ご購入・ご利用の前に必ずお読みください

- 本書記載の情報は、2021年10月現在のものを掲載しているため、ご利用時には変更されている可能性があります。OSやソフトウェアなどはバージョンアップされる場合があり、本書での説明とは機能内容や画面図などが異なってしまうこともありえます。本書ご購入の前にソフトウェアのバージョンをご確認ください。
- 本書の内容は、執筆時の最新バージョンである WordPress 5.8（以降WordPressと表記）を対象にしています。本文中の画像はWordPressとmacOS、Google Chrome（もしくはSafari）をもとにしています。
- 本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた運用は、必ずお客様自身の責任と判断によって行ってください。これらの情報の運用の結果、いかなる障害が発生しても、技術評論社および著者はいかなる責任も負いません。
- 本書で使用されている例はすべて架空のものであり、実在の人物・企業・団体とは一切関係がありません。

以上の注意事項をご承諾いただいた上で、本書をご利用願います。これらの注意事項に関わる理由に基づく、返金・返本を含むあらゆる対処を、技術評論社および著者は行いません。また本書の内容を超えた個別のご質問やトレーニングにあたるものについても対応できかねます。あらかじめ、ご承知おきください。

WordPressの名称およびロゴの商標は、非営利団体であるWordPressファウンデーションが保有・管理しています。その他、本書に掲載した会社名、プログラム名、システム名などは、米国およびその他の国における登録商標または商標です。なお、本文に™マーク、®マークは明記しておりません。

はじめに

あなたは、「オリジナルデザインでWordPressのテーマを作りたい」「HTMLやCSSで作った既存のウェブサイトをWordPress化したい」「WordPressのテーマを触ってみたい、改造してみたい」そんな風に思ってこの本を手にとられたことと思います。
とりあえず「外観」メニューから「テーマ」の中身を見てみるけれど、見慣れないタグがいっぱい…一体何をどうしたらいいんだろう…と途方に暮れる。10年前、駆け出しウェブデザイナーだった私がまさにそうでした。

本書「WordPressオリジナルテーマ制作入門」は、WordPressを管理画面から使うところから一歩進んで、HTMLとCSSを学習し、ひと通り身につけた方に向け、「WordPressのテーマを自分で作る」ための基礎知識をぎゅぎゅっと詰め込んだワークブックです。本書を読みながら一緒に手を動かすことで、テーマ制作に必要にして最小限の専門知識とワークフロー、そしてちょっとしたコツを理解できるようになっています。

特に以下の点に気をつけてこの本を書きました。

まずは、制作の土台となる基礎知識を丁寧に解説したことです。WordPressならではの「ループ」や「フック」といった専門用語、タグの使い方をはじめ、WordPressでよく使うPHPの基礎も盛り込みました。なるべく平易な表現を用いて、概念やイメージをつかんでいただくような工夫をしています。基礎知識を得て、仕組みを理解したうえでの制作は、今後起こるであろうトラブルやバージョンアップにも強いものとなります。プログラミングが苦手な方も大丈夫です、ゆっくりと1歩ずつ進めていきましょう。

あわせて、今回はクライアントワークを想定し実例に近いサンプルサイトを用意しました。本書の順を追って取り組むことで、オリジナルデザインの静的マークアップをWordPressのテーマにするまでの流れと、プロの仕事としても恥ずかしくない内容をききちんと把握することができます。

次に、ブロックエディターに対応した作り方をご紹介していることです。2018年にWordPress 5.0から導入されたブロックエディターは、今もなお革新的な進化を続けています。それにあたってテーマ上で対応しておきたい部分もしっかりと解説しました。今までのWordPressだけでなく、これからのWordPressにも対応してゆく力を身につけていきましょう。

最後に、より高度なテーマ制作への足がかりとなるヒントを散りばめたことです。本書はテーマ制作の入門に位置する本です。本書をひと通り学習し終わると、「ここはもっとこうしたい」といった気持ちが湧いてくると思います。そんな時、どのように調べていくか、どんな情報をよしとして取り入れていくのか、考え方のヒントになるコラムも書きました。本書が次のステップへの橋渡しになれば嬉しいです。

必要な部分を拾い読みしても構いませんが、ぜひ最初から最後まで、一緒に手を動かしてみてください。「だからこれがこうなるんだ！」と理解することで、本書のエッセンスを流用して、すぐにあなたのオリジナルテーマを作ることができるようになります。

ようこそテーマ制作の世界へ！

本書のワークを通して、あなたのWordPressを使ったウェブサイト制作がより楽しく充実していきますように。さぁ、はじめの一歩を一緒に踏み出しましょう。

著者を代表して　清水由規

本書の特徴と構成

本書は、HTMLとCSSをひと通り学習した方や、WordPressテーマのテンプレートタグやPHPをよりわかるようになりたい方が実際のクライアントワークを想定したサンプルサイトを順を追って作成することで、静的マークアップの状態からWordPressのテーマにするまでを学びます。
また、WordPress 5.0から導入されたブロックエディターに対し、テーマ側で対応しておきたい点も詳しく解説しています。

Chapter 1

WordPress の成り立ちや、WordPress の中でどのような動きを経てウェブページが表示されるのかといった、基本的な知識を解説します。

Chapter 2

テーマの作成に必要なファイル構成やテンプレートファイル、プログラミング言語の PHP、WordPress 特有の記述などを解説します。

Chapter 3

「Local」を使った開発環境の準備から、その使い方と WordPress の基本的な設定、プラグインの選び方とインストール方法を解説し、サンプルサイト作成に必要なコンテンツの準備を行います。

Chapter 4

WordPress のテーマとして必要最低限のファイルを作成し、テーマとして認識させるための準備を解説します。

Chapter 5

静的なコーディングを行った HTML ファイルを共通部分に分けたり、テンプレートタグに置き換えたりしながら、WordPress テーマの基本となるテンプレートファイルの作成を解説します。

