各種条件分岐処理およびループ処理

名古屋大学 情報基盤センター
情報基盤ネットワーク研究部門
基盤ネットワーク研究グループ
嶋田 創

条件分岐とは

- 変数の値を見てある処理を実行したりしなかったりするプログラムを制御する構文
- 代表的な物
  - if文: 「もし〇〇だったら」処理を実行
  - ループ処理: 「〇〇の条件」を満たしている間は処理を繰り返す
    - for文: 与えられた値の数だけ処理を繰り返す
- 条件の設定は命題論理(ブール論理)が重要になるので、苦手な人は要復習
  - とりあえず、論理積、論理和、論理否定は重要

if文

- もし「条件が成立する(真 = True)」ならば、ある処理を実行
  - 「条件が成立しない」は「偽 = False」とも呼ばれる
  - このあたりは命題論理と同じ(ブール論理)
- 書式(条件文の後にコロンを忘れずに!)

  if *条件文*:

  　　条件が成立したなら実行する処理(if文内)

  　　…

  if文に関係なく実行する処理(if文外)
- VS Codeではタブキーと Shift+タブキーでインデントの増減ができるので
  - if文を書いた直後は自動的にインデントが追加されてif文内を書ける状態になる
  - if文内を書き終わったらShift+タブキーでif文外のインデントに戻す

条件文の書き方(1/2)

- 基本的な条件文
  - 「a == b」: aとbが等しい(a equal to b)
    - 「a = b」と書くと「bをaに代入する」となってしまう点に注意
  - 「a != b」: aとbが等しくない(a not equal to b)
  - 「a > b」: aがbより大きい(a greater than b)
  - 「a >= b」: aがb以上である(a greater equal b)
  - 「a < b」: aがbより小さい(a less than b)
  - 「a <= b」 : aがb以下である(a less equal b)
- 英語表記は日本語よりも間違えにくいので覚えておくと良い
  - プログラミング言語によっては、英語表記の略を使うものもある
    - 例: eq(EQual), ne(Not Equal), gt(Greater Than), ge(Greater Equal)
- 条件文の演算子の優先順位は論理演算子の次に低い
  - 「a == b + 1」とすると、「a」と「b + 1」が比較される

条件文の書き方(2/2)

- 複数の条件を組み合わせる場合には論理演算子を利用
- 論理演算子
  - 論理積(and): 2つ以上の条件が同時に成立する
    - 例:「(a > b) and (c < d)」: aがbより大きく、なおかつ、cがdより小さい
  - 論理和(or): 2つ以上の条件のいずれかが成立する
    - 例:「(a > b) or (c < d) or (e == f)」: aがbより大きい、もしくは、cがdより小さい、もしくは、eとfが等しい
  - 否定(not): これを接頭辞とした条件は値が否定される(反転する)
    - 例:「not (a > b)」: aがbより大きくない
- 数値をそのまま条件文の値にできる
  - 「0」: 条件は偽である
  - 「0以外」: 条件は真である
- 文字列などは「空」であれば偽、それ以外は真となる

else節

- if文に「条件が不成立(偽)」の時のみ実行する処理を追加
- 書式

  if *条件文*:

  　　条件が成立したなら実行する処理(if節)

  　　…

  else:

  　　条件が不成立なら実行する処理(else節)

  　　…

  if文に関係なく実行する処理(if文外)
- 条件が偽の時のみ実行したい場合は?
  - 条件文の前に論理否定(not)を追加
  - if節に「pass(何もしないことを示す予約語)」だけを書いてelseを利用

elif節

- 条件文を指定できるelse節(else + if = elif)
- 書式

if *条件文A* :

　　条件Aが成立したなら実行する処理

　　…

elif *条件文B* :

　　条件Bが成立なら実行する処理

　　…

else:

　　条件A,条件Bともに不成立なら実行する処理

　　…

if文に関係なく実行する処理(if文外)

while文

- 条件が成立するならば、ある処理を繰り返し実行する
  - 処理が終了したら、whileの所に戻って再び条件を評価(ループ処理)
  - 条件文を間違えたり、条件文に使われている変数が(ミスで)更新されないと延々と処理を繰り返すことに(無限ループ)
    - 「Ctrl + C」でプログラムを止めましょう(どの行で止まった教えてくれます)
    - わざと無限ループにする場合は条件文に「1」とか「True」とかを書く
- 書式

while *条件文* :

    条件が成立したなら実行する処理

    …

    (条件に関する変数を更新する処理)

while文に関係なく実行する処理

ループを途中で抜けたい時(1/2)

- break
  - そのwhile文などを打ち切って抜け出す
  - 使用例: while中に別の条件を満たしたら打ち切り

while *条件文1* :
    条件が成立したなら実行する処理1
    if(*条件文2*):
        break
    条件が成立したなら実行する処理2
    (条件に関する変数を更新する処理)
while文に関係なく実行する処理

ループを途中で抜けたい時(2/2)

- continue
  - そのwhile文などの1回のループを抜け出す
  - continueの後の処理をせずに、while文などの条件評価の所に戻る
- exit()
  - プログラムを終了する
  - 厳密にはexit関数であって予約語ではない
    - 括弧の中にはOSに伝達する終了コードを入れるが、なければ空白でOK

いずれも使用例はbreakの説明に同じ

for文

- 変数の集合であるリスト(次回内容)などから1つずつ変数を取り出しながらループ
  - 取り出した変数をループ内で処理してもいいし、処理しなくてもいい
  - (広く使われているC言語ベースのfor文とは異なる点に注意!)
- 書式

```
variable_list = [a, b, c, ...]
for one_variable in variable_list :
    for文内で実行する処理
    sum = sum + one_variable * one_variable
    ...
for文に関係なく実行する処理
```

range関数を使ったfor文

- 「指定された数だけの数列のリスト」を作るrange関数をリストの生成に使うと、for文は使いやすくなる
  - 使用例1: 指定した回数のループ処理を行う
  - 使用例2: 指定した数列を作って、for文内部で個々の値を利用
- 書式(10回のループ処理)

for *one_variable* in range(10) :

    for文内で実行する処理

    ...

for文に関係なく実行する処理

range関数

- range(*回数*)
  - 0から「*回数* - 1」までの整数を生成したリストを返す
- range(*開始値*, *終了値*)
  - *開始値*から「*終了値* - 1」までの整数を生成したリストを返す
- range(*開始値*, *終了値*, *ステップ*)
  - *開始値*から「*終了値* - 1」まで*ステップ*おきに整数を生成
- 参考:
  - 回数や終了値に-1が入るのは他の言語のfor文の一般的な利用に合わせるため
  - コンピュータの世界では0から数字を数える風習が強い
    - 2進数n桁で表せる正の整数は0から「2のn乗 - 1」に関連

他の言語のfor文

- for文で、「初期値、条件文、1ループごとの値の更新処理」を実行させる言語の方が実は多い
  - 例: C言語のfor文

    ```
    for(i = 0; i < max_value; i++){
        for文で実行する処理
    }
    ```

  - 動作の違いに戸惑わないように注意
- シェルスクリプトのfor文はPythonと同じ
  - というか、Python側がこちらを参考にした?
  - 多くの言語では、この仕様のfor文はforeach文と名付けられていることが多い

while文/for文に対するelse節

- while文/for文に対するelse節は、「while文/for文が終了した時に実行する処理」となる
  - breakでwhile文/for文を抜けた時には実行されない
  - while文/for文内の処理が1回も実行されなくてもelse節は実行される
- 書式

