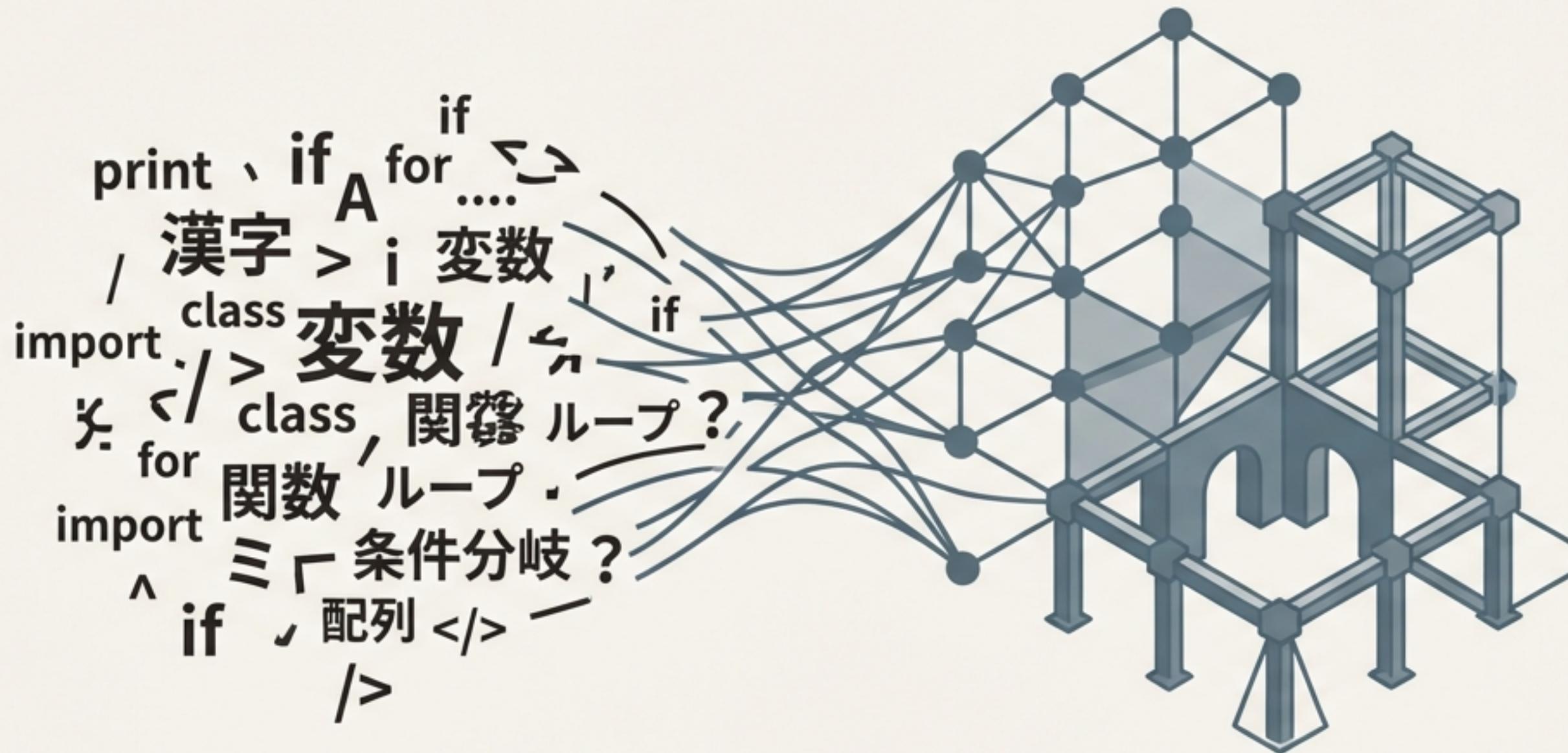


コードが「構造物」に見えた日

Python Web開発学習における視点の転換



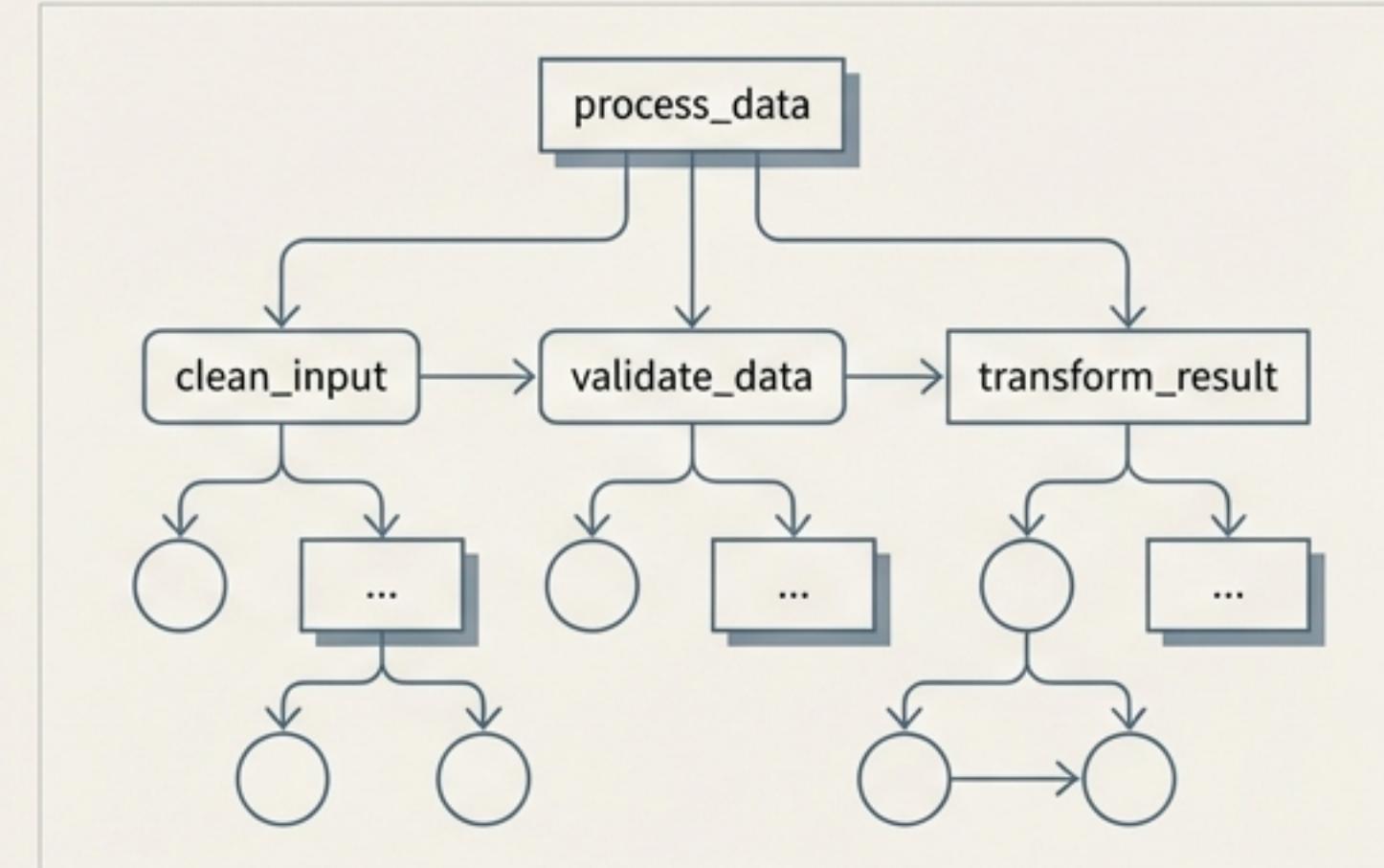
プログラムは「文字の塊」から「ノードグラフ」へ

Before

```
def process_data(input): if not input: return None temp = clean_input(input)
    validated = validate_data(temp) result = transform_result(validated)
    return result def clean_input(data): ... return None def clean_input(data): ...
    temp = process_noux(data) temp = validate_data(xdata): ... non otmuni:oroken
    validated def clean_input(data): ... moxerreturn None :transform_result(dat
    ta) validated temp = clean_input(ivainput): mon process_data(input);...val
    tto rant validate_data(temp) result = return None = process_input(data(
    temp) result = validate_result(misnilven(dado); temp = clean_input(input)
    onvn result ... def ciean_input(data): ... return None ... resurn x def clean_in
    put(data): ... clean_input(data): def transform_result(data=validated = results
    dao= proeess_input(data)temp = nox transform_result(validated)instont); def
    clean_input(data): ... waw validate_data = validate_data[temp) result =
    transform_result(validated) def clean_Input(data): ... r transform_resultdata
    )zomasom>ctum_ntozimiiis = clean_input(data); temp = resturn crnsors;input
    not :return var or=result = validate_data(temp) ... result = clean_input(input)
    ... trin\ clean_Input(input): ... return muw;data ... def clean_Input(data);
    resaxum o des itranofrm_msür anoane = validate_result[temp]:umiiny varelto
    forvn;dataattr: temp = clean_inputj ... warii = transform_resut(data) ... tanl
    validator = validate_data(temp) result = return None =hairrisal r:esult == clea
    n_input( ....: def cane_input(data): ... return (data); ... transform_result;
    cor clean_input(input)); ... result = clean_input(temp)[varonv] = transform
    result = transform_datal/validated = transform_resultivatiot'munati; def clea
    n_input(data); nxaf transform_resultisimep[tempe validate_data(temp) ... var
    ar metrm def iremp = transform_rosut(data); ... def clean_inpu(input); ...
```



