教育用計算機システム使用の手引き（付録）

2025 年度版

筑波大学 情報学群 情報科学類

序

この手引き (付録) は筑波大学情報学群情報科学類の授業などでよく利用するコマンド，LaTeX，言語処理などについて説明するものです．情報科学類計算機システムの利用方法については，手引き本体 https://www.coins.tsukuba.ac.jp/tebiki/2025/tebiki2025.pdf を参照してください．

注意事項

教育用計算機と計算機室の利用　教育用計算機ならびに計算機室の利用にあたっては，手引き本体に記載されている「情報科学類（情報学類）教育用計算機システム利用規定」を遵守することが求められます．利用規定をよく読み，理解したうえで利用してください．

質問や問い合わせ　教育用計算機システムは，情報科学類計算機運用委員会が方針を決め，運用・管理しています．また，有志の学生らによる学生管理者（coins-admin）も保守・運用に貢献しています．運用・管理上の方針などについて質問があるときは，計算機運用委員会の教員に問い合わせてください．また，細部に関する質問や指摘については，技術職員や学生管理者に直接問い合わせるか，ニュースやメールなどで問い合わせをしてください．連絡先は， 本書章に記載されています．システムの変更・停止などの情報は，次の計算機システムページに随時掲載されますので参照してください．https://www.coins.tsukuba.ac.jp/ce/

もし紙で本書を読んでいる方　本書は，電子ファイルのみで配布されています．PDF 形式のファイルを以下の URL からダウンロードできます．https://www.coins.tsukuba.ac.jp/tebiki/2025/tebiki_appendix2025.pdf

PDF 形式のファイルを用いることで，調べたい語句の検索などをより容易に行うことができます．有効に活用しましょう．

目次

第 1 章　コマンド

ここでは，UNIX をコマンドで操作する方法について，UNIX の一つである Ubuntu を使用して学びます．Windows の PowerShell や macOS のターミナルとの相違点はさほどありませんが, ある場合にはそれぞれの場所で特記します． 以降，1.1 でコマンドを入力する「端末」の起動方法，1.2 から 1.4 でコマンドの基本やディレクトリの構造を述べて，1.5 でコマンドでファイルを操作する方法について説明します．また，1.6 に UNIX コマンドの一覧を提示し，1.7 以降で各コマンドの使い方を紹介します．

1.1　端末の立ち上げ方

まず，コマンドを入力する「端末」の立ち上げ方を説明します．立ち上げ方は 3 つあります．ただし，ログイン方法によっては利用できないものもあります．

1. ショートカットキーを使う方法
   control と alt と T を同時に押します．
2. 「アプリケーションを表示する」から検索
   「アプリケーションを表示する」（お気に入りの一番下にある，● が 9 つ正方形状に並んだマーク．図 1.1 参照）をクリックすると，検索ウィンドウが出現します．このウィンドウに「term」と入力し，出てきた「端末」をクリックします（図 1.2）．