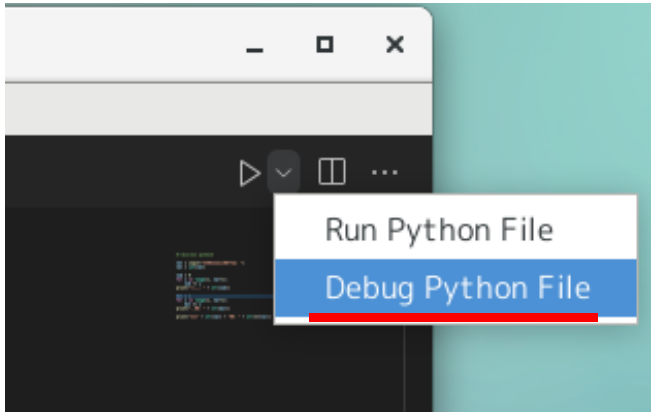
```
while 条件文:
    条件が成立したなら実行する処理
    …
    (条件に関する変数を更新する処理)
else:
    while文が終了したら実行する処理
while文に関係なく実行する処理
```

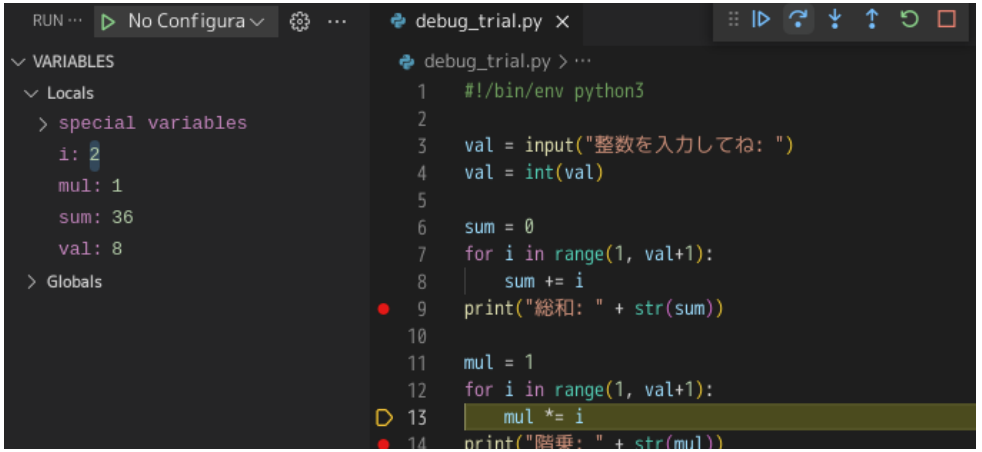
条件分岐やループのTips

- ループ中に「変数に対して加算した結果を同じ変数に格納」する場合は+=という演算子が使える
  - 例: i = i + 1　→　i += 1
  - ○○代入演算子などと呼ばれる(-=, *=, /=, %=, など)
- 大きなバグを起こさないように、if文を使ってassertionをかけることはよくやります
  - 例: 変数が想定していた範囲から外れていたら「変数がこんな値になった。おかしい。」と止まる
    - プログラムの設計上そんな値にならないはずでも、バグを起こしそうな込み入った所にしかけておく
  - 例: リストの要素数が想定していた範囲から外れていたら…(同上)
  - Pythonプログラムは値やリストがおかしくてもそれなりに動いたりするので、assertionで潜在的な問題をつぶすことは大事