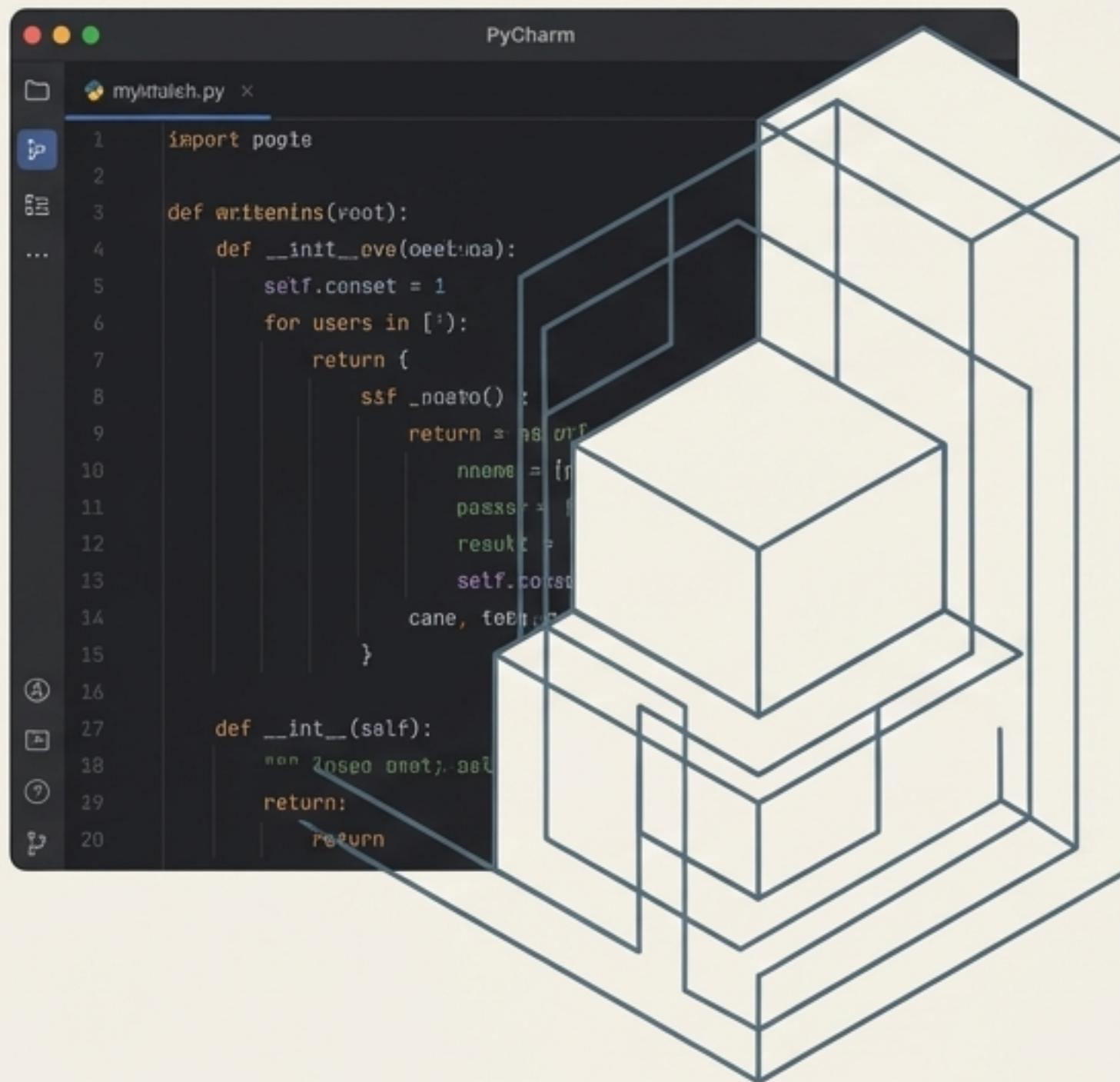
After



学習の初期段階では、プログラムは単なる「文字のかたまり」にしか見えませんでした。しかし、第3章の学習を通じて、コードが持つ視覚的な階層と繋がりを認識できるようになりました。IDEで見るコードは、まるで相互に接続されたブロックやノードで構成される「構造物」のように見え始めたのです。

「たくさん数をこなしていったらきっとIDEで見るコードが構造物の様に見えてくるのだろうなと思いました。」

転換のきっかけ：第3章 アドレス帳アプリ開発



この視点の転換は、第3章「アドレス帳アプリ」開発で、より深い階層構造を扱う中で起こりました。

特にPyCharmのインデント機能が、コードブロックの親子関係を視覚的に表現してくれたことが大きな要因です。

コードを書く行為が、テキストを打ち込む作業から、構造を組み立てる作業へと変化しました。

「Pycharmでのインデント移動に慣れてくるとコードの階層構造がビジュアル化でき、内包されている情報がブロックやノードの様に見えてきて…」

すべてが繋がり、流れが見える

この新しい視点により、これまで曖昧だった概念が明確になりました。
特に静的ファイルのような外部リソースの扱いが、「ロードし、アタッチし、適用する」という
具体的なプロセスとして理解できるようになります。
プログラム全体のデータフローが、まるでノードグラフ上を情報が流れていくように見えます。



