# SPD と SPD エディタ



## 1. SPD の発祥と再生

1980 年代、NEC は自社のメインフレームやミニコン向けの開発支援環境を展開していましたが、その設計フェーズを支える中核図法として使われていたのが SPD（Structured Programming Diagram、構造化プログラム設計図）です。

SPD は、当初から罫線文字（├ └ ─ │ ）や記号（◇ ↻）を使って、テキストとして書ける図法でしたが、それを生かすには NEC 製の専用ツールが必要でした。そのため、SPD の知識があってもツールが使えない場合は、手書きするしかありませんでした（P.〇参照）。

SPD は社内向けの技術だったので、やがて業界のトレンドがオブジェクト指向と UML（Unified Modeling Language）へと移っていく中で、あまり知られないままフェードアウトしていきました。

著者はプログラミング教育のために長い間 SPD を利用しています。アルゴリズムを考えるツールとしてとても有効なので、（もっぱら紙に手書きする方法で）教えてきました。SPD を使うと大枠から徐々に詳細化していく段階的詳細化の手法が自然に身に付くからです。

しかし、ユニコード文字の普及と AI エージェントの急速な発達により、事情は一変しました。SPD を書くと、AI エージェントに依頼して、そのままソースコードに変換できるようになったからです。

## 2. SPD とは

SPD は、プログラムの処理をわかりやすいことばで系統的に表記したものです。処理の構造を示すために分岐や反復の記号を使って、ツリー状にまとめます。

最初に処理の骨格を考え、内容を徐々に詳細化していく、段階的詳細化の考え方を、図の中に自然に表現できます。アルゴリズムを考える力を養成する目的にも、とても効果的な図法です。

**SPD の特徴**

- プログラムの構造を**視覚的**に表現できる
- **段階的詳細化**によるプログラム設計が自然にできる
- **アルゴリズムを考える力**が身に付く
- わかりやすい日本語で記述するので**理解が容易**である
- プログラムとほぼ**一対一に対応**する
- 普通の**テキスト**として作成できる
- AI エージェントを使って**ソースコードに変換できる**
- **AI と協働するリテラシーの育成に適している**
- **いろいろなプログラミング言語（Python、Java、C など）で利用できる**

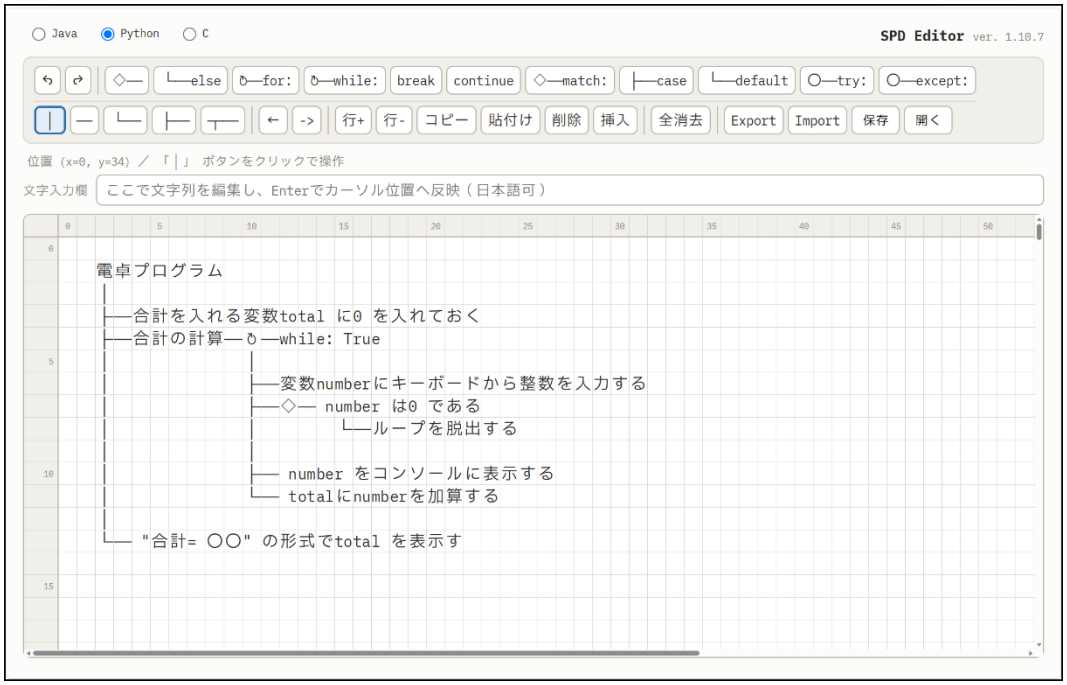
**簡単な SPD の例**

```
電卓プログラム
 │
 ├─合計を入れる変数 total に 0 を入れておく
 ├─合計の計算─↻─while: True
 │               │
 │               ├─変数 number にキーボードから整数を入力する
 │               ├─◇─number は 0 である
 │               │        └─ループを脱出する
 │               │
 │               ├─number をコンソールに表示する
 │               └─total に number を加算する
 │
 └─”合計= ○○” の形式で total を表示する
```