Chapter 6

サイト内検索や、ページファイルが見つからなかった場合の 404 ページ、投稿間の移動ナビゲーションなど、ウェブサイト内の回遊性や利便性を高めるためのテンプレートや機能の追加を解説します。

Chapter 7

テーマ用のカラーパレットの設定から、独自のカスタムスタイルの作り方、独自のブロックパターンの作成方法まで、テーマを作成する上で必要となるブロックエディターへの対応方法について説明します。

Chapter 8

カスタム投稿タイプを使った独自の投稿作成やカスタムテンプレート、ブロックエディターによるフロントページの編集や、投稿一覧ページの作成について解説します。

Contents

Chapter 1 WordPressとは？

Step 01 WordPressはCMSの1つ …… 14
Step 02 なぜWordPressを選ぶのか？ …… 17
Step 03 静的ウェブサイトとWordPressで作成したウェブサイトの違い …… 21
Column WordPressのライセンス …… 25

Chapter 2 テーマ作成に必要な基礎知識

Step 01 WordPressのテーマとは …… 28
Step 02 テーマのフォルダー構成 …… 30
Step 03 テンプレートファイルとは …… 32
Step 04 PHPのしくみと書き方を知る …… 40
Step 05 WordPressにおけるタグとは …… 51
Step 06 WordPressのループとは …… 53
Step 07 フックとは …… 58
Column テーマ開発者のためのハンドブック …… 60

Chapter 3 オリジナルテーマの開発環境を準備する

Step 01 Localをインストールして基本的な使い方を知る …… 62
Step 02 WordPressの初期設定を行う …… 67
Step 03 プラグインの選定とインストール …… 71
Step 04 サンプルページを作成する …… 74

Chapter 4 最低限のテーマを作成しWordPressに認識させる

Step 01 テーマフォルダーを作成する …… 96
Step 02 テーマに必要なファイルを作る …… 98
Step 03 テーマの表示を整える …… 102
Step 04 画像・CSS・JavaScriptを準備する …… 106

Chapter 5 ウェブサイト作成の基本となるテンプレートファイルを作成する

Step 01 ウェブサイトの共通部分を作成する …… 110
Step 02 固定ページ用のテンプレートファイルを作成する …… 138
Step 03 投稿ページ用のテンプレートファイルを作成する …… 160
Step 04 アーカイブページ用のテンプレートファイルを作成する …… 176
Step 05 ナビゲーションを作成する …… 207
Step 06 フロントページ用のテンプレートを作成する …… 219
Column テーマ作成のセキュリティ対策について …… 233

Chapter 6 ウェブサイトの利便性を向上する

Step 01 サイト内検索をできるようにする …… 236
Step 02 404ページを作成する …… 252
Step 03 パンくずリストを作成する …… 256
Step 04 前後の投稿へのリンクを追加する …… 265
Step 05 ウィジェットエリアを追加する …… 271
Column コメントを受け付けたいと思ったら …… 287

Chapter 7 テーマをブロックエディターに対応させる

Step 01 テーマをブロックエディターに対応させる …… 290
Step 02 コアブロックのスタイルを独自に追加する …… 306
Step 03 ブロックパターンを削除・追加する …… 314
Column ブロックエディターとGutenbergプロジェクト …… 322
Column カスタムフィールドは本来の使い方へ …… 324

Chapter 8 ウェブサイトの機能を拡張する

Step 01 カスタム投稿タイプを追加してコンテンツを拡張する …… 326
Step 02 カスタム投稿タイプに独自の分類を追加する …… 344
Step 03 カスタム投稿タイプに関連する部分の表示を整える …… 357
Step 04 カスタムページテンプレートで個別にデザインを変更する …… 368
Step 05 投稿インデックスページを作成する …… 374
Step 06 フロントページをブロックエディター化する その１～固定ページの指定とブロックの配置 …… 383
Step 07 フロントページをブロックエディター化する その２～ショートコードブロックの活用 …… 400
Column WordPressへの理解をより深めるために …… 411

Step

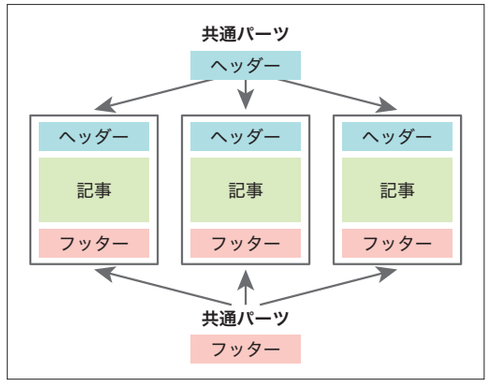
01 ウェブサイトの共通部分を作成する

■Step素材フォルダー	kuroneko_sample ＞ Chapter5 ＞ Step1
■学習するテーマファイル	•index.php •header.php •footer.php •functions.php

ヘッダー部分やフッター部分を独立したファイルにする

ウェブページは、大まかにヘッダー部分、記事部分、フッター部分から構成されています。そのうちヘッダー部分とフッター部分は、ウェブサイト全体の共通部分でもあります。これから作成する投稿ページや固定ページのテンプレートファイルには、共通部分のコードを直接記述することもできます。

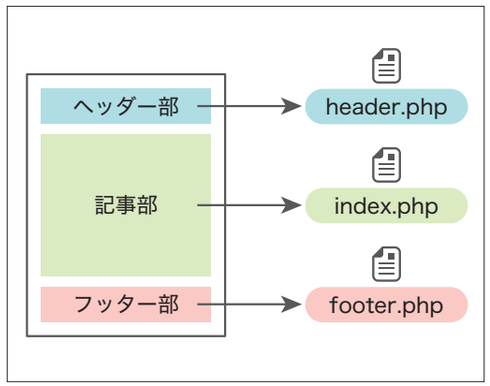
しかし共通部分に変更があった場合、すべてのテンプレートファイルに修正を反映させる作業が必要になってしまいます。そこでヘッダー部分やフッター部分を独立したファイルにすることで、共通部分のファイルを修正するだけでウェブサイト全体に変更が反映されるようにします。