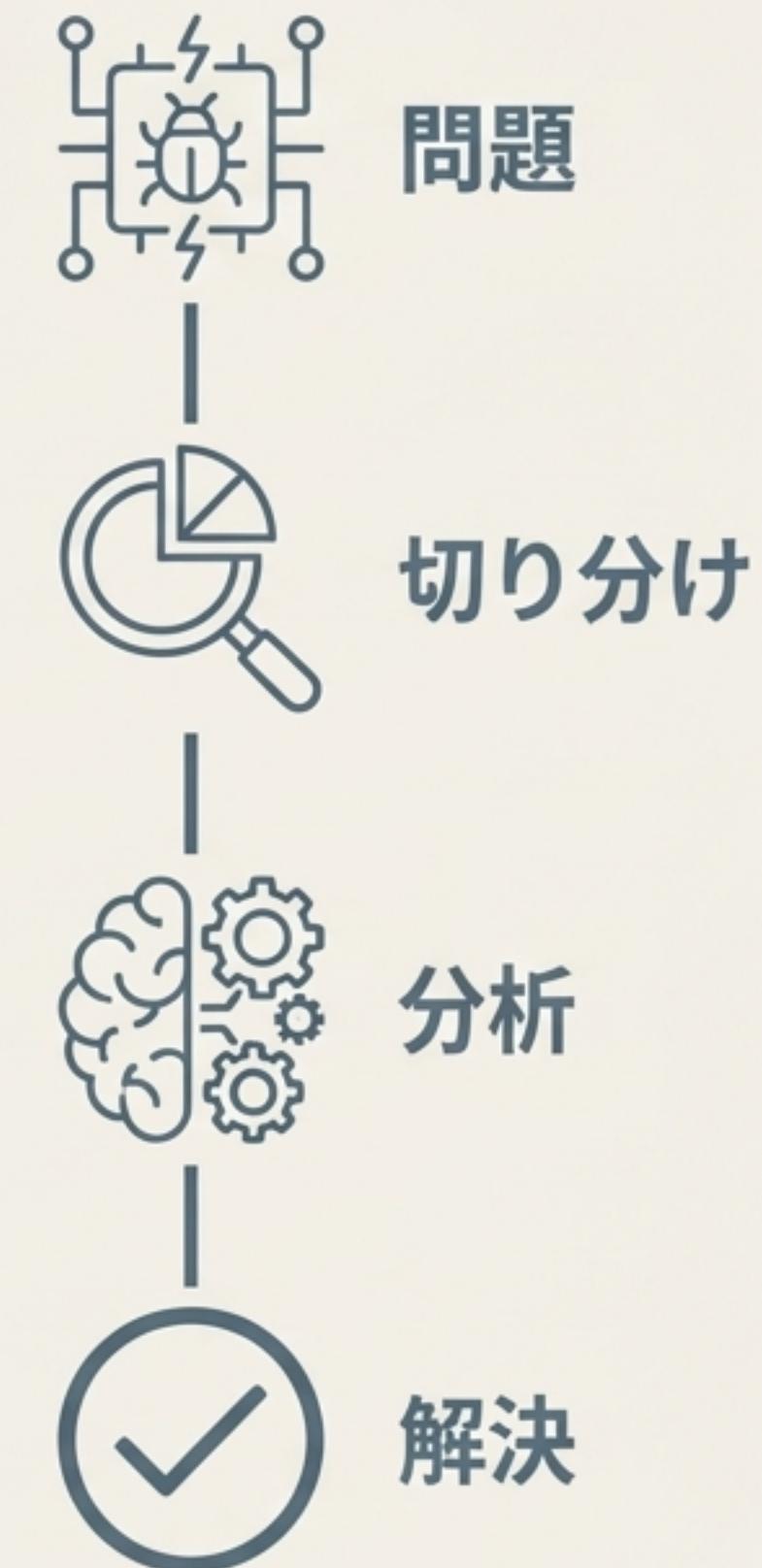
「上流から下流<出力>に流れていく様など、ノードグラフ
上のノードが繋がっている様に見えてきました。」

冷静なトラブルシューティング の実践

このメンタルモデルは、実践的な問題解決能力の向上にも繋がりました。学習途中でCSSの設定トラブルのが発生した際も、以前のように慌てることはありませんでした。コードの構造が見えているため、冷静に原因の切り分けを行い、適切な対策を検討・実行できました。

