラムの実行順序を制御するためにもの
- プログラムは、 順次 ， 選択（分岐） ， 反復（繰り返し、ループ）の3つ制御構文の組み合わせである
- この3つによって、あらゆる実行順序が記述できることが証明されている



選択（分岐）

if文

- 選択（分岐）には `if文` を用いる
- if文を用いることで、「もし〇〇だったら、処理A。そうでなく△△だったら、処理B。」のように、条件によって処理を変更できる

- **1つの条件を判定する場合**

```
if 条件式1:
  条件式1がTrue(真)ならば、ここに記述された処理を行う
```

条件式の後に ： (コロン)が付く（**付け忘れしやすいので注意**）

- **複数の条件を判定する場合**

```
if 条件式1:
  条件式1がTrue(真)ならば、ここに記述された処理を行う
elif 条件式2:
  条件式1がFalse(偽)で、条件式2がTrue(真)ならば、ここに記述された処理を行う
elif 条件式3:
  条件式1がFalse(偽)で、 条件式2がFalse(偽)で、条件式3がTrue(真)ならば、ここに記述された処理を行う
else:
  全ての条件式がFalse(偽)ならば、ここに記述された処理を行う
```

ifの条件式が満たされていなければ、elifの条件式が上から順番に判定される
これらの条件式がすべて満たされていなければ、else内の処理が実行される

[注意事項]

- if文の処理を記述する際は 空白（半角スペース）を入れて字下げ しなければならない（「 インデントを揃える 」という）。通常、半角スペース4つ
- 毎回半角スペースのキーを4回打つのは面倒なので、 Tab キーを押せば自動的に調整してくれる

変数numが100未満なら「AAA」と出力するプログラム

In [ ]:
```
num = 50
if num < 100:
    print("AAA")
```

変数numが100未満なら「AAA」、それ以外は「BBB」と出力するプログラム

In [ ]:
```
num = 101
if num < 100:
    print("AAA")
else:
    print("BBB")
```

変数numが100未満なら「AAA」、100以上200未満なら「BBB」、それ以外は「CCC」と出力するプログラム

In [ ]:
```
num = 150
if num < 100:
    print("AAA")
elif num < 200:
    print("BBB")
else:
    print("CCC")
```

変数numが100未満なら「AAA」、100以上200未満なら「BBB」、200以上300未満なら「CCC」、それ以外は「DDD」と出力するプログラム

In [ ]:
```
num = 150
if num < 100:
    print("AAA")
elif num < 200:
    print("BBB")
```

```
elif num < 300:
    print("CCC")
else:
    print("DDD")
```

numが1の場合はnumを1増やし、numが2の場合はnumを2増やし、それ以外の場合はnumを3増やすプログラム

In [ ]:
```
num = 3

if num == 1:
    num = num + 1
elif num == 2:
    num = num + 2
else:
    num = num + 3

print("出力: ", num)
```

if文では、変数に代入する値を様々に変更したいため、 外部入力 という方法で値を試すことができる

```
input()
```

外部入力では数値を入力したくても文字扱いになるため、プログラム内で数値で扱いたい場合は整数型や小数型に変換する必要がある

- 数字（文字）から整数値に変換する場合は `int関数` を用いる: `int(文字)`
- 数字（文字）から小数値に変換する場合は `float関数` を用いる: `float(文字)`

In [ ]:
```
num = input("入力:  ")
num = int(num) # 整数値に変換して置き換え

# if文
if num == 1:
    num = num + 1
elif num == 2:
    num = num + 2
else:
    num = num + 3

print("出力: ",num)
```

チケットの価格は以下のように設定されている

- 18歳未満の男は1000円
- 18歳未満の女は800円
- それ以外は2000円

年齢と性別によって価格を表示するプログラム

In [ ]:
```
nenrei  =  input("年齢:  ")
seibetu =  input("性別:  ")

nenrei  =  int(nenrei) # 整数値に変換して置き換え

if nenrei < 18 and seibetu == "男":
```

```
    price = 1000
elif nenrei < 18 and seibetu == "女":
    price = 800
else:
    price = 2000

print("価格: ",price)
```

