今週の無料プレゼント

AI駆動開発 実践プロンプト チートシート

今すぐ使える実践プロンプト20本

RO合同会社 / ai.ros.co.jp

目次

1. 要件定義の明確化

曖昧なユーザー要求を具体的な機能要件、非機能要件、受け入れ条件に整理し、開発の方向性を定めます。

実践プロンプト

```
以下のユーザー要求を基に、機能要件、非機能要件、そして各機能に対する受け入れ条件をリストアップしてください。
ユーザー要求：「顧客が簡単に商品を検索・購入できるECサイトを作りたい。決済はスムーズに、サイトは速く表示してほしい。」
【追加で考慮すべき点や制約があれば入力】
```

期待される出力例

機能要件: 商品検索、商品一覧表示、カート追加、購入手続き、決済処理。 非機能要件: 応答速度2秒以内、セキュリティ対策、多言語対応。 受け入れ条件: 検索結果が3秒以内に表示されること。

Pro Tip: 具体的なユーザーペルソナやビジネス目標を提示すると、より的確な要件が導き出されます。

2. 設計書アウトライン

開発プロジェクトの設計書作成を効率化するため、システム構成や主要コンポーネントの骨子を生成します。

実践プロンプト

```
【開発するシステム名】の設計書アウトラインを作成してください。以下の項目を含めてください：
1. システム概要
2. システム構成図（主要コンポーネント）
3. データモデル概要
4. API設計概要
5. 技術スタック
6. セキュリティ考慮事項
7. 運用・保守
【システムの目的や規模を追記】
```

期待される出力例

システム名: 顧客管理システム 1. システム概要: 顧客情報の一元管理、営業活動支援。 2. システム構成: Webサーバー、DBサーバー、APIサーバー。 3. データモデル: 顧客、担当者、案件テーブル。 ...といった形式で設計書の章立てと概要が提示されます。

Pro Tip: まずはAIに大枠を作らせ、その後で詳細な情報を追記していくと効率的です。

3. 関数・メソッド実装

指定された機能要件に基づき、特定のプログラミング言語で関数やメソッドのコードスニペットを生成します。

実践プロンプト

```
Pythonで、与えられた文字列が回文であるかを判定する関数 `is_palindrome` を実装してください。
- 大文字・小文字を区別しない。
- スペースや記号は無視する。
- ドキュメンテーション文字列と型ヒントを含める。
【他の言語や具体的な要件があれば追記】
```

期待される出力例

```python def is_palindrome(text: str) -> bool: """ 与えられた文字列が回文であるかを判定します。 ... """ processed_text = "".join(char for char in text if char.isalnum()).lower() return processed_text == processed_text[::-1] ```

Pro Tip: 具体的な入出力例を提示すると、AIはより正確なコードを生成しやすくなります。

4. テストコード生成

既存の関数やクラスに対して、網羅性の高い単体テストコードを自動生成し、品質保証を支援します。

実践プロンプト

````
以下のPython関数に対するpytestのテストコードを生成してください。
```python
def calculate_discount(price: float, discount_rate: float) -> float:
    """
   商品価格と割引率に基づいて割引後の価格を計算します。
    Args:
        price: 商品の元の価格。
        discount_rate: 割引率（0.0から1.0の範囲）。
    Returns:
       割引後の価格。
    """
    if not (0.0 <= discount_rate <= 1.0):
        raise ValueError("Discount rate must be between 0.0 and 1.0.")
    return price * (1 - discount_rate)
```
【エラーケースやエッジケースを含めるよう指示】
````

期待される出力例

`test_calculate_discount_normal_case` `test_calculate_discount_zero_rate` `test_calculate_discount_full_rate` `test_calculate_discount_invalid_rate_low` `test_calculate_discount_invalid_rate_high` といったテスト関数を持つPythonファイル。

Pro Tip: テスト対象のコードに加えて、期待するテストケースのパターンを具体的に示すと効果的です。
````

5. バグ原因特定

エラーメッセージやコードスニペットから、潜在的なバグの原因を分析し、修正案を提示します。

実践プロンプト

````
以下のPythonのトレースバックと関連コードから、バグの原因を特定し、修正案を提案してください。
トレースバック:
```
Traceback (most recent call last):
  File "main.py", line 10, in <module>
    result = divide_numbers(10, 0)
  File "main.py", line 5, in divide_numbers
    return a / b
ZeroDivisionError: division by zero
```
関連コード:
```python
def divide_numbers(a, b):
    return a / b
```
【発生状況や期待する挙動を追記】
````