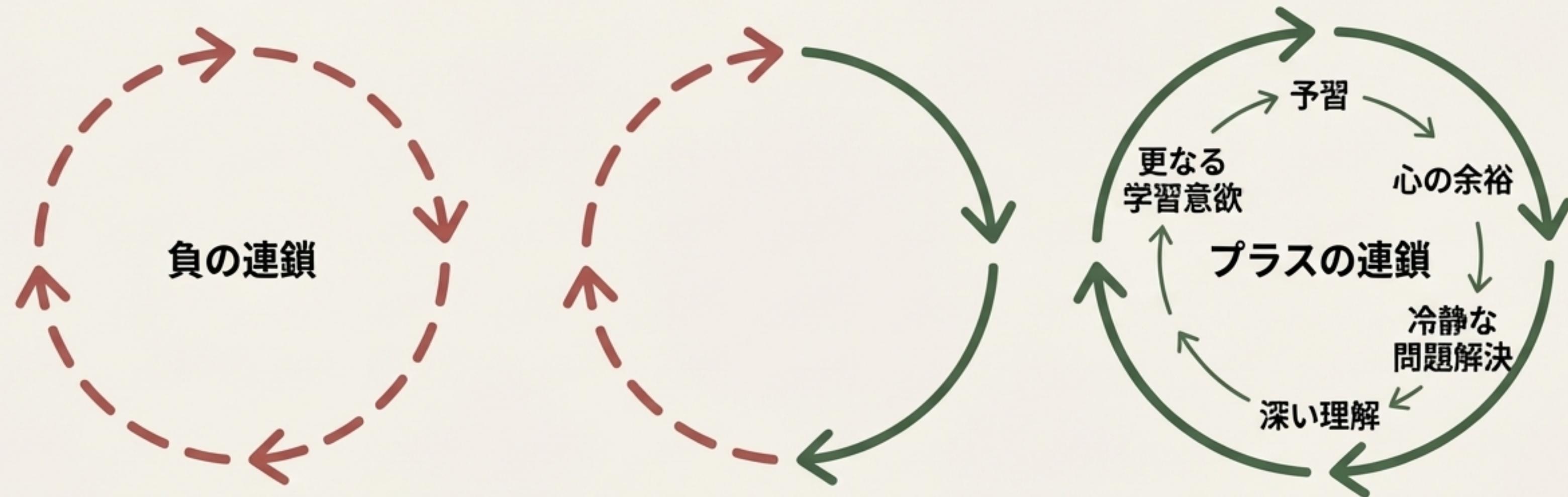
これは講義開始当初からの明確な成長です。

「講義を受け始めた頃とは違いやるべきことが見えてきてると思います」



成長のエンジン：「予習」によるプラスの連鎖

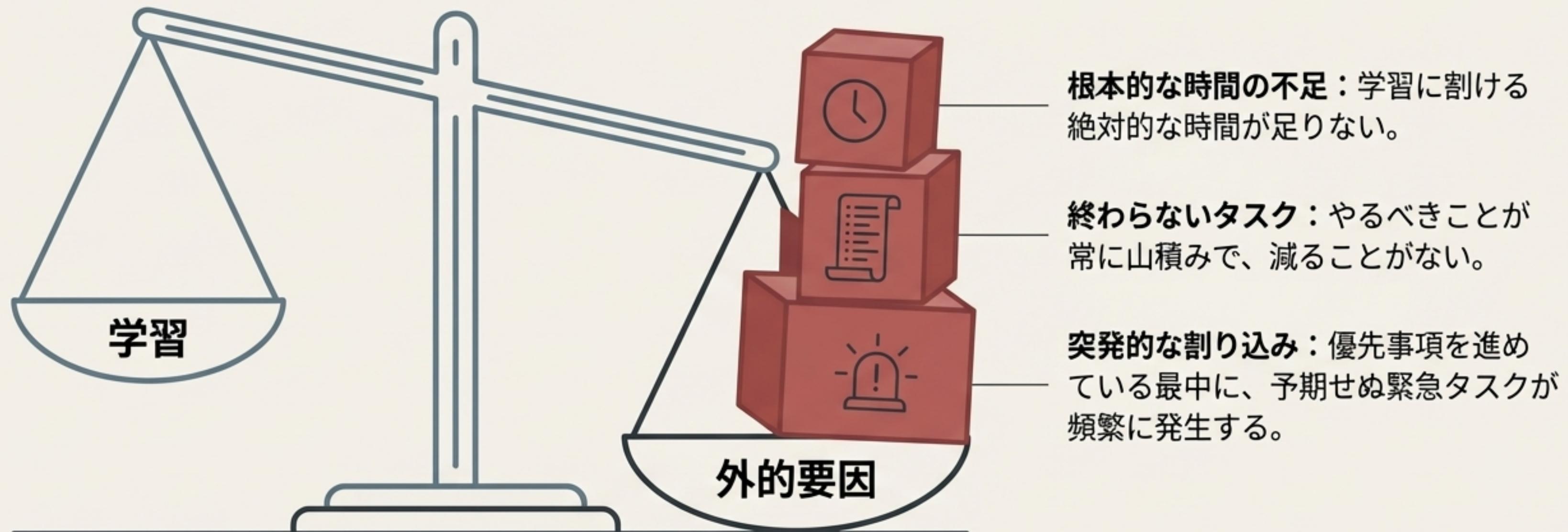
このようなブレークスルーは偶然ではありません。これまで継続してきた「予習」の成果です。事前に学習内容を把握しておくことで、トラブルが発生しても精神的な余裕が生まれ、「最悪アーカイブで取り返せる」という安心感が冷静な思考を支えました。