共通パーツ化のイメージ

WordPressでは、ヘッダー部分とフッター部分をそれぞれ「header.php」「footer.php」というテンプレートファイルとして作成し、決められたテンプレートタグを記述して共通部分を読み込みます。そこで、最初に静的コーディングしたHTMLファイルをもとにヘッダー部分とフッター部分のテンプレートファイルを作成し、ウェブページに読み込んで出力してみましょう。

ブラウザで表示したヘッダー部とフッター部



ファイルを分割するイメージ

このChapterの後半では、外部から読み込んでいるCSSファイルやJavaScriptファイルもWordPressのしくみを使って読み込みます。ウェブサイトの基本となる箇所なのでステップが少し多いですが、1つずつ落ち着いて進めば大丈夫です。それでは始めましょう！

Step 1-1 サイトの共通部分を分割して読み込む

あらかじめ用意されたHTMLファイルからサイトの共通部分を分割し、それぞれをPHPファイルとして作成していきましょう。

❶ヘッダー部分を分割する

最初に、ヘッダー部分にあたるテンプレートファイル「header.php」を作成します。「kuroneko-hair」フォルダー内にheader.phpという名前の空ファイルを作成し、テキストエディターで開きましょう。

次に、P.99で作成した「kuroneko-hair」フォルダー内のindex.phpを開きます。index.phpの1行目<!DOCTYPE html>から</header>までを選択してカットし、header.phpにペーストします。ペーストできたら、index.phpとheader.phpを保存します。

header.php

```
<!DOCTYPE html>  ← index.phpからカット＆ペーストする
<html lang="ja">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="description" content="ゆったり時間と癒しの美容室サンプルサイト">
    <title>Kuroneko Hair</title>
    <link href="https://fonts.googleapis.com/css2?family=Noto+Sans+JP:wght@500&disp
    lay=swap" rel="stylesheet">
    <link href="./assets/css/theme-styles.css" rel="stylesheet" media="all">
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
</head>

<body>
    <div class="content-Wrap">
        <header role="banner" class="header">
            <h1 class="header-SiteName">
                <a href="/" class="header-SiteName_Link">
                    <img src="./assets/img/logo.png" alt="Kuroneko Hair">
                </a>
```

```
            <span class="header-Tagline">ゆったり時間と癒しの美容室サンプルサイト
            </span>
        </h1>
        <nav class="header-Nav">
            中略
        </nav>
    </header>
```

❷フッター部分を分割する

header.phpの次は、フッター部分にあたるテンプレートファイル「footer.php」を作成します。「kuroneko-hair」フォルダー内にfooter.phpという名前の空ファイルを作成し、テキストエディターで開きましょう。

続いて、「kuroneko-hair」フォルダー内のindex.phpを開きます。index.phpの<footer role="contentinfo" class="footer">から</html>までを選択してカットし、footer.phpにペーストします。ペーストできたら、index.phpとfooter.phpを保存します。

footer.php

```
            <footer role="contentinfo" class="footer">
        <div class="footer-Widgets">
            中略
            <p class="footer-Copyright">
                <small>&copy; 2021 Kuroneko Hair Sample </small>
            </p>
        </footer>
    </div>
    <script src="./assets/js/theme-common.js"></script>
</body>
</html>
```

index.phpからカット＆ペーストする

❸テーマ共通のテンプレートファイルを読み込む

テーマ共通のテンプレートファイルとして分割したheader.phpとfooter.phpを、index.phpのHTMLをカットした部分に読み込んでみましょう。header.phpの読み込みにはテンプレートタグ「get_header()」、footer.phpの読み込みにはテンプレートタグ「get_footer()」を使います。

index.phpのヘッダー部分をカットした箇所に<?php get_header(); ?>を、フッターをカットした箇所に<?php get_footer(); ?>を記述します。記述できたら、index.phpを保存します。

index.php

```
<main class="main">
    <div class="container-fluid">
        <div class="home-Hero">
後略
```

↓

```
<?php get_header(); ?>
<main class="main">
    <div class="container-fluid">
        <div class="home-Hero">
後略
```

テンプレートタグを追加する

index.php

```
前略
        </section>
    </div>
</main>
```

↓

```
前略
        </section>
    </div>
</main>
<?php get_footer(); ?>
```

テンプレートタグを追加する

テンプレートタグ get_header(ファイル名)

テーマフォルダーにあるheader.phpを読み込む。

引数

ファイル名（任意）：ファイルの名前（sample）を指定すると、指定したヘッダーファイル（header-sample.php）を読み込む。

テンプレートタグ get_footer(ファイル名)

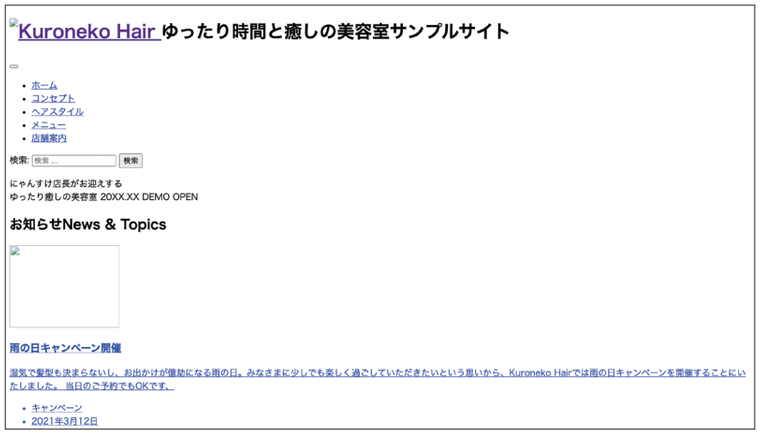
テーマフォルダーにあるfooter.phpを読み込む。

引数

ファイル名（任意）：ファイルの名前（sample）を指定すると、指定したフッターファイル（footer-sample.php）を読み込む。

Step 1-2 画像と外部ファイルのパスを修正する

ここまでで、ヘッダー部分とフッター部分をテーマ共通のテンプレートファイルとして作成し、index.phpに読み込むことができました。この状態で、一度ウェブサイトを表示してみましょう。画像が読み込まれず、CSSが適用されていないことが確認できます。