期待される出力例

エラー原因: `divide_numbers`関数でゼロ除算が発生している。 修正案: `b`がゼロの場合に`ZeroDivisionError`を捕捉するか、事前に`b`がゼロでないことをチェックする処理を追加する。 例: `if b == 0: raise ValueError("Cannot divide by zero")`

Pro Tip: エラーメッセージだけでなく、関連するコード全体と再現手順を詳しく提供すると、より正確な分析が得られます。

6. リファクタリング提案

既存のコード品質を向上させるため、可読性、保守性、パフォーマンスを改善する具体的なリファクタリング案を生成します。

実践プロンプト

以下のPythonコードをより読みやすく、保守しやすくリファクタリングしてください。特に、マジックナンバーの排除、関数の責務の分離、命名規則の改善に焦点を当ててください。

```python
def process_items(items):
    total = 0
    for item in items:
        if item['type'] == 'A':
            total += item['price'] * 1.1
        elif item['type'] == 'B':
            total += item['price'] * 1.05
        else:
            total += item['price']
    return total
```

【特定のリファクタリング手法を指示】

期待される出力例

定数を導入し、条件分岐を関数に分離した`process_items`関数の新しいバージョン。 例: `ITEM_TYPE_A_TAX_RATE = 1.1`、`_calculate_price_for_type_a`関数など。 変更点の説明と理由。

Pro Tip: リファクタリングの目的（例: テスト容易性向上、パフォーマンス改善）を明確にすると、AIはより的確な提案をします。

7. 技術ドキュメント生成

既存のコードベースや機能概要に基づき、開発者向けや利用者向けの技術ドキュメントのドラフトを生成します。

実践プロンプト

以下のAPIエンドポイントに関する技術ドキュメントを作成してください。Markdown形式で出力し、エンドポイントのURL、HTTPメソッド、リクエストパラメータ、レスポンス形式、エラーハンドリングを含めてください。
エンドポイント: `/users/{user_id}/orders`
目的: 特定ユーザーの注文履歴を取得
【具体的なリクエスト/レスポンス例を追記】

期待される出力例

Markdown形式で記述されたAPIドキュメント。 `# GET /users/{user_id}/orders` `## 概要` `特定のユーザーの注文履歴を取得します。` `## リクエスト` `### パスパラメータ` `- `user_id` (integer): ユーザーID` ...といった構成。

Pro Tip: ドキュメントの対象読者（開発者、運用担当者など）を指定すると、内容の深さと専門性が調整されます。

8. コードレビュー観点

プルリクエストの目的やコード変更内容から、コードレビューで確認すべき重要な観点をリストアップします。

実践プロンプト

```
以下のプルリクエストの説明文を読み、コードレビューで特に注意すべき観点を5つリストアップしてください。
プルリクエスト説明：「ユーザー登録フォームにメールアドレスのバリデーション機能を追加。正規表現で形式チェックを行い、重複チェックはDBで行う。」
【レビューアのスキルレベルやプロジェクトの規約を考慮させる】
```

期待される出力例

1. メールアドレスの正規表現が適切か（RFC準拠など）。 2. 重複チェック時のDBアクセス性能とトランザクション処理。 3. エラーメッセージがユーザーフレンドリーか。 4. セキュリティ（SQLインジェクションなど）対策は十分か。 5. テストコードがバリデーションロジックを網羅しているか。

Pro Tip: プロジェクトのコーディング規約やセキュリティガイドラインをAIに学習させると、より精度の高い観点が得られます。

9. CI/CDパイプライン

特定の技術スタックとデプロイ環境に基づき、CI/CDパイプラインの基本的な構成案とスクリプト例を生成します。

実践プロンプト

```
GitHub Actionsを使用して、PythonのWebアプリケーション（Flask）をAWS EC2にデプロイするCI/CDパイプラインの構成案をYAML形式で作成してください。
- テスト実行
- Dockerイメージのビルドと ECRへのプッシュ
- EC2インスタンスへのSSH接続とDockerコンテナの更新
【具体的なテストコマンドやデプロイ戦略を追記】
```

期待される出力例

deploy.ymlなどのファイル名で、GitHub ActionsのYAML定義。 `name: CI/CD Pipeline for Flask App` `on: push` `jobs:` ` build-and-deploy:` ` runs-on: ubuntu-latest` ` steps:` ` - uses: actions/checkout@v3` ` - name: Set up Python` ` uses: actions/setup-python@v4` ...といったステップを含むYAML。