図 1.1: 「アプリケーションを表示する」のアイコン

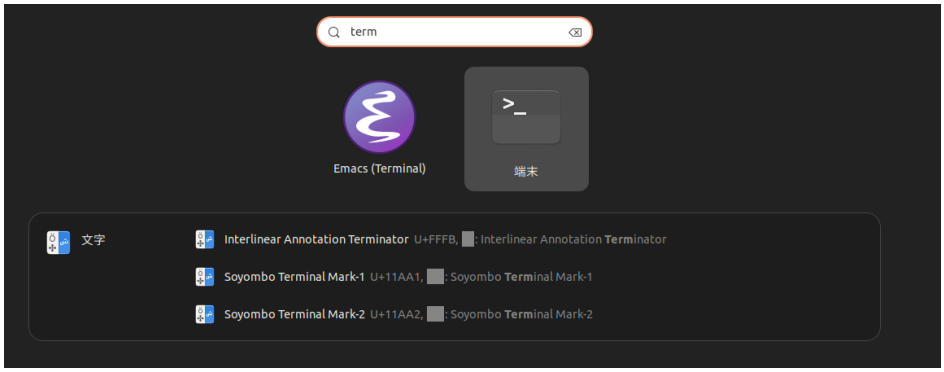


図 1.2: 検索ウィンドウに「term」と入力し，「端末」が見つかったところ．

なお，図 1.2 では「Emacs (Terminal)」も表示されていますが，これは別のアプリケーションです．

無事に「端末」が起動したら，以下のような画面が起動します．（図 1.3）

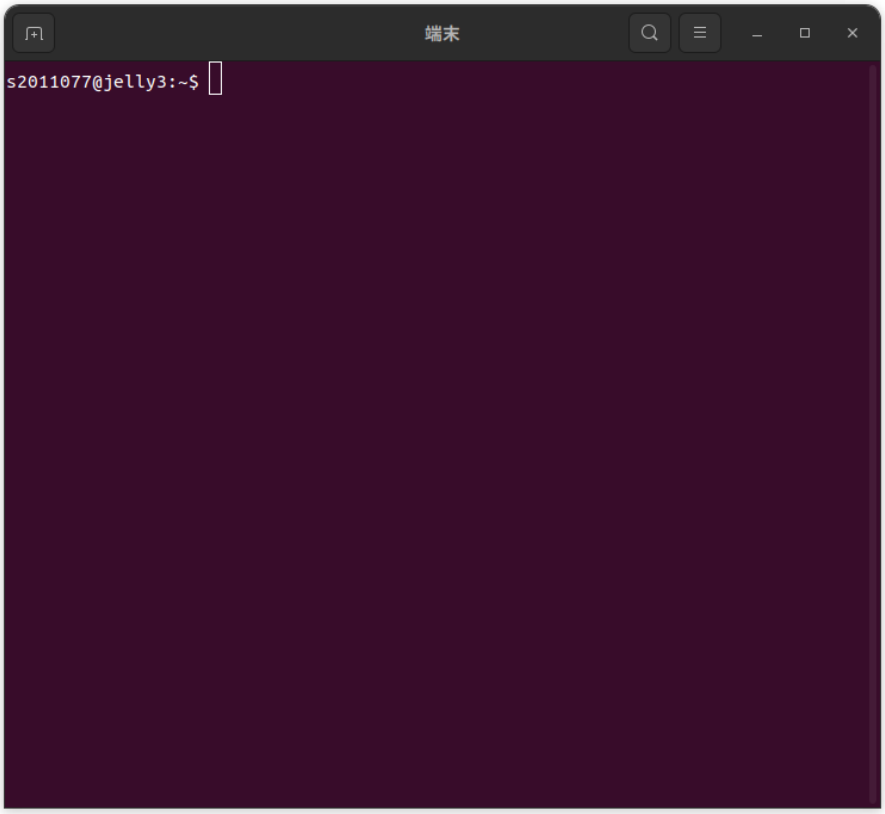


図 1.3: 「端末」の画面

1.2　イントロダクション

これ以降，$の前は本筋とは関係ないので省略します．これと$はあわせてプロンプトと呼ばれます．プロンプトはコマンドを受け付けることを示すために，コンピュータが出力します．このメッセージを改造して，時間や，これから説明するカレントディレクトリ，実行しているマシンの名前やユーザ名などを表示させたり，コマンドの実行が成功したか失敗したかによって，対応する顔文字を表示させたりする人もいます．かわいいプロンプトを使っている人を見かけたら，どういう風にしているか声をかけて聞いてみましょう

1.2.1　Hello, world

こんな感じの，味気ない表示がウィンドウの上の方にでていますか？

```
ls2011077@azalea13:~$
```

ここで，e と打つと，ちゃんと e と表示されます．

とにかく最初はなにがなんだかわからないかもしれないですが，習うより慣れろですから，とにかく echo Hello, world と打ってみましょう．こんな見ためになればとりあえず OK です．

```
$ echo Hello, world
```