この準備が「負の連鎖」を断ち切り、「プラスの連鎖」を生み出しました。

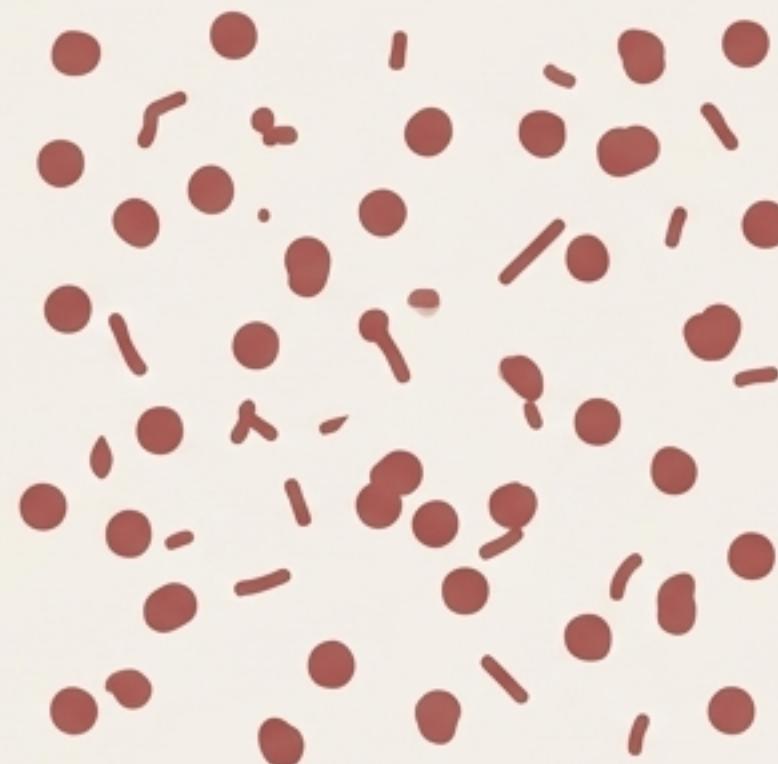
成長を阻む現実的な課題

ポジティブな変化の一方で、学習の継続を困難にする現実的な課題も存在します。これらの外的要因への対処が、今後の成長の鍵となります。



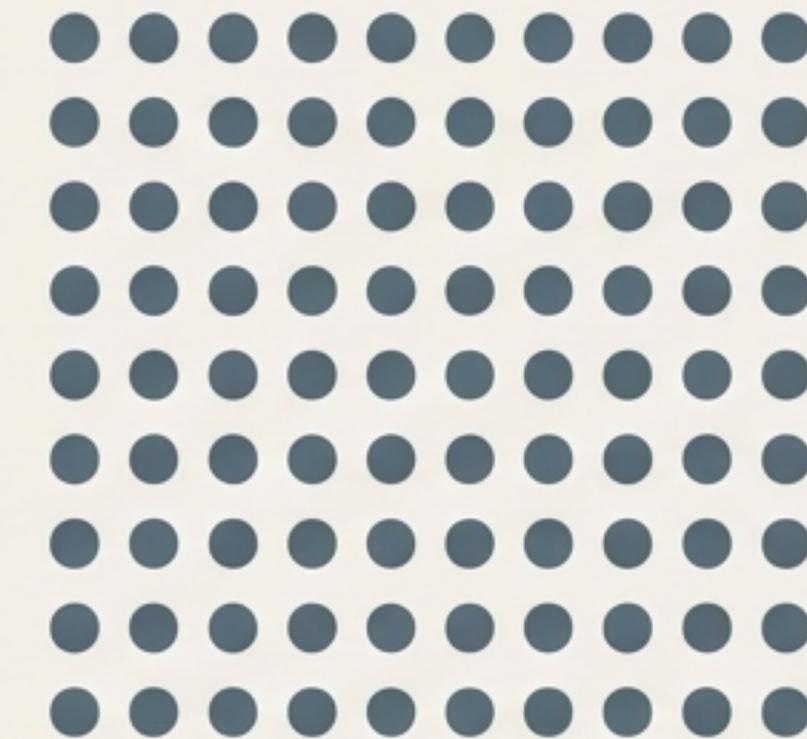
アウトプットに対する意識の変革

これまでの課題を踏まえ、アウトプットへのアプローチを根本的に見直す時期に来ていると感じています。これまで気分や勢いに任せた散発的なアウトプットでした。今後は、それを規律ある習慣へと変革する必要があります。