Pro Tip: 使用するクラウドプロバイダーやデプロイツールを具体的に指定すると、より実践的なスクリプトが生成されます。

10. セキュリティチェック

コードスニペットに対して、OWASP Top 10などの一般的な脆弱性の観点から潜在的な問題点を指摘し、対策案を提示します。

実践プロンプト

以下のPHPコードにセキュリティ脆弱性がないかチェックし、もしあれば指摘と修正案を提示してください。

```php
<?php
$username = $_GET['username'];
$password = $_GET['password'];
$query = "SELECT * FROM users WHERE username = '$username' AND password = '$password'";
$result = mysqli_query($conn, $query);
?>
```

【想定される攻撃シナリオを提示】

期待される出力例

脆弱性: SQLインジェクションの脆弱性がある。`$username`と`$password`がエスケープされずにSQLクエリに直接埋め込まれているため、悪意のある入力によって任意のSQLが実行される可能性がある。 修正案: プリペアドステートメントを使用する。 例: `mysqli_prepare($conn, "SELECT * FROM users WHERE username = ? AND password = ?");`

Pro Tip: OWASP Top 10のどの項目に該当するか、といった具体的なフレームワーク名を指定すると、より専門的な分析が得られます。

11. DBスキーマ設計

アプリケーションの要件に基づき、関係データベースのテーブル構造（エンティティ、属性、リレーション）を設計します。

実践プロンプト

ECサイトのデータベーススキーマを設計してください。以下のエンティティを含めてください：
- ユーザー (User)
- 商品 (Product)
- 注文 (Order)
- 注文アイテム (OrderItem)

各テーブルに必要なカラム（データ型、制約）と、テーブル間のリレーションシップ（主キー、外部キー）を定義してください。
【特定のリレーションシップの指定や非機能要件を追記】

期待される出力例

`users`テーブル: `id (PK)`, `name`, `email (UNIQUE)`, `password_hash`, `created_at` `products`テーブル: `id (PK)`, `name`, `description`, `price`, `stock` `orders`テーブル: `id (PK)`, `user_id (FK)`, `order_date`, `total_amount`, `status` `order_items`テーブル: `id (PK)`, `order_id (FK)`, `product_id (FK)`, `quantity`, `price_at_order` といった形でテーブル定義とリレーションが示されます。

Pro Tip: エンティティ間の関係性（一対多、多対多など）を具体的に指示すると、より正確なスキーマが作成されます。

12. API仕様書作成

特定の機能を持つRESTful APIの仕様書を、OpenAPI (Swagger) 形式で生成し、開発者間の連携をスムーズにします。

実践プロンプト

```
ユーザー情報を取得するRESTful APIのエンドポイント `/api/v1/users/{id}` (GET) のOpenAPI 3.0仕様書（YAML形式）を作成してください。
- パスパラメータ `id` (integer) は必須。
- 成功時のレスポンス (200 OK) と、ユーザーが見つからない場合のエラーレスポンス (404 Not Found) を定義。
- レスポンスボディのスキーマも定義。
【認証方法や追加のエラーケースを追記】
```

期待される出力例

YAML形式のOpenAPI仕様書。```yaml openapi: 3.0.0 info: title: User API version: 1.0.0 paths: /api/v1/users/{id}: get: summary: Get user by ID parameters: - in: path name: id schema: type: integer required: true description: ID of the user to retrieve responses: '200': description: OK content: application/json: schema: type: object properties: id: type: integer name: type: string email: type: string '404': description: User not found content: application/json: schema: type: object properties: message: type: string ```

Pro Tip: APIのエンドポイント名、HTTPメソッド、期待される入出力例を具体的に提示すると、精度の高い仕様書が得られます。

13. CLIヘルプ生成

コマンドラインインターフェース（CLI）ツールの機能説明に基づき、ユーザー向けのヘルプメッセージや使用例を生成します。

実践プロンプト

```
Pythonで開発されたCLIツール `mycli` のヘルプメッセージを生成してください。以下のサブコマンドとオプションを含めてください。
- `mycli init`: プロジェクトを初期化する。オプション: `--template <name>` (テンプレート指定)
- `mycli build`: プロジェクトをビルドする。
- `mycli deploy`: プロジェクトをデプロイする。
【ツールの目的や詳細な機能を追記】
```

期待される出力例

``` Usage: mycli [OPTIONS] COMMAND [ARGS]... A command-line tool for managing your projects. Options: --help Show this message and exit. Commands: init Initialize a new project. build Build the project. deploy Deploy the project. ``` 各コマンドの詳細なヘルプも含む。

Pro Tip: コマンドの引数やオプションのデータ型、必須/任意を明確に指示すると、より正確なヘルプが生成されます。
```