$はもとから出力されていますから，e から入力すればよいです．

さて，ここで ↵ を叩いてみましょう．下のようになれば成功です．

```
$ echo Hello, world↵
Hello, world
$
```

さて，今，何をしたのでしょうか．

1.2.2　コマンドと引数

実は，今，**コマンド**というものを実行しました．$ から最初の空白までがコマンド名です．

この場合， echoがコマンド名です．

そして，最初の空白以降は**引数**です．引数というのは何でしょうか．例えば， echoというのは様々な文章を出力できる汎用的なコマンドです．ですから，Hello のほかにも bye や Stand up Workers など様々なことを出力できます．そこで，コマンドを実行するときには具体的に何をすればいいのか教える必要があります．その教える内容というのが引数で，コマンドに続けて書くことで，コマンドに指示できます．

1.2.3　コマンドの基本

さて，コマンドは様々なことができると言いました．それでは，具体的にどのように，何ができるのでしょうか．なぜ，わざわざターミナルからコマンドを利用するのでしょうか．

コマンドは，今やってみたとおり，文字をベースにコンピュータに下す命令です．コンピュータは，プロンプトから ↵ の入力までの文字列をコマンドとコマンドへの引数であると解釈して命令を実行します．文字で命令を書くことで，手順書のようにコンピュータに何をすればいいのかあらかじめ教えておくことができます．そうすることで，時間のかかる操作や何度も繰り返す単調な操作を，人間の手間をかけることなく行うことができます．

コマンドを使いこなすことで，たとえば，1000 枚の写真と 200 個の文書が含まれている 100 個の ZIP ファイルを全部インターネットからダウンロードして開き，写真だけをすべて 50%縮小して 1 つの ZIP にいれ，文書は特定の語を含むファイルだけ抜きだし 1 つのテキストファイルにまとめて友達のコンピュータに送信するといった，マウスを使って操作していると気の遠くなるようなことが簡単にできるようになります．

それには，この章で説明する内容を越えた知識が必要となりますが，本書では最初のステップを踏み出す手引きをします．echo コマンドではファイルを必要としませんでしたが，上の例のようなファイル操作をするためには，まずファイルについて知る必要があります．

1.3 用語の整理

ここで，ディレクトリ構造の話をするときによく出てくる用語を紹介します．

- **コマンド (command)**
  コマンドは，コンピュータに対する命令です．UNIX システムには，たくさんのコマンドが搭載されていて，自分で増やすこともできます．コマンドを発行するときは，端末，端末エミュレータなどと呼ばれるソフトウェア (本章では Ubuntu のデフォルトのシェルである bash に限定して解説してあります) を用いて文字を使ってコマンドを発行します．
- **引数 (argument)**
  コマンドに，動作の詳細を教えるものです．コマンドに続けて書きます．引数の数はいくらでもいいのですが，認識できる限界や順序，書式はコマンドごとに限界があります．
- **ディレクトリデリミタ (directory delimiter)**
  ディレクトリの名前と名前をわけるものです ． /の下にある bin に入っている ls というファイルを指すには/bin/ls と表記します ． ルートディレクトリ (**??**節参照) をさす"/"とデリミタの"/"は同じ文字ですが，意味が違うことに注意しましょう ． 同じ文字である必然性はないので，システムによっては違う文字になっていますが，UNIX ではどちらも"/"という文字を使うという約束になっています . 同じ文字なら同じ意味だと思い込んでいると，理解の妨げになるので注意しましょう ．
- **木構造**
  階層を持つデータ構造の 1 つです．階層が深くなるごとに枝分かれしていく様子が木に似ていることからこう呼ばれます．