画像が表示されず、CSSが適用されていない

現時点でサンプルサイトの画像やCSS、JavaScriptのファイルは、「./assets/css/theme-styles.css」のように相対パスで参照していますが、実際にはテーマフォルダー（/wp-content/themes/kuroneko-hair/）内のファイルを参照する必要があります。そこで、有効になっているテーマフォルダーへのパスを取得する関数「get_template_directory_uri()」を使って、ファイルのパスを修正しましょう。

例えばテーマのCSSファイルの場合、この関数を記述するとテーマフォルダーのURI（例：https://example.com/wp-content/themes/kuroneko-hair）を取得し、次のように出力されます。

get_template_directory_uri() 追加前のパス

```
<link href="./assets/css/theme-styles.css" rel="stylesheet" media="all">
```

get_template_directory_uri() 追加後のパス

```
<link href="https://example.com/wp-content/themes/kuroneko-hair/assets/css/theme-styles.css" rel="stylesheet" media="all">
```

それでは、header.phpとfooter.phpをテキストエディターで開き、CSS、JavaScript、画像の相対パスをget_template_directory_uri()関数を使って置き換えましょう。有効になっているテーマフォルダーまでのパスをget_template_directory_uri()関数を使って出力するには、P.44で解説した「echo」と「esc_url()」を組み合わせて次のように記述します。

```
<?php echo esc_url( get_template_directory_uri() ); ?>
```

この時、次のように「esc_url()」がない状態でもURIは出力されますが、セキュリティ上のリスクを減らすため、ここでは「esc_url()」を使用します。詳しくはP.233を参照してください。

```
<?php echo get_template_directory_uri(); ?>
```

header.php

```
<!DOCTYPE html>
<html lang="ja">

<head>
    中略
    <link href="https://fonts.googleapis.com/css2?family=Noto+Sans+JP:wght@500&disp
    lay=swap" rel="stylesheet">
    <link href="./assets/css/theme-styles.css" rel="stylesheet" media="all">
    <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
</head>

<body>
    <div class="content-Wrap">
        <header role="banner" class="header">
            <h1 class="header-SiteName">
                <a href="/" class="header-SiteName_Link">
                    <img src="./assets/img/logo.png" alt="Kuroneko Hair">
                </a>
                <span class="header-Tagline">ゆったり時間と癒しの美容室サンプルサイト
                </span>
            </h1>
後略
```

↓

```
<!DOCTYPE html>
<html lang="ja">

<head>
    中略
    <link href="<?php echo esc_url( get_template_directory_uri() ); ?>/assets/css/
```

CSSファイルの相対パスを関数に置き換える

```
    theme-styles.css" rel="stylesheet" media="all">
    <script src="https://code.jquery.com/jquery-3.3.5.min.js"></script>
</head>

<body>
    <div class="content-Wrap">
        <header role="banner" class="header">
            <h1 class="header-SiteName">
                <a href="/" class="header-SiteName_Link">
                    <img src="<?php echo esc_url( get_template_directory_uri() );
                    ?>/assets/img/logo.png" alt="Kuroneko Hair">
                </a>
                <span class="header-Tagline">ゆったり時間と癒しの美容室サンプルサイト
                </span>
            </h1>
後略
```

画像ファイルの相対パスを関数に置き換える

footer.php

```
前略
        </footer>
    </div>
    <script src="./assets/js/theme-common.js"></script>
</body>

</html>
```

↓

```
        </footer>
    </div>
    <script src="<?php echo esc_url( get_template_directory_uri() ); ?>/assets/js/
    theme-common.js"></script>
</body>

</html>
```

JavaScriptファイルの相対パスを関数に置き換える

index.php内の画像ファイルへのパスも、すべて同じように修正しましょう。

index.php

```
前略
<img src="./assets/img/rainy-day-thumb.jpg" alt="" width="200" height="150"
load="lazy">
後略
```

↓

```
前略
<img src="<?php echo esc_url( get_template_directory_uri() ); ?>/assets/img/rainy-
day-thumb.jpg" alt="" width="200" height="150" load="lazy">
後略
```

画像ファイルの相対パスを関数に置き換える

画像が読み込まれ、CSSが適用された

ここで、フロントページを再読み込みしてみましょう。CSS、JavaScript、画像の各ファイルが正しく読み込まれ、表示が静的コーディングされたHTMLと同じ状態になりました。

関数 get_template_directory_uri()

現在有効になっているテーマディレクトリーのURIを取得する。子テーマを使用している場合は、親テーマのテーマディレクトリーのURIを取得する。子テーマのディレクトリーを取得するには、get_stylesheet_directory_uri() を使用する。

Step 1-3 header.php を作り込む

この時点で、header.phpはテーマ共通のテンプレートファイルとして利用できるようになっています。しかし、このままではWordPressで設定した言語属性やウェブサイトのタイトル、キャッチフレーズなどの出力ができません。そのため、該当箇所を関数やテンプレートタグで置き換えていきます。

❶サイトの言語属性を出力する

HTMLでは、その文書で使われる言語の種類がhtmlタグのlang属性に記述されます。header.phpで、lang属性は「language_attributes()」関数で置き換えます。ここには、P.67で設定した「サイトの言語」が反映されます。サンプルサイトでは「日本語」を選択したので、「lang="ja"」と出力されます。

header.php

```
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
後略
```

↓

```
<!DOCTYPE html>
<html <?php language_attributes(); ?>>  ← 関数に置き換える
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
後略
```