14. UI提案

特定の機能やユーザー体験の要件に基づき、Webアプリケーションやモバイルアプリの UI要素やレイアウトのアイデアを提案します。

実践プロンプト

```
新規ユーザー登録フォームのUI要素とレイアウトを提案してください。
- 必須項目： メールアドレス、パスワード（確認用含む）、ユーザー名
- セキュリティ： パスワード強度表示、利用規約への同意チェックボックス
- ユーザー体験： エラーメッセージの表示位置、登録ボタンの配置
【ターゲットユーザー層やデザインガイドラインを追記】
```

期待される出力例

1. メールアドレス入力欄: プレースホルダー「メールアドレス」、タイプ「email」。 2. パスワード入力欄: プレースホルダー「パスワード」、タイプ「password」。入力中にパスワード強度バーを表示。 3. パスワード確認入力欄: プレースホルダー「パスワード（確認用）」。 4. ユーザー名入力欄: プレースホルダー「ユーザー名」。 5. 利用規約チェックボックス: 「利用規約に同意する」リンク付き。 6. 登録ボタン: フォーム下部に配置。 エラーメッセージは各入力欄の下に赤字で表示。

Pro Tip: ユーザーフローや具体的なユースケースを提示すると、AIはよりユーザー中心のUI提案を生成できます。

15. 正規表現作成

特定の文字列パターンにマッチする正規表現を生成し、入力検証やテキスト処理の効率化を支援します。

実践プロンプト

```
以下の条件を満たす文字列にマッチする正規表現をPythonの`re`モジュールで使用できるように作成してください。
- 半角英数字のみ。
- 8文字以上16文字以内。
- 少なくとも1つの大文字、1つの小文字、1つの数字を含む。
【マッチさせたい具体例とマッチさせたくない具体例を提示】
```

期待される出力例

```regex ^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z0-9]{8,16}$ ``` この正規表現は、指定されたパスワードポリシーに合致する文字列にマッチします。

Pro Tip: マッチさせたい文字列と、マッチさせたくない文字列の具体例を複数提示すると、より正確な正規表現が生成されます。
```

16. マイグレーションスクリプト

データベーススキーマの変更（テーブル追加、カラム変更など）を適用するためのマイグレーションスクリプトを生成します。

実践プロンプト

```
SQLAlchemy Alembicを使用して、既存の`users`テーブルに`phone_number`という新しいカラムを追加するPythonのマイグレーションスクリプトを作成してください。
- `phone_number`は文字列型（`String(20)`）。
- Nullable。
【デフォルト値やユニーク制約など、追加の制約を追記】
```

期待される出力例

```python """Add phone_number to users table""" from alembic import op import sqlalchemy as sa # revision identifiers, used by Alembic. revision = '...' down_revision = '...' branch_labels = None depends_on = None def upgrade(): op.add_column('users', sa.Column('phone_number', sa.String(length=20), nullable=True)) def downgrade(): op.drop_column('users', 'phone_number') ```

Pro Tip: データベースの種類（PostgreSQL, MySQLなど）やORM（SQLAlchemy, Django ORMなど）を明記すると、より適切なスクリプトが得られます。

17. エラーハンドリング

特定のプログラミング言語で、例外処理やエラーレスポンスを適切に実装するためのコードスニペットを生成します。

実践プロンプト

```
Pythonで、外部APIを呼び出す際に発生しうるエラーを適切にハンドリングするコードを実装してください。
- `requests.exceptions.ConnectionError`: ネットワーク接続の問題
- `requests.exceptions.Timeout`: タイムアウト
- HTTPステータスコード 4xx/5xx: APIからのエラーレスポンス
- それ以外の予期せぬエラー
各エラーに対して、適切なログ出力とユーザーへのフィードバックを想定した処理を含める。
【具体的なAPIエンドポイントや返り値の形式を追記】
```

期待される出力例

```python import requests import logging logging.basicConfig(level=logging.INFO) def fetch_data_from_api(url: str, timeout: int = 5) -> dict: try: response = requests.get(url, timeout=timeout) response.raise_for_status() # HTTP 4xx/5xx エラーを発生させる return response.json() except requests.exceptions.ConnectionError: logging.error("API Connection Error: Could not connect to the API.") raise ConnectionError("サービスに接続できませんでした。") except requests.exceptions.Timeout: logging.error("API Timeout Error: Request timed out.") raise TimeoutError("APIからの応答がありませんでした。") except requests.exceptions.RequestException as e: logging.error(f"API Request Error: {e}") raise ValueError(f"APIエラーが発生しました: {e}") except Exception as e: logging.error(f"An unexpected error occurred: {e}") raise RuntimeError("予期せぬエラーが発生しました。") ```