- **ルートディレクトリ (root dhirectory)** ディレクトリの木構造において根の位置にあるディレクトリを指します.
- **カレントディレクトリ (current directory)，カレントワーキングディレクトリ (current working directory)**
「自分が今作業をしているディレクトリ」のことです．たとえば，図 1.6 の中の「bin」というディレクトリで作業をしている場合，「カレントディレクトリは/bin である」といいます.
- **パス名 (path name)**
ここでは，あるディレクトリやファイルの場所を表す文字列を意味します ．たとえばディレクトリデリミタの例に出てきた"/bin/ls"はパス名の例です ．
- **相対パス名と絶対パス名**
力学には相対座標と絶対座標という言葉がありますが，UNIX においてもあるディレクトリやファイルまでの経路を表す方法に絶対パス名を記述する方法と相対パス名を記述する方法があります．絶対パス名はルートディレクトリから中継するすべてのディレクトリを記述したものです．bin ディレクトリ以下の ls を指定するには下記のように指定します.

```
$ /bin/ls↵
```

これに対して相対パス名とは，カレントディレクトリを基準とした経路を表すものです．図 1.6 において bin のディレクトリにて作業をしているとします．ここで bin ディレクトリ以下の ls を指定するには以下のように指定します.

```
$ ./ls↵
```

ここで「.」はカレントディレクトリを表します.
ファイルの指定を「/」記号から始めると，それは「絶対パス名」であるとみなされます[1)]．「/」以外の文字始めると，それは「相対パス名」であるとみなされます.
- **(ファイル名の) 拡張子 (extension)**
ファイル名は (文字種の制限を守る限り) 自由に付けることができますが，ファイル名の最後に，ファイルの種類を表わす文字列を「.」で区切って書くことがあります．これを拡張子といいます．たとえば，Java 言語のプログラムが書かれたファイルには「.java」，音楽は「.mp3」や「.ogg」といったものです.

1.3.0.1 拡張子の表示の仕方

拡張子は以下の方法で表示させることができます. 適宜画像を参考にしてください.

– Windows の場合

1. エクスプローラーを開く
2. 表示をクリック

1) 電話番号が 0 から始まると市外局番になるようなものです．ファイルに至る経路をすべて記述することから，「フルパス」とも呼ばれます.