# 3. SPD エディタについて

SPD は、罫線、記号、文字の組み合わせなので、エディタやワープロでも作成できますが、SPD エディタを使うと、もっと軽快に作成できます。また、修正や再利用も簡単です。

**SPD エディタ　（https://k-webs.jp/spd/spd-editor.html）**




SPD エディタは、HTML + CSS + JavaScript による単一の HTML ファイルです。上記の URL を開くだけで使えます。著者の GitHub（https://github.com/kawaba/spd）からソースをダウンロードするとローカルでも起動できます。**誰でも無償で使用できます**。

使い方は、「**5. SPD エディタの使い方**」を見てください。

**フォントのインストールについて**

SPD エディタは、等幅フォントで、全角スペースを□で表示しない PlemolJP HS フォントを優先フォントとしています。OS 固有のフォントでも使用できますが、このフォントをインストールするとより快適に使えます。次の URL からダウンロードできます。

https://github.com/yuru7/PlemolJP/releases/tag/v3.0.0

URL を開いて、PlemolJP_HS_v3.○.○.zip をダウンロードし展開します。いくつかのフォントセットが含まれていますが、”35”や”console”などの接尾辞が付かないフォルダを開き、PlemolJPHS-Regular.ttf と PlemolJPHS-Bold.ttf の 2 つをインストールしてください。

# 4. SPD の書き方

## (1) 基本構造

タイトルを書いて、具体的な処理を上から下へ順に、罫線でつないだものが基本の SPD です。プログラムの知識が無い人でもわかるような、平易なことばで書きます。

```
代入と計算（順次構造）
 │
 ├─変数 number に 10 を代入する
 ├─number を 1 増やす
 └─mumber を表示する
```

## (2) 選択構造

◇ は条件判断の記号です。if 文や match 文で使います。”◇─条件” のように書き、条件から └─ の罫線を引いて実行する内容を書きます。また、”それ以外”は、”└─else”です。

```
サイコロゲーム（if-else 文）
 │
 ├─tkxlib から dice 関数をインポートする
 ├─サイコロを振って（dice()）出目を変数 number に代入する　※ dice()は tkxlib の関数
 └─◇─number は 6 である
    │      └─ ”当たり”と表示する
    └─else
           └─”はずれ”と表示する
```

※は、SPD では**コメント**です。※から行末までがコメントになります。

多重の選択構造は◇─ を縦に並べます。

```
身長の階級（if-else if 文）
 │
 ├─キーボードから身長（cm）を変数 height に入力する
 │
 └─身長の階級を求める─◇─180cm 以上
                       │      └─”A”と表示する
                      ◇─170cm 以上
                       │      └─”B”と表示する
                      ◇─160cm 以上
                       │      └─”C”と表示する
                       └─else
                              └─”D”と表示する
```