Pro Tip: エラーの種類ごとに、ユーザーへの表示メッセージと開発者向けのログ出力内容を区別して指示すると良いでしょう。

18. パフォーマンス改善

遅延しているコードやシステム構成に対し、具体的なボトルネックを特定し、パフォーマンスを改善するための提案を生成します。

実践プロンプト

以下のPythonコードのパフォーマンス改善案を提案してください。特に、ループ処理とデータ構造の選択に焦点を当ててください。
```python
def find_duplicates(data_list):
    duplicates = []
    for i in range(len(data_list)):
        for j in range(i + 1, len(data_list)):
            if data_list[i] == data_list[j] and data_list[i] not in duplicates:
                duplicates.append(data_list[i])
    return duplicates
```
【データ量や実行環境の制約を追記】

期待される出力例

現在のコードはO(N^2)の計算量を持つため、大規模データではパフォーマンスが低下する。 改善案: `set`データ構造を利用して重複を検出することで、計算量をO(N)に削減できる。 修正コード例: ```python def find_duplicates_optimized(data_list): seen = set() duplicates = set() for item in data_list: if item in seen: duplicates.add(item) else: seen.add(item) return list(duplicates) ```

Pro Tip: パフォーマンスプロファイリングの結果や、具体的な遅延が発生している箇所を明示すると、より的確な改善策が得られます。

19. 環境構築スクリプト

特定のOSやプロジェクトの依存関係に基づき、開発環境や本番環境をセットアップするためのスクリプトを生成します。

実践プロンプト

Ubuntu 22.04 LTS環境で、Python 3.10、PostgreSQL 14、Node.js 18をインストールし、基本的な開発ツール（git, vimなど）をセットアップするためのBashスクリプトを作成してください。
- 各ソフトウェアのバージョンを固定する。
- 冪等性を考慮する（既にインストール済みならスキップ）。
【プロジェクト固有の依存ライブラリや設定ファイルを追記】

期待される出力例

```bash #!/bin/bash set -euo pipefail # Update and upgrade system sudo apt update && sudo apt upgrade -y # Install basic development tools if ! command -v git &> /dev/null; then echo "Installing git..." sudo apt install -y git vim build-essential fi # Install Python 3.10 # ... (pyenv or specific apt commands) # Install PostgreSQL 14 # ... (apt commands for postgresql-14) # Install Node.js 18 # ... (nvm or specific apt commands) echo "Development environment setup complete!" ```

Pro Tip: 仮想化ツール（Docker, Vagrant）やプロビジョニングツール（Ansible, Chef）の使用を指示すると、より堅牢なスクリプトが生成されます。
```

20. ログフォーマット設計

アプリケーションのログ出力について、可読性、分析のしやすさ、検索性を考慮したフォーマットを設計します。

実践プロンプト

```
Pythonの`logging`モジュールを使用したJSON形式のログ出力フォーマットを設計してください。以下の情報を含めてください:
- タイムスタンプ (`asctime`)
- ログレベル (`levelname`)
- ロガー名 (`name`)
- メッセージ (`message`)
- ファイル名 (`filename`)
- 行番号 (`lineno`)
- 実行中の関数名 (`funcName`)
【追加で記録したい情報（ユーザーID、リクエストIDなど）を追記】
```

期待される出力例

```python import logging import json class JsonFormatter(logging.Formatter): def format(self, record): log_record = { "timestamp": self.formatTime(record, self.datefmt), "level": record.levelname, "logger": record.name, "message": record.getMessage(), "filename": record.filename, "lineno": record.lineno, "function": record.funcName, # "user_id": getattr(record, 'user_id', None), # 例: 追加情報 } return json.dumps(log_record) logger = logging.getLogger(__name__) logger.setLevel(logging.INFO) handler = logging.StreamHandler() handler.setFormatter(JsonFormatter()) logger.addHandler(handler) # 使用例 logger.info("ユーザーログイン成功", extra={'user_id': 123}) ```

Pro Tip: ログを収集・分析するツールの種類（ELK Stack, Datadogなど）を考慮すると、より最適なフォーマットが得られます。
```

もっと深く学びたい方へ

Premium加入で全レッスン受け放題

¥3,300/月

全レッスン読み放題・毎月MP自動付与

https://ai.ros.co.jp/learn/premium/

MPパックでレッスン単品購入

必要な分だけチャージして学習

https://ai.ros.co.jp/learn/mp-pack/

参考書籍：ゼロからはじめるAI駆動開発

Amazonで見る