関数 language_attributes(ドキュメントの種類)

WordPress「サイトの言語」で設定した言語属性をhtmlタグに出力する。

引数

ドキュメントの種類（任意）：「html」または「xhtml」を指定することで、ドキュメントの種類を変更できる。

❷文字コードを出力する

metaタグで記述している文書の文字コードを、テンプレートタグ「bloginfo()」で置き換えます。このテンプレートタグには複数のオプションがありますが、ここではカッコの中に「'charset'」を記述し、<?php bloginfo('charset'); ?>とします。WordPressで使用される文字コードは「UTF-8」なので、テンプレートタグへの置き換え後は変更前と同じく「charset="UTF-8"」と出力されます。

bloginfo()タグは、この後の「キャッチフレーズ」や「サイトのタイトル」の出力にも登場するので覚えておきましょう。

header.php

```
<!DOCTYPE html>
<html <?php language_attributes(); ?>>
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
後略
```

↓

```
<!DOCTYPE html>
<html <?php language_attributes(); ?>>
<head>
    <meta charset="<?php bloginfo( 'charset' ); ?>">  ← テンプレートタグに置き換える
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
後略
```

テンプレートタグ bloginfo(表示する情報名)

WordPressの情報を出力する。

引数

表示する情報名：追加する機能名を指定する。

主な機能名

charset　：ウェブサイトの文字コードを出力する。
description：一般設定「キャッチフレーズ」を出力する。
name　：一般設定「サイトのタイトル」を出力する。

❸キャッチフレーズを出力する

P.68で設定した「キャッチフレーズ」のテキストを、メタディスクリプションとして出力します。ここでもテンプレートタグ「bloginfo()」を使用します。カッコの中に「'description'」を指定し、<?php bloginfo('description'); ?>と記述します。

header.php

```
前略
<meta name="description" content="ゆったり時間と癒しの美容室サンプルサイト">
後略
```

↓

```
前略
<meta name="description" content="<?php bloginfo( 'description' ); ?>">
後略
```

テンプレートタグに置き換える

❹ロゴ画像のリンクと代替テキストを置き換える

ヘッダーのサイトロゴ部分には、ホームへのリンクが設定されたロゴ画像と、ウェブサイトの内容を端的に示すテキストであるタグラインが表示されています。Step 1-2でロゴ画像のパスは修正したので、ここではロゴ画像のリンク、ロゴ画像の代替テキスト、タグラインを変更します。

ロゴ画像のリンクは、テンプレートタグ「home_url()」でWordPressの一般設定「サイトアドレス(URL)」の内容を出力します。このテンプレートタグは、「echo」「esc_url()」と組み合わせて<?php echo esc_url(home_url()); ?>と記述します。

ロゴ画像の代替テキストは、テンプレートタグ「bloginfo('name')」でWordPressの一般設定「サイトのタイトル」を出力します。

WordPressの一般設定

タグラインは、メタディスクリプションと同じ「キャッチフレーズ」の設定を出力したいので、テンプレートタグ「bloginfo('description')」に置き換えます。

header.php

```
前略
<h1 class="header-SiteName">
    <a href="/" class="header-SiteName_Link">
        <img src="<?php echo esc_url( get_template_directory_uri() ); ?>/assets/img/
        logo.png" alt="Kuroneko Hair">
        <span class="header-Tagline">ゆったり時間と癒しの美容室サンプルサイト</span>
    </a>
</h1>
後略
```

```
前略
<h1 class="header-SiteName">
    <a href="<?php echo esc_url( home_url() ); ?>" class="header-SiteName_Link">
        <img src="<?php echo esc_url( get_template_directory_uri() ); ?>/assets/img/
        logo.png" alt="<?php bloginfo( 'name' ); ?>">
    </a>
    <span class="header-Tagline"><?php bloginfo( 'description' ); ?></span>
</h1>
後略
```

テンプレートタグに置き換える

テンプレートタグに置き換える

テンプレートタグに置き換える

テンプレートタグ　home_url(パス, URLスキーム)

WordPressの一般設定「サイトアドレス (URL)」の内容を取得する。末尾のスラッシュは省略される。

引数

パス (任意)　：ホームURLからの相対パス「'/'」や「'pagename'」などを記述すると、次のようなURLを出力する。
home_url('/') → https://example.com/
home_url('pagename') → https://example.com/pagename

URLスキーム (任意)：ホームURLに使うスキーム。http、https、相対パスが指定できる。

❺bodyタグのクラスを動的に出力する

固定ページなどでページごとにデザインを変えたい場合、bodyタグに固有のクラス名をつけておくと、CSSを記述するときに便利です。WordPressではテンプレートタグ「body_class()」を使って、そのページの種類(投稿、固定ページ…など)を表す文字列をbodyタグのclass属性として出力できます。

例えばフロントページでは「home」、投稿ページでは「single」、固定ページでは「page-id-123」といったクラス名が出力されます。

header.php

```
前略
<body>
後略
```

```
前略
<body <?php body_class(); ?>>
後略
```

テンプレートタグを追加する

Step

01 テーマをブロックエディターに対応させる

■Step素材フォルダー　kuroneko_sample ＞ Chapter7 ＞ Step1
■学習するテーマファイル　●functions.php

テーマのブロックエディター対応

テーマをブロックエディターに対応させるには、最初にコアブロック（P.76）のスタイルをテーマに対応させる必要があります。コアブロックには、あらかじめ最低限のスタイルが用意されています。しかし、いくつかのスタイルはテーマに合わせて変更したり、独自のスタイルを追加したりする必要があるでしょう。ここでは、各ブロックに共通のカラーパレットや文字サイズの設定（Step 1）、ボタンブロックのスタイル（Step 2）などをテーマのデザインに合わせて変更していきます。