練習

numという変数に代入された数値によって以下の表示になるプログラムを作成してみよう

- 2の倍数だが、3の倍数でないの場合は、「2の倍数」と表示
- 3の倍数だが、2の倍数でないの場合は、「3の倍数」と表示
- 3の倍数、かつ、2の倍数であるの場合は、「2の倍数かつ3の倍数」と表示
- それ以外は、「2の倍数でも3の倍数でもない」と表示

ヒント：Xの倍数の判定する場合、数値をXで割って余りが0ならXの倍数となる

In [ ]:

```
```

解答例1

In [ ]:

```
num = input("数値:  ")
num = int(num)

if num % 2 == 0 and num % 3 != 0:
    print("2の倍数")
elif num % 2 != 0 and num % 3 == 0:
    print("3の倍数")
elif num % 2 == 0 and num % 3 == 0:
    print("2の倍数かつ3の倍数")
else:
    print("2の倍数でも3の倍数でもない")
```

解答例2

In [ ]:

```
num = input("数値:  ")
num = int(num)

# 最初のif文で2の倍数かつ3の倍数を判定させると、
# elifでは2の倍数か3の倍数かだけを判定すればよい
if(num % 2 == 0 and num % 3 == 0):
    print("2の倍数かつ3の倍数")
elif(num % 2 == 0):
    print("2の倍数")
elif(num % 3 == 0):
    print("3の倍数")
else:
    print("2の倍数でも3の倍数でもない")
```

反復（繰り返し、ループ）

- 反復（繰り返し、ループ）には `for文` や `while文` を用いる

- for文 は、指定した回数を繰り返す（繰り返し範囲を指定するためには、リストやrangeをin演算子とともに用いる）
- while文 は、ある条件を満たすまで繰り返す
- 繰り返しの途中で処理を中断したい場合には、 break文 を使う
- 繰り返しの途中で特定の処理だスキップしたい場合には、 continue文 を使う

今回は、特に重要なfor文を扱います。while文、break文、continue文は付録にします

for文

順次で記述した場合（for文を使わない）

0から9までの整数をカンマ区切りで出力する例

In [ ]:
```
print(0, end=',') # 末尾は改行ではなくコンマに指定
print(1, end=',')
print(2, end=',')
print(3, end=',')
print(4, end=',')
print(5, end=',')
print(6, end=',')
print(7, end=',')
print(8, end=',')
print(9, end=',')
```

0から1000まで出力したいとなった場合は、効率が悪すぎる

練習

「programming language」の文字列をカンマ区切りで5回出力してみよう。

In [ ]:
```

```

解答例

In [ ]:
```
print("programming language", end=',') # 末尾は改行ではなくコンマに指定
print("programming language", end=',')
print("programming language", end=',')
print("programming language", end=',')
print("programming language", end=',')
```

for文を用いた反復（繰り返し、ループ）の構文

繰り返し範囲を指定するためには、 range関数 や リスト のオブジェクト（データの集合）をin演算子とともに用いる。
データを1個ずつ変数に代入して反復処理を行う

```
for 変数 in オブジェクト:
    実行する処理
```

[注意事項]

- if文の処理を記述する際は 空白（半角スペース）を入れて字下げ しなければならない（「 インデントを揃える 」という）。通常、半角スペース4つ
- 毎回半角スペースのキーを4回打つのは面倒なので、 Tab キーを押せば自動的に調整してくれる

リストについては、次回の学習予定のためrange関数のみを扱う

range関数 の使い方

```
range(n)
```

- 0 から n-1 までの n個 の整数を順番に並べたデータ集合を生成
- 例えば、range(3)の場合、0,1,2のデータの集合が生成される
- for文では0,1,2を順番に変数に代入して処理を繰り返す
- つまり、3回処理を反復することができる

```
range(開始番号，終了番号)
```

- 開始番号から開始して、 終了番号-1まで反復処理を実施

```
range(開始番号，終了番号，ステップ数)
```

- 開始番号からして、 終了番号-1まで、ステップ数の間隔で反復処理を実施

0から9までの整数をカンマ区切りで出力するプログラム

In [ ]:
```
for i in range(10):
    print(i, end=',')
```

「programming languate」をカンマ区切りで10回出力するプログラム

In [ ]:
```
for i in range(10):
    print("programming language", end=',')
```

5から9までの整数をカンマ区切りで出力する例

In [ ]:
```
for i in range(5, 10):
    print(i, end=',')
```

0から9までの偶数をカンマ区切りで出力する例

In [ ]:
```
for i in range(0, 10, 2):
    print(i, end=',')
```

練習

(1) for文を使って0から199までの整数をカンマ区切りで出力してみよう

In [ ]:
```

```

(2) for文を使って9999の整数を17回カンマ区切りで出力してみよう

In [ ]:

(3) for文を使って342から365までの整数をカンマ区切りで出力してみよう

In [ ]:

(4) for文を使って3から30までの3の倍数をカンマ区切りで出力してみよう

In [ ]:

解答例

In [ ]:
```
#(1)
for i in range(200):
    print(i, end=',')
```

In [ ]:
```
#(2)
for i in range(17):
    print(9999, end=',')
```

In [ ]:
```
#(3)
for i in range(342, 366):
    print(i, end=',')
```

In [ ]:
```
#(4)
for i in range(3, 31, 3):
    print(i, end=',')
```

for文を使った合計値を計算

1から100までの整数を足し合わせたときの合計値

In [ ]:
```
sum = 0 # 合計を保持する変数で、初期値を0としておく

for i in range(1, 101): # 開始は1で終了は101-1=100
    sum = sum + i       # sumにiを加算
    print(sum, end=',') # sumの反復過程を出力

print()                 # 改行のみ
print("合計: ", sum)    # 合計出力
```

1から100までの偶数の合計を計算

In [ ]:
```
sum = 0 # 合計を保持する変数で、初期値を0としておく

for i in range(2, 101, 2): # 開始は2で終了は101-1=100で2コ飛ばし
    sum = sum + i          # sumにiを加算
    print(sum, end=',')    # sumの反復過程を出力

print()                    # 改行のみ
print("合計: ", sum)       # 合計出力
```

練習

1から100までの奇数の合計を計算してみよう

In [ ]:
```
```

解答例

In [ ]:
```
sum = 0 # 合計を保存する変数で、初期値を0としておく
for i in range(1, 101, 2): # 開始は1で終了は101-1=100で2コ飛ばし
    sum = sum + i          # sumにiを加算
    print(sum, end=',')    # sumの反復過程を出力

print()               # 改行のみ
print("合計: ", sum)  # 合計出力
```

for文とif文の組み合わせ

8月に31日間アルバイトをしたとする。1日目から15日目までは日給は3000円、16日目から31日目までは3500円であった場合、8月の合計収入を計算してみよう。for文でカレンダー上の1日から31日まで反復とする。

In [ ]:
```
sum = 0 # 合計を保存する変数で、初期値を0としておく

for i in range(1, 32):      # 開始は1で終了は32-1=31
    if(i <= 15):
        sum = sum + 3000    # sumに3000を加算  sum += 3000でもOK
    else:
        sum = sum + 3500    # sumに3500を加算  sum += 3500でもOK

    print(sum, end=",")

print()
print("合計: ", sum, "円")  # 合計出力
```

練習

8月に31日間アルバイトをしたとする。偶数の日の日給は4100円、奇数の日の日給は3800円であった場合、8月の合計収入を計算してみよう。for文でカレンダー上の1日から31日まで反復とする。偶数と奇数の判定は、2で割った余りが0か0以外かで判定できる。

In [ ]:
```
```

解答例

In [ ]:
```
sum = 0                    # 合計を保存する変数で、初期値を0としておく
for i in range(1, 32):     # カレンダーの1日から31日まで繰り返し
    if(i % 2 == 0):        # 偶数の日の判定
        sum += 4100        # sumに4100円加算
    else:                  # 奇数の日
        sum += 3800        # sumに3800円加算
print("合計収入:", sum,"円")  # 合計出力
```

以降、付録

---

while文

- ある条件を満たしている間繰り返しの処理をするためには、while文を用いる
  `while 条件式:`
  　実行する処理
  条件式がTrue(真)の間は繰り返す

[注意事項]

- if文の処理を記述する際は 空白（半角スペース）を入れて字下げ しなければならない（「 インデントを揃える 」という）。通常、半角スペース4つ
- 毎回半角スペースのキーを4回打つのは面倒なので、 Tab キーを押せば自動的に調整してくれる

