共通テスト用プログラム表記

2025年からの共通テストで使われる文法

変数と値

変数名は、英字で始まる英数字とアンダースコア「_」の並びで表す。

特に指示がなければ、

・小文字で始まる変数は通常の変数を表す。
例）kosu
・アンダースコア「_」を文字途中で入れることも可能。

例）kosu_gokei

・大文字で始まる変数は配列を表す。
例）Tokuten

・全て大文字の変数は実行中に変化しない値（定数）を表す。
例）TOKUTEN

数値・文字列の表し方

- 数値は特に断らない限り、10進数で表す。
  例）100
  　　99.99
- 文字の並びを「”」と「”」でくくって表す。
  例）”Hello world”
  　　“こんにちは　世界”
- 文字列は + で連結できる
  例）”あれは” + “りんごだ”　⇨　　あれはりんごだ

出力（表示文）

- 数字や文字列を表示する場合は、表示関数を使う
  カンマで連結することも可能。

例）表示する(“こんにちは”)
　　実行結果：こんにちは

例）kosu = 3
　　表示する(kosu, “個見つかった”)
　　実行結果：3個見つかった

代入文

kosu に 3 を代入する場合

kosu = 3

Tokuten配列の添字 4 に100を代入する場合

Tokuten[4] = 100

Tokuten配列の１番目に87、２番目に45,
３番目に72、４番目に100を代入する場合

Tokuten = [87,45,72,100]

算術演算

加減乗除の四則演算は「+」、「-」、「*」、「/」で指定する。
整数の除算では、商を「÷」、余りを「%」で計算することもできる。
べき乗は「**」で表す

- 例）atai = 7 / 2　　変数ataiに3.5が代入される
- 例）syo = 7 ÷ 2　　変数syoに3が代入される
- 例）amari = 10%3　　変数amariに1が代入される
- 例）beki = 2**3　　変数bekiに2を3乗した8が代入される

演算子の優先順位は数学と同じで
「*」、「/」、「÷」、「%」、は「+」、「-」より先に計算される
また丸括弧「(」、「)」で式をくくって、演算の順序を明示することができる

比較演算（数値）

- 数値の比較演算は、
  「==」、「!=」、「>」、「>=」、「<」、「<=」で指定する。
  演算結果は、真か偽の値になる。
- 例）kosu > 3　　kosuが3より大きければ真となる。
- 例）ninzu*2 <= 8　ninzuの2倍が8以下であれば真となる。
- 例）kaisu != 0　　kaisuが0でなければ真となる。
- 例）kaisu == 0　　kaisuが0ならば真となる

比較演算（文字列）

- 文字列の比較演算は、「==」、「!=」を使うことができる。

「==」は、左辺と右辺が同じ文字列の場合に真となり、それ以外の場合は偽となる。

「!=」は、左辺と右辺が異なる文字列の場合に真となり、それ以外の場合（同じ文字列の場合）は偽となる。

- 例）”あいうえお” == “あいうえお”　　真となる
- 例）”AbCd” == “abcd”　　偽となる
- 例）”ABCD” != ”ABCD”　　偽となる
- 例）”Abdh” != “APPle”　　真となる

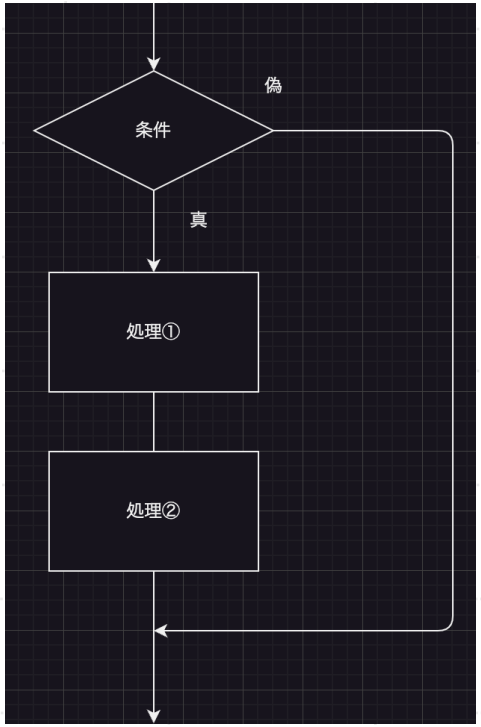
論理演算

- 真か偽を返す式に対する演算で、
  「and」（論理積）、「or」（論理和）、「not」（否定）の演算子で指定する。
- 「<式１> and <式２>」は、<式１>と<式２>の結果がいずれも真である場合に真となり、それ以外の場合は偽となる。
  例）kosu >= 12 and kosu <= 27　　kosuが12以上27以下なら真となる
- 「<式１> or <式２>」は、<式１>と<式２>の結果のどちらかが真である場合に真となり、それ以外の場合は偽となる。
  例）kosu%2 == 0 or kosu < 0
  kosuを2で割った余りが0になる場合とkosuが0未満の場合真となる。
  つまり、kosuが偶数か負の値なら真となり、それ以外の場合は偽となる。