テーマの用途によっては必要のないブロックや機能もあるので、必ずしもすべてのブロックをテーマに対応させる必要はありません。しかしその選択をするためには、それぞれのブロックの特性をしっかりと理解しておくことが大切です。そのためにも、まずはブロックエディター自体の操作に慣れるようにしてください。

WordPressには、テーマをブロックエディターに対応させるためのいくつかの設定が用意されています。これらの設定は、すべてのテーマで同様の記述が必要というわけではありません。実際の制作時には、どの設定をどのように変更するのか、必要に応じて判断してください。このStepではサンプルテーマでの記述例を紹介しながら、それらの設定を１つ１つ解説していきます。

Step 1-1 ブロックエディター用CSSをテーマに読み込ませる

WordPress 5.8には、80個のコアブロックが含まれています（P.76）。これらブロックには、骨子となる最低限のスタイルがエディター側および公開サイト側両方にあらかじめ読み込まれています。それに加えて、テーマ側で利用するかどうかを選択できるスタイルが用意されています。例えば、引用ブロックの左側についている線などがそれにあたります。作成したテーマにそれらのスタイルを読み込ませる設定が「add_theme_support('wp-block-styles')」です。add_theme_support() 関数については、P.125で詳しく解説しています。

functions.phpに次の記述を追加すると、これらのスタイルが読み込まれるようになります。

functions.php

```
前略
add_action( 'widgets_init', 'neko_widgets_init' );
```

← 最終行に続けて記述する

↓

```
前略
add_action( 'widgets_init', 'neko_widgets_init' );

function neko_block_setup() {
  add_theme_support( 'wp-block-styles' );
}
add_action( 'after_setup_theme', 'neko_block_setup' );
```

← ブロック用CSSを有効化する

← after_setup_themeフックに関数を追加する

この記述を追加せずに、自分で書いたCSSによって個別に対応させることもできます。しかしブロックの数も多く作業が大変になるので、まずは上記の記述でブロック用のスタイルを読み込ませておき、ブロックエディターへの対応に慣れてきたら読み込ませないという選択を考えるとよいかもしれません。実際に配布されている有料テーマのいくつかでは、読み込ませていないケースも見られます。

関数 add_theme_support('wp-block-styles')

ブロック用のCSSを有効化する。

Step 1-2 埋め込みブロックをレスポンシブに対応させる

ブロックエディターには、たくさんの埋め込みブロックがあります。例えばTwitterのURLを入力すると、自動判別して該当する内容を埋め込んでくれるようなブロックです。これらのブロックにはiframeが利用されています。iframeで指定された内容の縦横比を保持したままレスポンシブ表示に対応させるためには、bodyタグに「wp-embed-responsive」というクラスが必要です。しかしそのままでは付与されないので、「add_theme_support('responsive-embeds'); 」の記述を追加します。

functions.phpに、次の記述を追加します。

functions.php

```
function neko_block_setup() {
  add_theme_support( 'wp-block-styles' );
  ←ここに記述する
}
add_action( 'after_setup_theme', 'neko_block_setup' );
```

↓

```
function neko_block_setup() {
  add_theme_support( 'wp-block-styles' );
  add_theme_support( 'responsive-embeds' );  ←埋め込み要素のレスポンシブスタイルを適用する
}
add_action( 'after_setup_theme', 'neko_block_setup' );
```

表示を確認すると、bodyタグに該当のクラス名が追加されています。埋め込みブロックに対して特別な対応がない限りは、記述しておきましょう。なお、埋め込みブロックを一切用いないテーマの場合、この記述は必要ありません。

bodyタグにwp-embed-responsiveクラスが追加されている

関数 add_theme_support('responsive-embeds')

埋め込みブロックによる埋め込み要素に、レスポンシブのスタイルを適用させる。

Step 1-3 幅広のスタイルに対応させる

ブロックエディターでは、コンテンツの幅として「標準」「幅広」「全幅」の３種類のスタイルが用意されています。

ツールバー上で選択できる「幅広」と「全幅」のコンテンツ幅

テーマ側で何も設定しないと、利用できるのは「標準」の幅のみになります。「幅広」「全幅」にも対応させたい場合は、functions.phpに「add_theme_support('align-wide')」を記述します。

functions.phpに、次の記述を追加します。

functions.php

```
function neko_block_setup() {
  add_theme_support( 'wp-block-styles' );
  add_theme_support( 'responsive-embeds' );
  ←ここに記述する
}
add_action( 'after_setup_theme', 'neko_block_setup' );
```

↓

```
function neko_block_setup() {
  add_theme_support( 'wp-block-styles' );
  add_theme_support( 'responsive-embeds' );
  add_theme_support( 'align-wide' );  ←「幅広」「全幅」のスタイルに対応させる
}
add_action( 'after_setup_theme', 'neko_block_setup' );
```

これで、ブロックエディター側で「幅広」「全幅」が選択できるようになります。「全幅」スタイルを使ったブロックの使い方は、P.392で紹介しています。

関数 add_theme_support('align-wide')

ブロックエディターで「幅広」「全幅」を選択できるようにする。

Step 1-4 カラーパレットを設定する

ブロックエディターには、カラーパレットの機能が用意されています。例えばカバーブロックや段落ブロックなどで設定サイドバーにカラーパレットを表示し、文字や背景の色を指定することができます。

カバーブロックのカラーパレット

テーマでは「add_theme_support('editor-color-palette')」を使って、カラーパレットの色を自由に指定することができます。テーマに合わせた色を指定しておくことで、どの色を選んでもテーマの雰囲気から逸脱しないデザインにすることができます。

add_theme_support() では、() 内の第２引数に連想配列 (P.47) を指定して各色の設定を読み込ませます。例えば「アクセントカラー１」という名前で「#3E8CB8」の色を設定したい場合、次のような３つのキーと値の組み合わせを記述します。slugは、自動生成されるCSSのクラス名で使用される文字列です。

１色分の表記の例

```
add_theme_support(
  'editor-color-palette',
  array(
    'name' => 'アクセントカラー',
    'slug' => 'accent-color',
    'color' => '#3D8CB8'
  ),
);
```