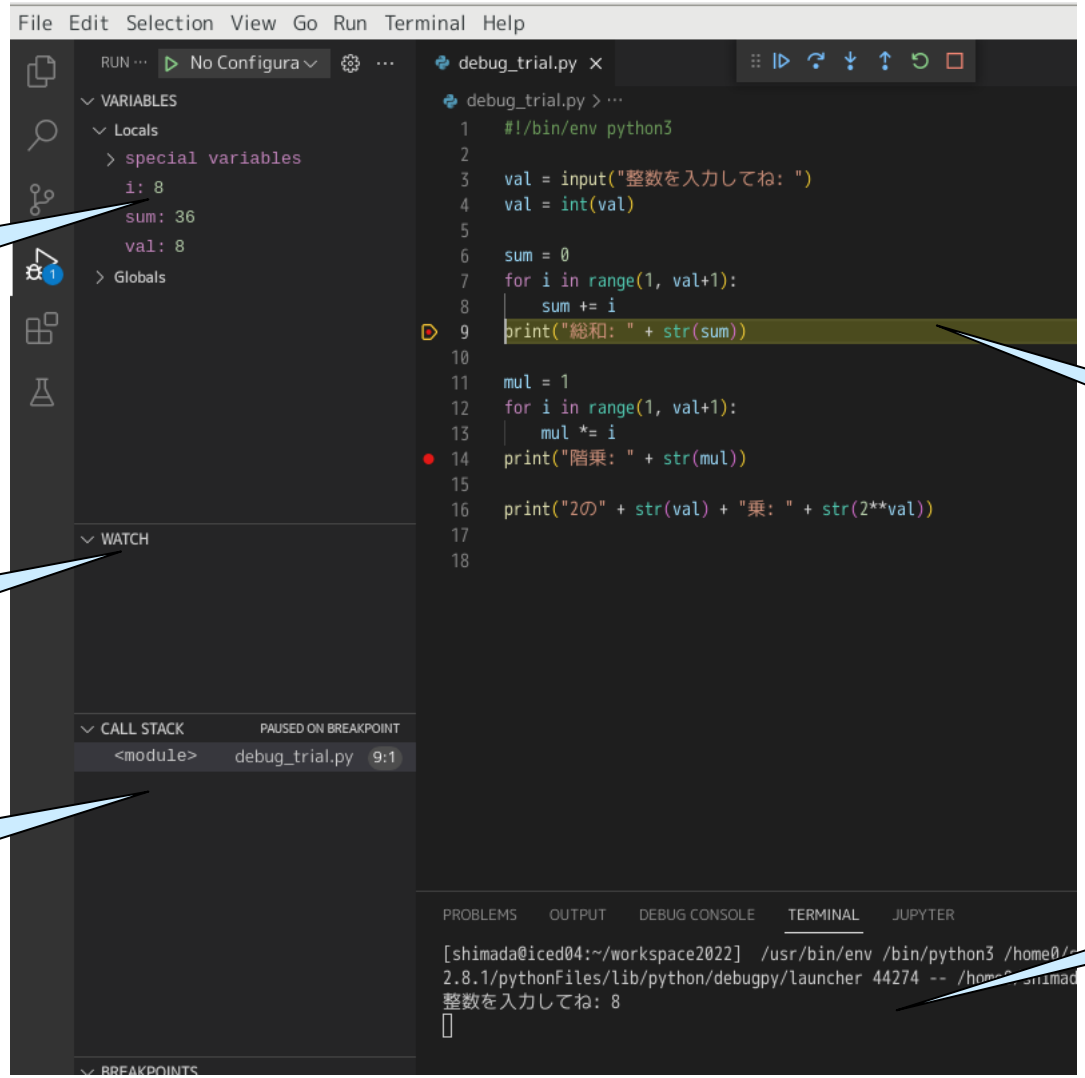
VS codeでのデバッグの初歩(1/3)

- 実行アイコンから、「Debug Python File」を選択することでデバッグ機能の下で実行できる
- デバッグ機能
  - 実行中の変数の値を見れる
  - 1行ずつ実行して出力や変数を見れる(ステップ実行)
  - ブレークポイントを設定して、そこまで実行した時点で止めて出力や変数を見れる
    - 設定したブレークポイントにさらに止める条件を追加することも





VS codeでのデバッグの初歩(2/3)



デバッガが勝手に表示した変数

自分で設定した監視する変数

関数呼び出しの状態(後で)

現在実行している行

シェル上の標準入出力

VS codeでのデバッグの初歩(3/3)

- ステップ実行は上のステップ実行アイコンをクリック
- ブレークポイント設定/解除は行の左をクリック
- 一旦停止から再開すると、次のブレークポイントまで実行
- (関数の出入り: 特に指示しない限り、関数呼び出しをしている行は、呼び出しの結果のみを受け取って次の行に行く)

この講義は初歩的な話なので、ブレークポイント設定と(勝手に表示してくれる)変数表示で特に困らないと思う

- もっと高度なデバッグ機能に興味がある人は自分で調べること