### (3) 反復（繰り返し）構造

↻ は繰り返しの記号です。for 文は "↻—for:反復範囲"、while 文は"↻—while:条件"のように使います。反復実行する内容は、↻記号の下に └— を引いて書きます。

```
リストの要素を表示する（for 文、表示処理）
 │
 ├—整数のリストを変数 numbers に代入する— numbers ← [10, 20, 30]
 │
 └—全ての要素を表示する—↻—for:n ← numbers
                          │
                          └— n をコンソールに表示する
```

”n ← numbers”は、「取り出す要素がある間、numbers から毎回 1 つずつ要素を取り出して変数 n に代入する」をコンパクトに表現する書き方です。Java でも Python でも同じ書き方ができます。

繰り返し処理の内容が複数ある場合は、├— を使って複数の枝を作ります。次の例は while 文の例です。

```
電卓プログラム（while 文、無限ループのパターン）
 │
 ├—合計を求める┬— 合計を入れる変数 total に 0 を代入する
 │             │
 │             └—↻—while:True
 │                  │
 │                  ├—変数 number にキーボードから整数を入力する
 │                  ├—◇—number は 0 である
 │                  │        └— ループを脱出する
 │                  │
 │                  ├—number をコンソールに表示する
 │                  └—total に number を加算する
 │
 └— ”合計= ○○”の形式で total を表示する
```

## (4) 段階的詳細化

まず、処理の骨格（部分的な処理のタイトル）を先に決めて、全体を完成し、詳細は後から考えます。

```
段階的詳細化の例示（骨格）
 │
 ├─準備
 │
 ├─集計処理
 │
 └─結果の表示
```

次に､詳細を書き足して完成します｡このような書き方を**段階的詳細化**といいます。部分ごとに詳細化すればよいので、手順を考えやすくなります。

★SPD エディタでは、段階的詳細化をしやすいように、後から行を挿入したり削除したりが簡単にできるようになっています。行の挿入、削除に応じて**縦罫線が自動補完されます**。

```
段階的詳細化の例示（詳細化後）
 │
 ├─準備─変数 total に 0 を入れる
 │
 ├─集計処理─↻─while:True
 │              │
 │              ├─変数 data にキーボードから整数を代入する
 │              ├─◇─data は 0 である
 │              │        └─ループを脱出する
 │              └─total に data を加える
 │
 └─結果の表示─total を”合計＝〇〇”の形式で表示する
```

SPD が横に広がりすぎた場合は、**枝を折り曲げる**とコンパクトになります。

```
段階的詳細化の例示（枝を折り曲げた形）
 │
 ├─準備─変数 total に 0 を入れる
 │
 ├─集計処理
 │    └─↻─while:True
 │          │
 │          ├─変数 data にキーボードから整数を代入する
 │          ├─◇─data は 0 である
 │          │        └─ループを脱出する
 │          └─total に data を加える
 │
 └─結果の表示─total を”合計＝〇〇”の形式で表示する
```

## (5) 書き方の Tips

**・積極的に段階的詳細化を使う。**

最初に骨格（部分的な処理のタイトル）を作ると、処理の手順を作成しやすくなります。また、処理内容もわかりやすくなります。

**・プログラムのコードをそのまま書くのではなく、わかりやすいことばで表現する。**

— 変数 total に 0 を代入する
— キーボードから変数 number に整数を入力する
— 平均値を小数点以下 1 桁に丸めて、”平均値＝〇〇.〇”の形式で表示する。
— p の属性のうち商品名（p.name）と価格（p.price）をコロンとタブで区切って表示する