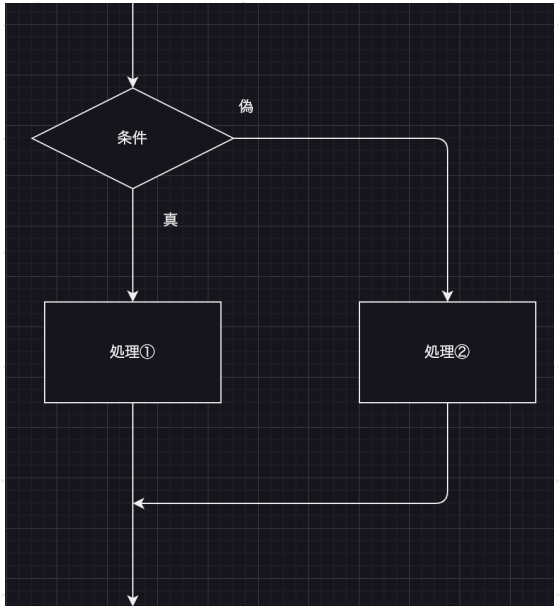
条件文

- <条件>が成立するかどうかによって、実行する処理を切り替える制御範囲を|で表し、範囲の最後を|で表す。

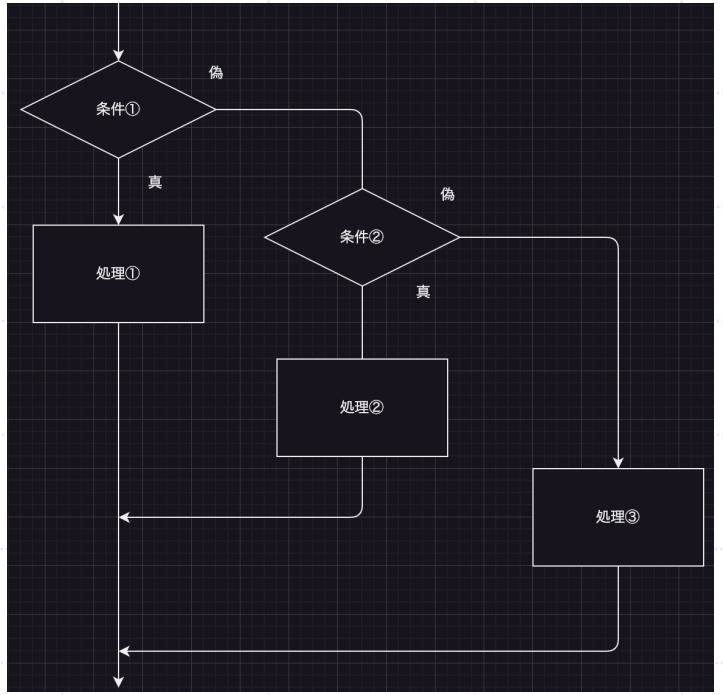
もし<条件>ならば:
|　<処理①>
|　<処理②>



もし<条件>ならば:
|　<処理①>
そうでなければ:
|　<処理②>



もし<条件①>ならば:
|　<処理①>
そうでなくもし<条件②>ならば:
|　<処理②>
そうでなければ:
|　<処理③>



条件文

```
atai = 11.37
Index = 10.97

もし atai == Index ならば:
|  表示する(atai,“は基準値と等しいです”)
|
そうでなくもし atai < Index ならば:
|  表示する(atai,“は基準値より小さい値です”)
|
そうでなくもし atai > Index ならば:
|   表示する(atai,”は基準値より大きい値です”)
|
そうでなければ:
|  表示する(atai,”は好ましくない値です”)


もし atai == 11.37 ならば:
|  atai = atai + 0.63
|
そうでなければ:
|   atai = atai – 3.14

表示する(“最終的な値は：”,atai)
```

解説

ataiという変数を宣言 ataiに11.37を代入
indexという変数を宣言 indexに10.97

ataiとindexの値が等しいなら:
| ataiの中身を表示し「は基準値と等しいです」と表示する
|
ataiとindexの中身が等しくなく、indexの方が大きければ:
| ataiの中身を表示し「は基準値より小さい値です」と表示する
|
ataiとindexの中身が等しくなく、さらにindexの方が大きくなく、ataiの方がindexより大きい場合:
| ataiの中身を表示し「は基準値より大きい値です」と表示する
|
ataiとindexの中身が等しくなく、さらにindexの方が大きくなく、さらにataiの方がindexより大きくもない、つまり何にも当てはまらない場合:
| ataiの中身を表示し「は好ましくない値です」と表示する

ataiの中身が11.37なら:
| atai に ataiの中身に0.63を足した値を代入する
|
そうでなければ:
| atai に ataiの中身から3.14を引いた値を代入する

「最終的な値は：」と表示し、ataiの中身を表示する

条件繰り返し文

- <条件>が成り立つ間、<処理>を繰り返し実行する
- <処理>を実行する前に、<条件>が成り立つかどうか**判定**されるため、<処理>が1回も実行されないこともある。
- 例）

```
n<10 の間繰り返す:
|   sum = sum + n
|    n = n + 1
```

順次繰り返し文

- <変数>の値を増やしながら、<処理>を繰り返し実行する。
  <変数>の値を減らしながら、行うこともある

  <変数>を<初期値>から<終了値>まで<差分>ずつ増やしながら繰り返す:
  |　<処理①>
  |　<処理②>

  例）
  x を 0 から 9 まで 1 ずつ増やしながら繰り返す:
  |　sum = sum + atai[x]

関数（値を返却しない）

- 処理の塊のようなもの
  例として ”りんご” と表示する関数 【apples()】を用意する

- 例）[apples()の中身]
  関数 apples() を定義する:
  　表示する(“りんご”)

- 例）
  apples()

「りんご」と表示される

関数（値を返却する）

- 処理の塊で値を返すことができるもの
  例として ２つの値をを入力させ、
  それらを足し合わせたものを返す関数【plus()】を用意する

- 例）[plus()の中身]
  関数 plus(a,b) を定義する:
  　a = a + b
  　a を返す

- 例）
  atai = plus(2,4)
  表示する(atai)　　6と表示される