必要な色の数だけこのセットをつなげて、パレットに色を指定していきます。

これで、該当のパレットの色を選択するとslug名の前に「has-」、後に「background-color」(背景色の場合) または「color」(文字色の場合) というクラス名が自動生成され、そのブロックに付与されます。そのため、生成されるクラス名に対応した記述がCSS側に必要になります。例えば上記の例の場合は、次のような記述がCSS側 (サンプルテーマではtheme-styles.cssとeditor-styles.cssの２つのファイルが該当します) で必要になります。

１色分の表記の例に対するCSSの記述例

```
.has-accent-color-background-color {
  background-color: #3D8CB8;
}
.has-accent-color-color {
  color: #3D8CB8;
}
```

それでは、サンプルテーマ独自のカラーパレットを設定していきましょう。

❶functions.phpに記述する

functions.phpに、次の記述を追加します。サンプルテーマでは、テーマに使用している色を基準に５色を指定します。

functions.php

```
function neko_block_setup() {
  add_theme_support( 'wp-block-styles' );
  add_theme_support( 'responsive-embeds' );
  add_theme_support( 'align-wide' );
    ← ここに記述する
}
add_action( 'after_setup_theme', 'neko_block_setup' );
```

```
function neko_block_setup() {
  add_theme_support( 'wp-block-styles' );
  add_theme_support( 'responsive-embeds' );
  add_theme_support( 'align-wide' );
  add_theme_support(                    ← カラーパレットの設定
    'editor-color-palette',
    array(
      array(
        'name' => 'スカイブルー',
        'slug' => 'skyblue',
        'color' => '#00A1C6',
      ),
      array(
        'name' => 'ライトスカイブルー',
        'slug' => 'light-skyblue',
        'color' => '#ECF5F7',
      ),
      array(
        'name' => 'ライトグレー',
        'slug' => 'light-gray',
        'color' => '#F7F6F5',
      ),
      array(
        'name' => 'グレー',
        'slug' => 'gray',
        'color' => '#767268',
      ),
      array(
        'name' => 'ダークグレー',
        'slug' => 'dark-gray',
        'color' => '#43413B',
      ),
    )
  );
}
add_action( 'after_setup_theme', 'neko_block_setup' );
```

❷CSSに色のスタイルを記述する

上記で設定した値の通りに、CSSにスタイルを記述します。 サンプルテーマでは、次のように該当のスタイルがあらかじめ用意されていますので、実際の作業は必要ありません。

theme-styles.css

```
.has-skyblue-background-color {
  background-color: #00A1C6;
}
.has-skyblue-color {
  color: #00A1C6;
}
.has-light-skyblue-background-color {
  background-color: #ECF5F7;
}
.has-light-skyblue-color {
  color: #ECF5F7;
}
.has-light-gray-background-color {
  background-color: #F7F6F5;
}
.has-light-gray-color {
  color: #F7F6F5;
}
.has-gray-background-color {
  background-color: #767268;
}
.has-gray-color {
  color: #767268;
}
.has-dark-gray-background-color {
  background-color: #43413B;
}
.has-dark-gray-color {
  color: #43413B;
}
```

これで、カラーパレットの色がテーマ独自のものになり、公開サイト側でも再現されるようになります。

テーマ独自のカラーパレットが適用された

Step 1-5 フォントサイズの設定

標準のフォントサイズのプルダウンメニュー

カラーパレットと同じように、文字の大きさも右側のサイドメニューのフォントサイズから選択して、ブロックごとに指定できます。フォントサイズは、「add_theme_support('editor-font-sizes')」の第２引数に配列として記述して設定します。

例えば次の例のように記述すると、「注釈」という名前の、12pxのサイズで表示されるフォントスタイルが作成されます。

フォントスタイルの指定の例

```
add_theme_support(
  'editor-font-sizes',
  array(
    'name' => '注釈',
    'slug'. => 'annotation',
    'size' => 12
  ),
);
```

自動生成されるクラス名は、slug名の前に「has-」、後に「-font-size」という命名規則になっています。上記の例では、「has-annotation-font-size」となります。カラーパレットと同様、CSS側でこのクラス名に対応するスタイルを記述します。次の例のようにline-heightなど、font-size以外にもスタイルを加えて独自のフォントスタイルとして拡張することもできます。

フォントスタイルの指定の例に対するCSSの記述例

```
.has-annotation-font-size {
  font-size: 12px;
  line-height: 1.4;
}
```

それでは、サンプルテーマ独自のフォントサイズを設定していきましょう。

❶functions.phpに記述する

functions.phpに、次の記述を追加します。サンプルテーマでは、フォントサイズを５つ指定します。

functions.php

```
function neko_block_setup() {
  add_theme_support( 'wp-block-styles' );
  add_theme_support( 'responsive-embeds' );
  add_theme_support( 'align-wide' );
  add_theme_support(
    'editor-color-palette',
    中略
  );
  ここに記述する
}
add_action( 'after_setup_theme', 'neko_block_setup' );
```

```
function neko_block_setup() {
  add_theme_support( 'wp-block-styles' );
  add_theme_support( 'responsive-embeds' );
  add_theme_support( 'align-wide' );
  add_theme_support(
    'editor-color-palette',
    中略
  );
  add_theme_support(                    ← フォントサイズの設定
    'editor-font-sizes',
    array(
      array(
        'name' => '極小',
        'size' => 14,
        'slug' => 'x-small',
      ),
      array(
        'name' => '小',
        'size' => 16,
        'slug' => 'small',
      ),
      array(
        'name' => '標準',
        'size' => 18,
        'slug' => 'normal',
      ),
      array(
        'name' => '大',
        'size' => 24,
        'slug' => 'large',
      ),
      array(
        'name' => '特大',
        'size' => 36,
        'slug' => 'huge',
      ),
    )
  );
}
add_action( 'after_setup_theme', 'neko_block_setup' )
```