3. 表示をクリック
4. ファイル名拡張子にチェックを入れる

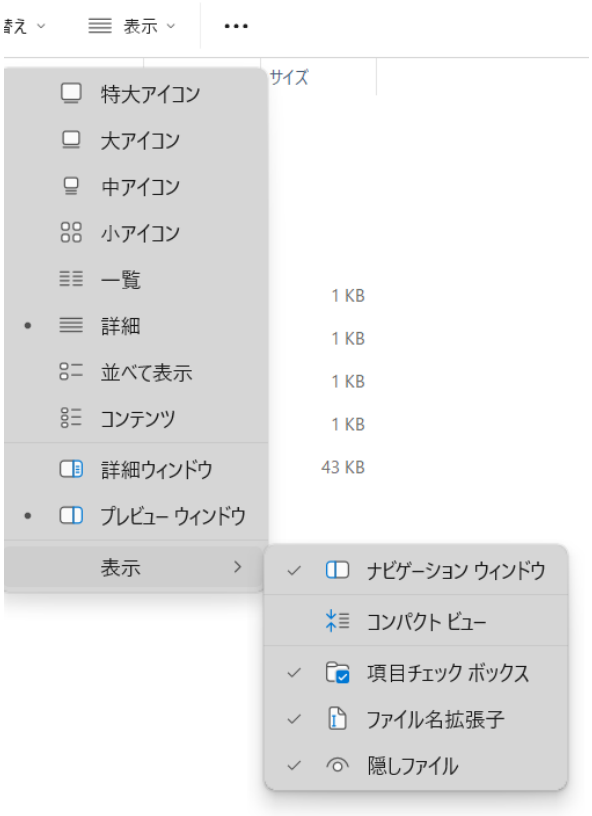


図 1.4: 拡張子を表示させる際の画面

1.3.1 ホームディレクトリ (home directory) について

ユーザには，各自が自由に使える「ホームディレクトリ」が割当てられています．みなさんの実際の作業は，各自のホームディレクトリの下で行います．ホームディレクトリは，「~」(チルダ) 記号で表わされます．たとえば「自分のホームディレクトリにある『デスクトップ』というディレクトリの…」というようなときに「~/デスクトップ/…」のような使いかたをします．また，他のユーザのホームディレクトリも「~ユーザ名」とすると表せます．以下に例を示すので，自分のホームディレクトリと他人のホームディレクトリの表し方を確認してみましょう．

```
$ cd ~/デスクトップ ⏎ ← ホームディレクトリにある デスクトップ
    というディレクトリに移動する
$ pwd ⏎
/home/ugrad/xx/sxxxxxxx/デスクトップ
$ cd ~syyyyyyy ⏎ ← syyyyyyy というユーザのホームディレクトリに移動する
$ pwd ⏎
/home/ugrad/yy/syyyyyyy
```

```
$
```

1.3.2　ファイルのパーミッション (permission)

各ファイル・ディレクトリは，ファイルのアクセス権を設定するための「パーミッション (permission)」という属性を持っています．パーミッションについては，「ls コマンド 1.7.1」「chmod コマンド 1.8.7」の所でもう少し詳しく述べます．ファイルのパーミッションの設定には気をつけてください．設定を間違えると，自分のメールを他人に読まれたり，大事なファイルが他人に消されたりということが起こり得ます．ファイルシステムについてのより詳しい説明については，UNIX についての本がたくさん出ていますので，それらを参照するのが良いでしょう．

1-1

1.4　ディレクトリ構造と木構造

ここでは操作の対象となる**ファイル**と**ディレクトリ**について説明します．まず身近にあるファイルについて考えてみましょう ．1 つファイルを想像してみてください ．たとえば，この手引きの PDF ファイルでもいいでしょう．それを念頭にちょっと話を進めていきます．

UNIX では， ファイルに英文字・数字・いくつかの記号からなる名前 (ファイル名) をつけて管理します．そして，ファイルをまとめるものとして**ディレクトリ** *(directory)*[2)] というものがあり，このディレクトリの中に，ファイルとディレクトリを格納できます．

ちょうど，入学式でもらったファイルをまとめておくと，あとあと困らないので，バインダーに閉じたり[3)]，ひとところに纏めて置いたりしたと思いますが，それと同じようなことが，人間が使いやすいようにするためにコンピュータでもできるようになっているわけです ．

そしてこういったお約束ごとがあるので，ひとつのコンピュータに複数のファイルを分類して入れることができるというわけです．

なお，1 つのディレクトリ内に同じファイル名のファイルを 2 つ以上格納することはできません．ただし，図 1.5 のように，別のディレクトリにあるファイル同士はファイル名が同じでも共存可能です．