★具体的な書式文字列などを書くのは避けます。細かな文法要素は AI が生成します。

**・SPD にデータを記述する方法。**

データ操作をことばで書いておき、 └— を引いて詳細説明として代入操作を書きます。

— 文字列のリストを作って変数 data に代入する。

```
└— data ← ["apple", "banana", "cherry"]
```

（注）＝の代わりに←を使うと、Java でも Python でも同じ表現で済みます。

— 辞書のリスト users を作成する。※Python

```
└— users = [{"id": 103, "name": "tanaka", "role": "admin"},
            {"id": 105, "name": "sasaki", "role": "user"},
            {"id": 100, "name": "maeda", "role": "user"},]
```

**・for 文では積極的に←を使う。**

```
—↻—for:fruit ← fruit_list          ★リストからの要素の取り出し
—↻—for:key, value ← score.items()  ★辞書からの要素の取り出し(Python)
```

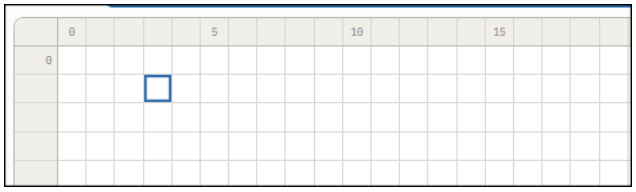
★この他の書き方については、書籍「**わかりやすい Python**」（2026.9 秀和新社）をご覧ください。Java についても順次書籍化する予定です。また、SPD エディタで💡ボタンを押すと、すべての書き方のパターンと生成されるソースコードを対比して見ることができます。

# 5. SPD エディタの使い方

（注）SPD エディタは将来のバージョンアップにより仕様が変更される場合があります。

## (1) セルポインタ

エディタ上の□をセルポインタといいます。

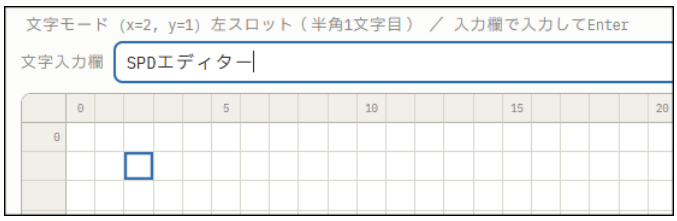


文字入力中でなければ、

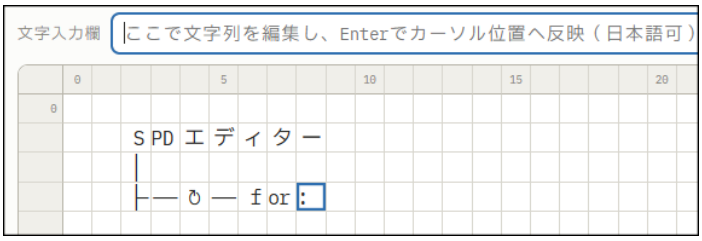
・矢印キーでエディタ上を**移動**できます。

## (2) 文字と罫線の入力

・セルを**ダブルクリック**すると入力モードになり、文字入力欄で文字を入力できます。
Enter キーでセル位置に挿入されます。



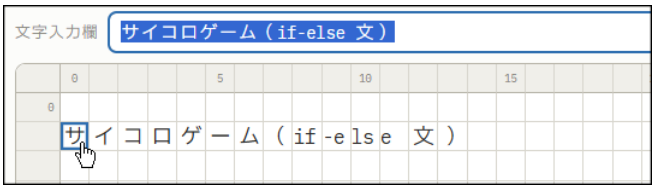

・SPD 記号や罫線ボタンを押すと、セルポインタのある位置に入力されます。




・SPD 記号や一部の罫線を入力すると、自動的に文字入力モードになります。

・**Esc** を押すと文字入力モードを解除できます。

・すでに文字が入力されているセルをダブルクリックすると、文字列全体を入力欄で編集できます。




### (3) 文字の削除と消去（文字入力モードでない時）

・**Delete**、**Backspace** キーを押すとセルポインタの位置の文字が**削除**されます。
・Shift キーを押しながら Delete、Backspace キーを押すと文字が**消去**されます。
（消去では文字と空白が置き換わるだけで、左側の文字は詰められません。）

