この本の目的

私は、知的生産術の良い参考書が欲しいです。人に知的生産術を教えるときに、お勧めできる本が欲しいです。

私は、サイボウズで知的生産性の研究に10年間従事してきました[注1]。業務の一環として、京都大学サマーデザインスクールで、考えを整理してアウトプットする方法のワークショップを行ったり、首都大学東京の非常勤講師として、大学生に研究によって新たな知識を生み出すことについて教えたりしてきました。しかし、限られた時間では伝えたいことが伝えきれません。参考書を紹介しても、たくさん紹介したのでは全部は読んでもらえません。私の伝えたいことが1冊にまとまった本が欲しいです。

でも、ちょうど良い本がないんです。何か1冊だけお勧めするなら川喜田二郎の『発想法』[注2]ですが、これは1966年の本です。抽象的な考え方は今でも十分有効ですが、具体的な方法論が50年前の技術水準を前提にしていて、古臭く感じてしまう人も多いです。次の50年のための本が必要です。

ないなら作りましょう。

私は、プログラミング言語を比較して学ぶ『コーディングを支える技術』[注3]を2013年に執筆しました。この本は発売から5年経つ今でも言及され、売れ続けているロングセラーです。執筆の過程で何か知的生産が行われたのは間違いないでしょう。ならば執筆の過程で使われた手法を、ほかの人にも使えるように解説しましょう。

『コーディングを支える技術』の執筆時には、川喜田二郎のKJ法をベースとした手法を使いました。また書籍の内容では、「複数のプログラミング言語を比較することで、何が変わる部分で、何が変わらない部分かを理解しよう」というアプローチと、「プログラミング言語は人が作ったものだから、何か目的があって作ったはずだ。目的に注目しよう」というアプローチを使

注1　サイボウズはグループウェアを開発しているソフトウェアメーカーです。グループウェアは、複数人のグループで使うためのソフトウェアです。なぜ企業がグループウェアを導入するのかというと、それによって社員の生産性が上がるからです。私はグループウェアが、単純な作業の生産性を上げるだけでなく、知識を生み出すような仕事の生産性も上げると考えており、またそこに注力していくべきだと考えています。

注2　川喜田二郎著『発想法――創造性開発のために』中央公論新社、1966年。改訂版の『発想法 改版――創造性開発のために』が2017年に出ています。

注3　西尾泰和著『コーディングを支える技術』技術評論社、2013年

いました。今度は「プログラミング言語」の代わりに「知的生産術」に対して同じことをやってみましょう――こうして本書の企画がスタートしました。

知的生産とは何か

知的生産とは、知識を用いて価値を生み出すことです。具体的には執筆やプログラミングなどがイメージしやすいでしょう。しかしそのほかの仕事でも、ありとあらゆるところに知的生産の機会があります。私は、自ら新しい知識を生み出すことが、価値の高い知的生産のために重要だと考えています。他人から与えられた知識を使うだけでは、大した価値にはなりません。

プログラミングの例で考えてみましょう。教科書のサンプルコードをコピーするだけでは、多くの場合あなたの達成したい目的を達成できません。目的を果たすためには、サンプルコードを噛み砕いて理解し、あなたの置かれた状況に合わせて、修正し、組み合わせ、新しいプログラムを作る必要があります。知的生産術についても同じです。本に書いてある知識をコピーするだけではなく、修正し、組み合わせ、新しい手法を作ることが必要です。

この本を読むメリット

読者は、この本を読むことで知的生産術について学ぶことができます。知的生産術を学ぶうえで、私はこの本が一番お勧めです。

この本の原稿をレビューしてくれた方のコメントを紹介します。

- **気付いていなかったことに気付けた**
- **無意識にやっていたことが言語化できた**
- **これから役に立ちそうなことが満載でとてもやる気が出た**

この本の刺激によって盲点に気付いたり、自分の経験が言語化されたりして、今後さらに改善できそうだと感じ、ワクワクしているわけです。

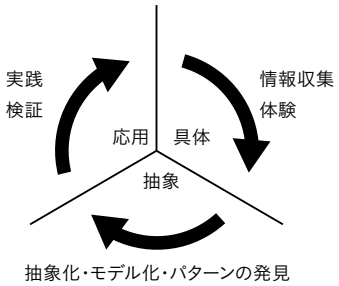
一方で逆の意見もあります。

- **地に足が付いていない**
- **「で、どうしたらよいの？」がわからない**

材料がそろっていないと、結合は起きません。「地」は経験です。本書を読んでしっくりこなかったなら、今回は残念ながら材料が足りなかったようです。でも大丈夫です。経験は日々あなたの中に蓄積されていくので、いつか「あ、これか」とつながるときが来るでしょう。半年経ってからまた読みなおしてみてください。きっと何かが変わるでしょう。