2)　フォルダ (folder) と呼ばれることもあります．
3)　ただし，コンピュータの中ではバインダーの中にバインダーを閉じるといったことは容易に行えます

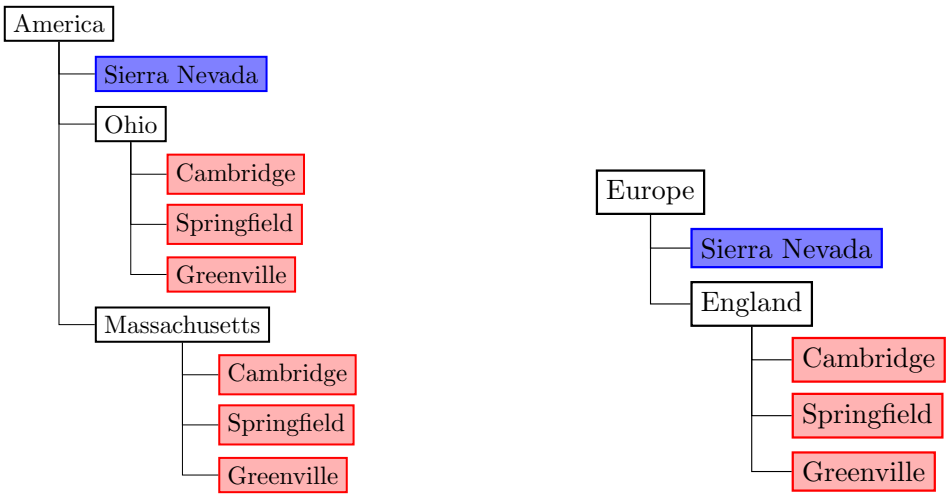


図 1.5: ファイル名が同じでも，ディレクトリが違えば共存できる

こういった構造や，それを支えるシステムをディレクトリ構造，と呼び習わすことになっています．そういうわけで，UNIX のディレクトリ構造は，図 1.6 のように，ディレクトリにファイルを入れることができます．

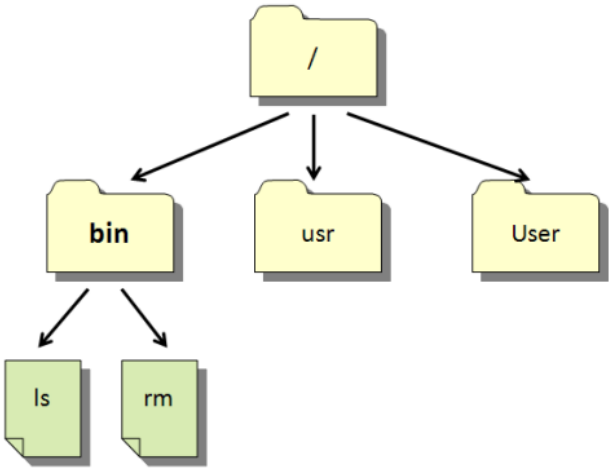


図 1.6: ディレクトリ構造の例

ディレクトリ構造とは，こういったファイルやディレクトリを扱う**システム**と構造の事です．ディレクトリをさらにディレクトリに入れることができるので，図 1.6 のような入れ子構造を作ることができます．

さて、ディレクトリ構造のような概念を指す用語として「木構造（tree strategy）」という言葉があります。この言葉は、ディレクトリ構造をはじめ、構造的に

図 1.6 の例では，まず「/」で表される「木の根」(親ディレクトリのないディレクトリ__を**根** (あるいは *root*/**ルート**) と呼びます．

」にあたる**ルートディレクトリ**というものがあります．

これから皆さんが触れていく UNIX では，1 つのコンピュータはただひとつのディレクトリツリーをもち，接続されたドライブやネットワーク上のディレクトリ構造もすべてこのディレクトリツリーに参加することになっています．

つまり，ルートディレクトリとは，コンピュータからみえるすべてのファイルとディレクトリの根っこにあたるものです．これより上のディレクトリはありませんし，ここからたどれないディレクトリ構造はありません．UNIX のディレクトリ構造は「/」というディレクトリからなる唯一のディレクトリツリーを持ち[4]，すべてのディレクトリの “根” になっています．

さて，図 1.6 のルートディレクトリには，その中に “bin”“usr”“User” という 3 つのディレクトリがありますね．ディレクトリ “bin” の中にはさらに，“ls”“rm” というファイルがあります．これらは実はファイルであるだけでなく，プログラムでもあります．プログラムはファイルなのです．

Windows には`tree`というコマンドがあり，カレントディレクトリ (後述) をルートとしたディレクトリ構造が見られます．Mac では似たコマンドとして`ls`コマンドがあります．