### (4) ツールバー


1 段目には SPD 記号を入力するボタンがあります。2 段目には罫線入力ボタン、代入(←)、型ヒント記号（->)、その他の機能ボタンがあります。右端の保存、開くのボタンは作成した SPD データの保存と読み込みです。

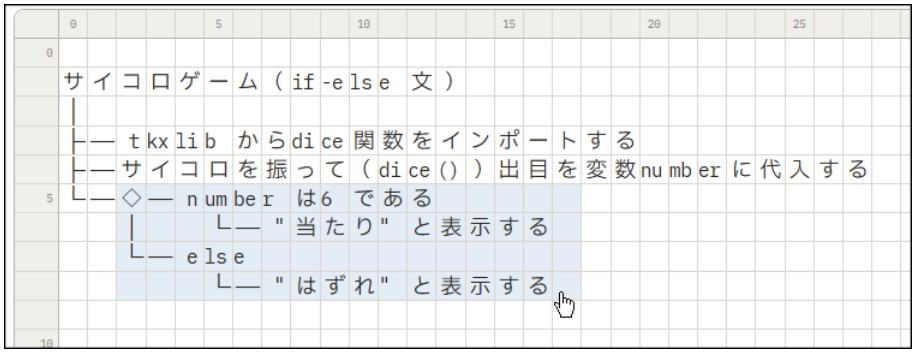
★言語選択のラジオボタンを押すと各言語用に表示されるボタンが変わります。

**機能ボタンの機能**

| 機能ボタン | 機　能 |
|---|---|
| ↶ | 元に戻す |
| ↷ | やり直し |
| 行＋ | セルポインタがある行の上に、必要な罫線を補完して 1 行挿入する |
| 行ー | セルポインタがある行を削除する |
| コピー | 選択した範囲をコピーする |
| 貼付け | コピーした内容をセルポインタのある位置に貼り付ける |
| 削除 | 文字削除、連続して押すと複数文字を削除できる |
| 挿入 | 文字挿入、連続して押すと複数の空白文字を挿入できる |
| 全消去 | 画面全体をクリアする |
| Export | SPD 全体をクリップボードにコピーする |
| Import | クリップボードにある SPD を取り込む |
| Shift + Export | 選択範囲をクリップボードにコピーする |
| Shift + Import | 現在のセル位置に SPD を取り込む |
| 保存 | SPD を JSON 形式でファイルに保存する |
| 開く | SPD を JSON 形式のファイルから読み込む |

## (5) 範囲選択とコピー、移動・消去

・マウスでドラッグすると**範囲選択**できます。




・**コピーボタン**で選択範囲を**コピー**できます。
・**貼付けボタン**でコピーした内容を、セルポインタのある位置に**貼り付け**できます。
・範囲選択直後に、他のセルをクリックするとその位置へ選択した内容が**移動**します。
・文字入力モードでない時は、選択した範囲を **Delete** または **Backspace** で**消去**できます。

### (6) VS Code などのエディタとのやり取り

・Export ボタンを押すと、作成した SPD 全体をクリップボードにコピーします。

・範囲を選択し、Shift キーを押しながら Export ボタンを押すと、選択した範囲だけをクリップボードにコピーできます。

・VS Code ではメニューで、「編集」⇒「貼り付け」により SPD を貼り付けることができます。

・逆に、VS Code のエディタで、「編集」⇒「コピー」とすると、それを Import ボタンで取り込むことができます。

・特定のセルをクリックし、Shift キーを押しながら Import ボタンを押すと、コピーした内容をセルの位置に取り込むことができます。既存の SPD はそのまま残ります。

### (7) ファイルに保存・ファイルから読み出す

・保存ボタンで、作成した SPD をファイルへ保存できます。
保存形式は JSON です。
・開くボタンで、保存していた SPD を読み込むことができます。