❷ CSSにフォントサイズのスタイルを記述する

上記で設定した値の通りに、CSSへスタイルを追記します。サンプルテーマにあらかじめ含まれているCSSの記述では、該当するpx相当のremに読み替えているため、次のような記述となっています。これで、テーマに合わせたフォントのスタイルをブロックエディターで指定できるようになりました。

theme-styles.css

```
.has-x-small-font-size {
  font-size: 0.875rem;      ← 14px相当
}
.has-small-font-size {
  font-size: 1rem;          ← 16px相当
}

.has-normal-font-size {
  font-size: 1.125rem;      ← 18px相当
}
.has-large-font-size {
  font-size: 1.5rem;        ← 24px相当
}
.has-huge-font-size {
  font-size: 2.25rem;       ← 36px相当
}
```

Step 1-6 エディター側にスタイルを適用する

ここまで行ってきた設定は、いずれも適用したスタイルを公開サイト側で表示させるためのものでした。このままでは、ブロックエディターでの編集中にこれらのスタイルが反映されたかどうかを確認することはできません。ブロックエディターでは、記事作成者（エディター側を使用するユーザー）の混乱を防ぎ利便性を高めるためにも、エディター側にも同様のスタイルを反映させていくことが望ましいです。

WordPress 5.8時点でのブロックエディターの仕様上、マージンやパディング等の細かいコントロールが難しく、すべてのスタイルを公開サイト側とまったく同じように反映させることが難しい部分もあります。しかし、フォントサイズなど無理なく反映させられるスタイルも多いので、公開サイト側とエディター側でスタイルの差異がなるべく少なくなるような設定を行いましょう。

ブロックエディターは、クラシックエディターと同じくテーマ用のエディタースタイル（編集画面で読み込まれるスタイル）をサポートしています。しかしWordPress 5.8時点でブロックエディターのCSSの読み込まれ方は、クラシックエディターとは少し異なります。具体的には、エディター上のコンテン

ツだけにスタイルが読み込まれるように、CSSが自動変換されます。例えば「body {...}」として記述したスタイルは「.editor-styles-wrapper { … }」へ書き換えられます。これは、編集画面でのコンテンツ以外の要素（各種メニューなど）のスタイルに影響が出ないようにするためです。

この書き換えを行う設定が、「add_theme_support('editor-styles')」です。これを設定した上でエディター側に読み込ませるCSSファイルを「add_editor_style()」関数で指定することで、エディター側にスタイルを反映することができます。

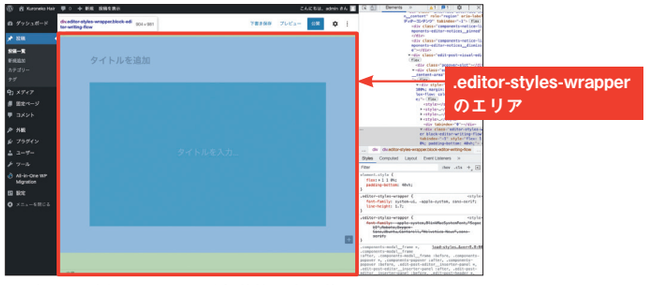
editor-styles.css

```
body {
  font-family: system-ui, -apple-
  system, sans-serif;
  line-height: 1.7;
}
  中略
.has-skyblue-background-color {
  background-color: #00A1C6;
}
```

エディター側の管理画面のインラインCSSへ変換される

```
.editor-styles-wrapper {
  font-family: system-ui, -apple-
  system, sans-serif;
  line-height: 1.7;
}

.editor-styles-wrapper .has-skyblue-
background-color {
  background-color: #00A1C6;
}
```



エディター側でスタイルが自動的に書き換えられる.editor-styles-wrapperのエリア

この自動変換が行われるため、CSSの書き方には少し注意が必要です。慣れるまでは、1か所ずつ確かめながら実装したほうがよいでしょう。

それでは、実際にサンプルテーマを使ってエディター側のスタイルを指定していきます。本来は必要に応じてCSSを自作する必要がありますが、サンプルテーマにはeditor-styles.cssがあらかじめ用意されています。こちらを読み込ませてみましょう。また同じadd_editor_style()関数を使って、公開サイト側と同じように（P.132）Googleフォントを読み込ませます。

❶functions.phpに記述する

functions.phpに、次の記述を追加します。これで、editor-sytles.cssに記述された内容が編集画面に反映されるようになります。

functions.php

```
function neko_block_setup() {
  add_theme_support( 'wp-block-styles' );
  中略
  );
  ←ここに記述する
}
add_action( 'after_setup_theme', 'neko_block_setup' );
```

↓

```
function neko_block_setup() {
  add_theme_support( 'wp-block-styles' );
  中略
  );
  add_theme_support( 'editor-styles' );  ←独自のエディタースタイルを有効化する
  add_editor_style( 'assets/css/editor-styles.css' );  ←独自のエディタースタイルを指定する
  add_editor_style( 'https://fonts.googleapis.com/css2?family=Noto+Sans+JP:wght@500&
  display=swap' );  ←エディター側にGoogleフォントを読み込む
}
```

関数 add_editor_style(スタイルシートへのパス)

独自のエディタースタイルをエディター側に反映させる。

引数

スタイルシートへのパス（オプション）：エディター側に反映させたいスタイルシートのパス。省略すると、テーマディレクトリー内でeditor-style.cssというファイル名を探し、存在すればそちらを反映する。

完成コード

Step1の作業は、これで終わりです。今回作成したファイルは次のようになりました。

functions.php

```
前略
function neko_block_setup() {
  add_theme_support( 'wp-block-styles' );  ←ブロック用CSSを有効化する
  add_theme_support( 'responsive-embeds' );  ←埋め込み要素のレスポンシブスタイルを適用する
  add_theme_support( 'align-wide' );  ←幅広、全幅のスタイルに対応する
```