プログラミングはどうやって学ぶか

知的生産術について考えていくうえで、「学び方」について学ぶことは避けられません。しかしこれは抽象的なので、まずはもっと具体的に、プログラミングの学び方について考えてみましょう。私はプログラミングの学びのプロセスは「情報収集・モデル化・検証」の3要素の繰り返しだと考えています[注4]。



学びは情報収集・モデル化・検証の繰り返し

まずは具体的に情報収集する

最初の一歩は、具体的に情報収集することです。

プログラミングを学ぶとき、多くの場合、まずほかの人が書いたプログ

注4　初出: 西尾 泰和著「エンジニアの学び方」『WEB+DB PRESS Vol.80』、技術評論社、2014年

ラムを読みます。また読むだけではなく、まねて入力するテクニック「写経」もよく使われています。これが「具体的な情報収集」です。本書でもいくつもの知的生産にまつわる課題とその解決策を紹介します。これはプログラミングのサンプルコードのようなものです。

抽象化してモデルを作る

具体的な情報収集が進んであなたの脳の中に材料がそろってくると、次に「抽象化」が起きます。抽象化は、複数の具体的情報から共通するパターンを見いだすことや、どこが重要でどこが枝葉かを判断することと、強い関連を持っています。

複数のものを見比べ、共通点を見いだすことが、抽象化の助けになります。ここで「Hello, world! と表示するコード」をプログラミング言語Pythonで書いたものを見てみましょう。

Hello, world! と表示するコード1

```
print("Hello, world!")
```

出力結果

```
Hello, world!
```

Pythonの経験がない人でも、このコードと出力結果とを見比べると、共通部分があることに気付くことでしょう。これがパターンの発見です。

あなたが見いだしたパターン

```
print("きっとここに書いたものが表示される")
```

もし「Bye, world! と出力したい」なら、どこをどう書き換えればよいかわかりますね。つまり、パターンを発見したことで、自分の目的に合わせてプログラムを修正する能力を得たわけです。

次のコードもPythonで書かれた「Hello, world! と表示するコード」ですが、これを初めてPythonを学ぶ人に見せても、修正できるようにはならないでしょう。出力の「Hello, world!」とソースコードの間に共通点を見つけることができないからです[注5]。

注5　勘の良い人ならコード1とコード2を比較して、"".join から始まる複雑なコードが "Hello, world!" という文字列を作っているのではないかと思うかもしれません。それは正解ですが、本書はPythonの教科書ではないので詳しい解説はしません。

Hello, world! と表示するコード2
```
print("".join(map(chr, [72, 101, 108, 108, 111, 44, 32, 119, 111, 114, 108,
 100, 33])))
```

この本では、知的生産術を比較することで、みなさんが自分の中に知的生産術のモデルを作ることを手助けしたいと思っています。

実践して検証する

さて、ここまででであなたはパターンを見いだし、「Python で Bye, world! と出力したい場合には、こう書けばよいのではないか？」と思い付く能力を手に入れました。

あなたが見いだしたパターン
```
print("きっとここに書いたものが表示される")
```

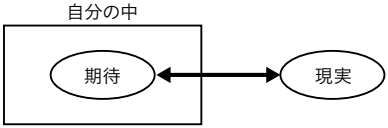
「Bye, world!」と出力するだろうと思われるコード
```
print("Bye, world!")
```

しかし、これはあくまで仮説です。本当にその方法で「Bye, world! と出力する」という目的が達成できるかは、実際に試してみなければわかりません。試してみると、この具体例に関しては、目的が達成できることがわかります。

では複数行に分けて「Hello,（改行）world!」と出力したい場合はどうでしょう？ 仮説が正しいなら、こう書けばよいはずです。

「Hello,（改行）world!」と出力するだろうと思われるコード
```
print("Hello,
world!")
```

試してみると、このコードは構文エラーになり、期待どおりに動かないことがわかります。これは失敗ではありません。「この方法ではうまくいかない」という具体的情報を発見したのです。これは学びのチャンスです。「どうして思ったように動かないのか？」――こう考えることで、あなたは理解を深めていくことができます。



期待と現実のギャップを発見した

この例では「改行をprintしたいとき、単に改行したのではうまくいかない」という事実を発見しました。次は、この課題の解決方法を探します。たとえば検索エンジンなどで探す[注6]と、役に立ちそうな情報を含んだサンプルコードが見つかるでしょう。そのサンプルコードは、あなたの「Hello,(改行)world!」を出力するという目的を達成する方法を直接教えてくれるわけではありません。きっと何か別のものを表示するコードでしょう。しかし、この新しいサンプルコードを、今までにあなたが集めた情報と比較すれば、あなたはパターンを見いだし、「なるほど、じゃあこう実装すれば目的が達成できるはずだ」という新しい仮説を生み出すことができるでしょう。