0から1を足すことを繰り返して100以上になれば終了するプログラム

In [ ]:
```
sum = 0                 # 合計を保持する変数で、初期値を0としておく
while sum < 100:
    sum = sum + 1
    print(sum, end=',') # sumを表示

print()
print("合計: ", sum)
```

`sum < 100` の条件式は、sumが100になるまでTrue(真)となる
`sum = sum + 1` がなければ、sumが0のまま更新されずに条件式が常にTrueになる。つまり、繰り返しが終わることがない無限ループと呼ばれる状況に陥るので注意
`sum = sum + 1` は、 `sum += 1` と書くことも多い

練習

0から13を足すことを繰り返して、1000以上になったら終了させてみよう

In [ ]:
```

```

解答例

In [ ]:
```
sum = 0             # 合計を保持する変数で、初期値を0としておく
while sum < 1000: # sumが1000未満の間は繰り返し
    sum += 13
    print(sum, end=',') # sumを表示

print()
print("合計: ", sum)
```

break文

- 繰り返しの途中で処理を中断したい場合には、 break文 を使う
- break文 が実行された時点でその繰り返し文から抜け出る

- if文 とともに用いることが多い

0から1を足すことを繰り返して20以上になったら終了させる。このとき、7の倍数が初めて現れたら繰り返しを中断するプログラムは以下のようになる

In [ ]:
```
sum = 0 # 合計を保持する変数で、初期値を0としておく
while sum < 20: # sumが20以下の間は繰り返し
    sum += 1
    print(sum, end=',') # sumを表示

    if sum % 7 == 0: # 7の倍数の判定（7で割った余りが0）
        break        # 7の倍数の場合は、繰り返しを中断

print()
print("初めて現れる7の倍数: ", sum)
```

練習

8月に日給3万円のアルバイトをした場合、合計収入が20万円以上になるのは何日目かを出力してみよう。ここではfor文を使うものとする。

In [ ]:
```

```

解答例

In [ ]:
```
sum  = 0 # 合計を保持する変数で、初期値を0としておく

for i in range(1, 32): # カレンダーの1日から31日まで繰り返し
    sum  += 3 # 3万円ずつ加算
    print(sum, end=',') # sumを表示

    if sum >= 20: # 20万円以上になったら繰り返しを中断
        break

print()
print(i, "日目")
```

continue文

- 繰り返しの途中で特定の処理だスキップしたい場合には、 continue文 を使う
- continue文 が実行された時点で、それ以降の処理は実行されずに、次の繰り返しに移る
- if文 とともに用いることが多い

0から1を足すことを繰り返して結果を出力するものとする。ただし、3の倍数が現れたら出力をスキップするプログラムは以下のようになる

In [ ]:
```
sum = 0          # 合計を保持する変数で、初期値を0としておく
while sum <= 20:
    sum += 1

    if sum % 3 == 0: # 3の倍数を判定（3で割った余りが0）
        continue     # 3の倍数の場合は、以降のprint文をスキップする

    print(sum, end=',') # sumを表示
```

練習

8月の7の倍数の日に日給5000円のアルバイトを行ったとする。8月の合計収入を出力してみよう。ここではfor文を使うものとする。

In [ ]:

```
```

解答例

In [ ]:

```
sum = 0 # 合計を保持する変数で、初期値を0としておく

for i in range(1, 32): # カレンダーの1日から31日まで繰り返し
    if i % 7 != 0: # 7の倍数でない場合はスキップ
        continue

    sum += 5000 # 7の倍数の日は5000円を加算
    print(sum, end = ",") # sumを表示

print()
print("合計収入:", sum, "円")
```

文字列の切り出し

- 変数名[開始位置:終了位置]
  【注意】 実際に切り出される文字列は 開始位置 から 終了位置－1 まで

文字列	S	e	e		y	o	u	.
インデックス(文字位置) 前方から	0	1	2	3	4	5	6	7
インデックス(文字位置) 後方から	0	-7	-6	-5	-4	-3	-2	-1

In [ ]:

```
text  = "See you."     # 文字列型 (str)

# Seeのみを切り出し
print("前方から:", text[0:3])
print("後方から:", text[0:-5])

# youのみ切り出し
print("前方から:", text[4:7])
print("後方から:", text[-4:-1])

# See以降を最後まで切り出し (終了位置を指定しない)
print("前方から:", text[4:])
print("後方から:", text[-4:])
```