これまで

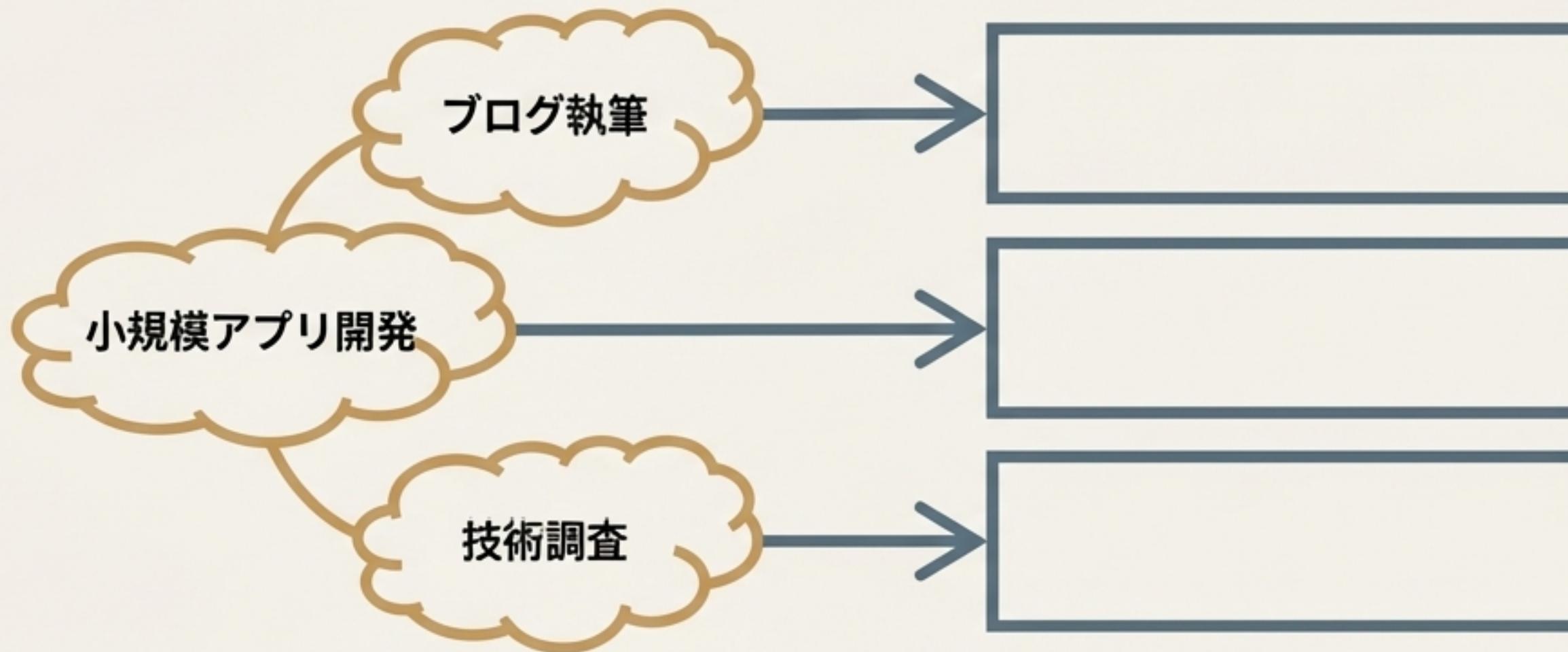
「やりたいこと」を、やりたい時に
アウトプットする



これから

「仕組み」としてのアウトプットをまず作り、
そこに「やりたいこと」を当てはめる

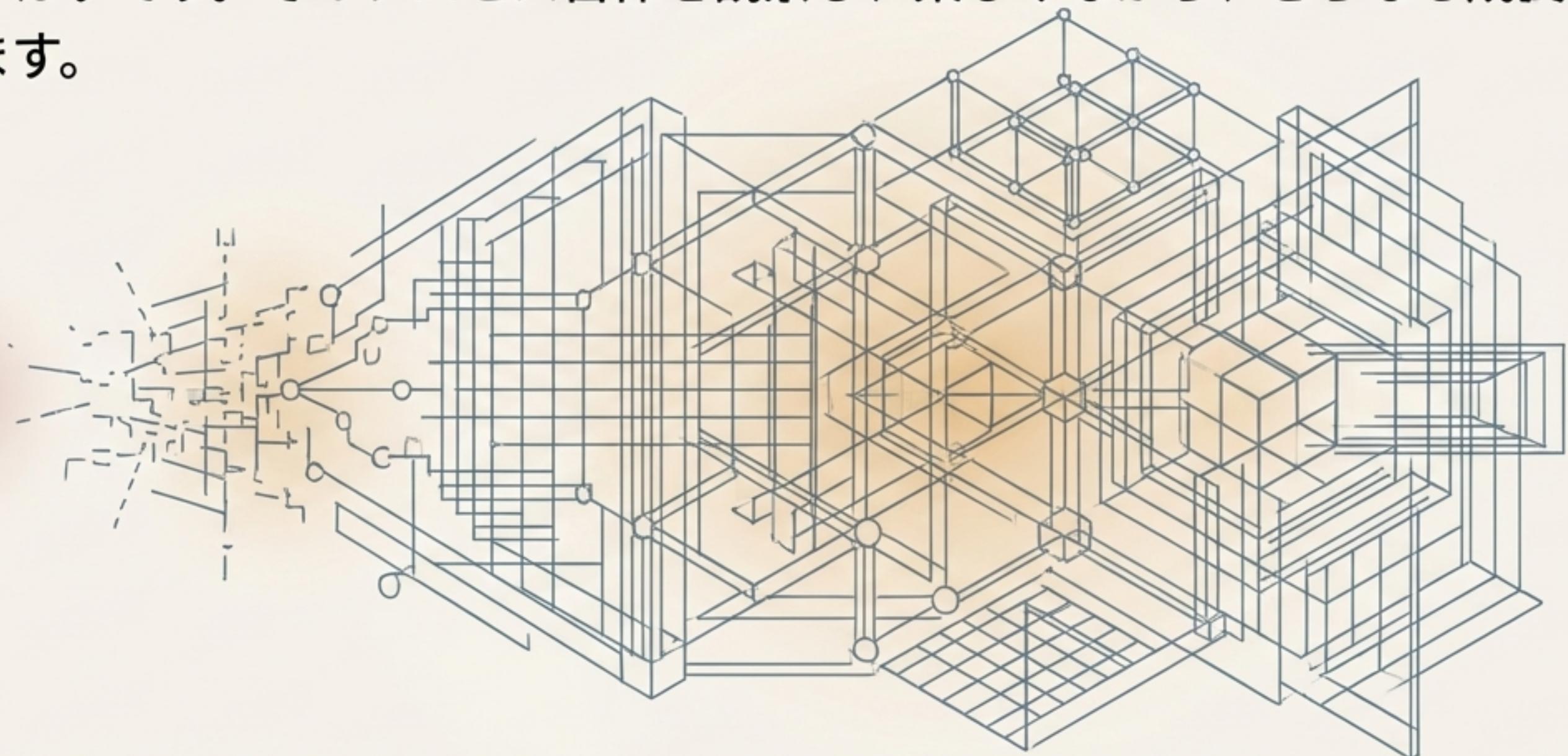
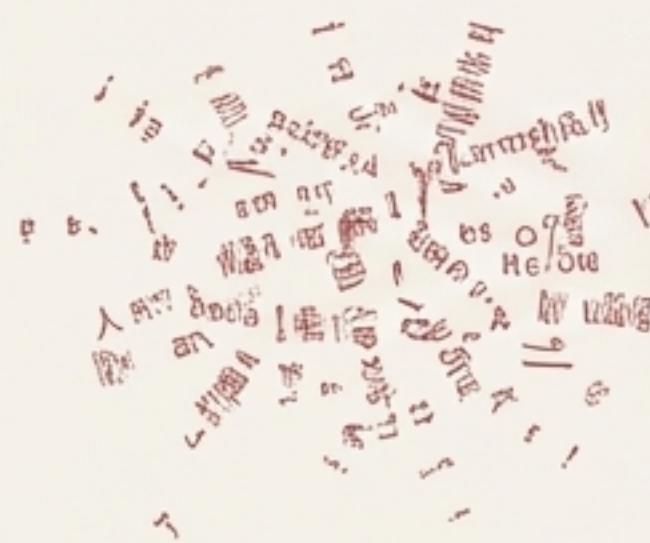
「やりたいこと」を「仕組み」に組み込む



具体的なアクションとして、まず「アウトプットする時間枠」を周期的に確保します。
そして、その決められた枠の中に、実行したい学習項目やプロジェクトを計画的に当てはめていく。
この仕組みにより、モチベーションの波に左右されず、継続的なアウトプットと成長を担保します。
「決められた周期で繰りかえされるアウトプットにやりたい項目を当てはめる形に移行すべき。」

これからの「見え方」の変化を楽しみに

コードが「構造物」に見えたように、学習を続けることで、これからも自分自身の物の見方や感じ方が変化していくはずです。そのプロセス自体を観察し、楽しみながら、さらなる成長成長を目指していきます。



「これから講義や課題を進めるにあたって、自分の物の見え方や感じ方が
どのように変化していくのか気になる所がありました」