具体的に情報収集し、比較してパターンを発見し、実践して検証し、期待と現実のギャップを発見して、また情報収集をしました。このサイクルを繰り返すことで、あなたはプログラミング能力を学ぶことができます。サイクルを繰り返すことで、新しいプログラムを生み出す力が鍛えられるわけです。知的生産術の学び方も同じです。具体的な情報収集、比較してパターン発見、実践して検証を繰り返すことで学んでいくのです。

この本の流れ

第1章「新しいことを学ぶには」では、正解のないことをどうすれば学べるのかについて考えます。丸暗記ではなく、状況に合わせた応用ができる

注6　具体的には、Google検索で"Python 改行 print"と検索するなどです。

ようになるために、サイクルを回して学んでいく方法を詳しく解説します。

学びのサイクルを回すためには、燃料として「やる気」が必要です。第2章「やる気を出すには」では、どうすればやる気が高い状態を維持できるかについて、12,000人以上のやる気が出ない人のデータを踏まえて解説します。

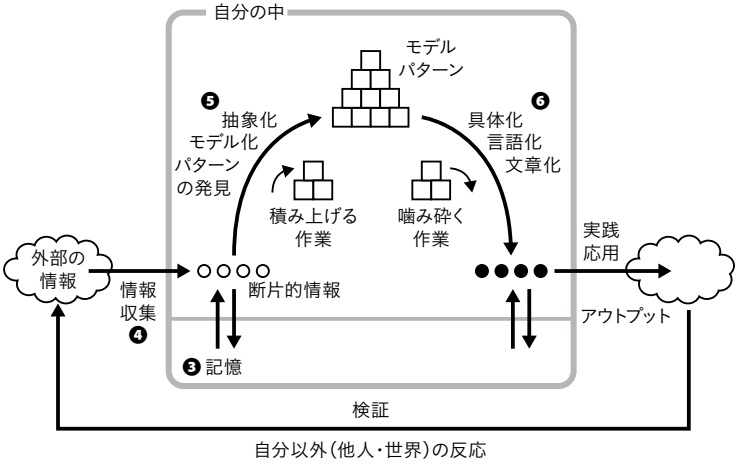
学んだことは覚えておきたいですよね。第3章「記憶を鍛えるには」では、脳というハードウェアのしくみや、学び方に関する実験結果、そしてソフトウェアによって可能になった記憶を効率良く定着させる方法について解説します。

本を買いすぎて、山積みになってしまう人も多いのではないでしょうか？第4章「効率的に読むには」では、本を読むことを中心とした情報のインプット効率の改善について、速読術とゆっくり読む方法を比較して考えます。

第5章「考えをまとめるには」は出力に関する話です。「学ぶ＝入力」と考えがちですが、出力して検証することは大事です。しかし、たくさんインプットして知識量が増えると、それを整理して人に伝えることに苦しみを感じるようになります。人間は脳内の知識を他人に丸ごとコピーできないので、脳内の知識ネットワークを、切り取り、束ね、並び替えて、言葉や図に変換していく作業が必要です。本章では、たくさんのインプットを他人にアウトプットできる形にまとめる方法として、川喜田二郎のKJ法と私の執筆方法をベースに解説します。

知的生産術は新しいアイデアを思い付く方法だ、と考える人も多いと思います。私は「アイデアを思い付く」と「理解を深める」と「パターンを発見する」には共通の要素があると考えています。第6章「アイデアを思い付くには」では、アイデアを思い付き実現する方法について考えます。

ほかにもいろいろ語りたいことは尽きませんが、ページ数は限られています。第1章〜第6章では「何を学ぶか」(what)は決まっている前提で、「どう学ぶか」(how)を説明します。しかし、私はhowよりもwhatのほうが大事な問いだと考えています。第7章「何を学ぶかを決めるには」では、この問いについて考えていきます。



学びのサイクルと第3章〜第6章の関係

謝辞

この書籍は、執筆段階からレビュアーのみなさんに協力いただき、1章3週間のイテレーションで各章をリリースしていく、アジャイル開発の手法を採用しました。感謝の意を込めてここで紹介します。

中山ところてん、近藤秀樹、Kuboon、湯村翼、假屋太郎、KUBOTA Yuji、原田惇、鈴木茂哉、加藤真一、庄司嘉織、渋川よしき、風穴江（@windhole）、hatone、安達央一郎、Takuya OHASHI（敬称略、順不同）

また、私の説明を辛抱強く聞いて、わかりにくいところを指摘し、改善案を提案してくれた妻にとても感謝しています。