ディレクトリの木構造についてさらに詳細な内容は，次のページを参考にしてください．

https://www.coins.tsukuba.ac.jp/~yas/coins/literacy-2018/2018-04-24/index.html

1.5 ファイルをコマンドラインで操作する

1.5.1 ls - 一番有名なコマンドを使ってみる

とにかく，`ls /`と打ってみましょう．最初の$はもう画面にでているはずなので打たないこと，最後に ↵ を押すことを忘れずに．

```
$ ls /↵
V-Boot   cdrom   home   lib32   lost+found   opt   run   srv   tmp   vdsk
bin   dev   host   lib64   media   proc   sbin   swapfile   usr
boot   etc   lib   libx32   mnt   root   snap   sys   var
$
```

このコマンドは，引数にわたされた文字列をディレクトリ名と解釈して，そのディレクトリの内容を表示する命令です．では，“/” とはどんなディレクトリかといいますと，??節で述べたルートディレクトリのことです．いちいちルートディレクトリと打つとかったるいですし，打ち間違えはバグの元ですから，短く “/” という名前になっているのです．そのコマンドの結果が，上のリスト 2 行め以降の内容になります．

4) Windows では，ハードディスクは C:\，USB メモリは Z:\，といったようにディレクトリツリーが別れており，いわば森のような構造になっています．このようにな構造をとっている理由としては，ハードディスクが高価だった頃の名残で，起動用のフロッピー，アプリケーション起動用のフロッピーや保存用のハードディスクなど，媒体を分けていたためです．

次に，単に`ls`と打つとどうなるでしょう．

```
$ ls↵
```

なにか出てきたかもしれませんし，何も表示されなかったかもしれません．あなたがもう何かファイルを作っていた場合にはそれが表示されるのです．よく分からないと思うので，ちょっと手習い用のディレクトリを作成して，そこですこし実験してみましょう．

1.5.2 最初の手習い - ディレクトリの作成

まず，`mkdir erste`と打ってみましょう．これは，ディレクトリを作成するコマンドです[5]．すでに同名のディレクトリがあった場合は，別の引数を渡しましょう[6]．

Listing 1.1: ディレクトリが正しく作成された場合

```
$ mkdir erste↵
$
```

Listing 1.2: ディレクトリがすでに存在したので重複して作成できなかった場合

```
$ mkdir erste↵
mkdir: ディレクトリ ‘erste’ を作成できません: ファイルが存在します
$
```

もういちど`ls`してみると，前に`ls`したときと比べて，今作成したディレクトリが新しく登場していると思います．

mkdir コマンドは，ディレクトリを作成するコマンドだといいました．しかし，このコマンドはディレクトリを作成するだけで，その中に移動するということまでは自動ではやってくれません．[7]．

そこで，次に自分でそのディレクトリに移動することによって，作業をするディレクトリを変える必要があります．

1.5.3 今いる場所を知る

ところで，「作業をするディレクトリ」とはどういう意味でしょう．実はこうしてコマンドを打つとき，あなたもファイルと同様ルートディレクトリから辿れるある場所に立って，そこからコマンドを打っているのです．その場所を知るために，pwd というコマンドを使って調べてみます．

```
$ pwd↵
```

`/home/ugrad/25/s2511xxx` というようなあるディレクトリが表示されるかもしれませんし，もしかしたら別のところにいるのかもしれませんが，どこにいるのかに関わらず，ある場所が存在し

5） MaKe DIRectory の略です
6） `mkdir zweite`などと打ってください
7） これは，一度にたくさんディレクトリを作るときなどに便利です．このように，一つのコマンドがあれやこれやお節介を焼くべきではない，言われたことだけちょうど行うにとどめるべきだ，というプログラムに関する哲学に基づいて，一つのコマンドごとにだいたいざっくりと一つの機能だけに絞り込んで UNIX のコマンドは作られています

て，そこでディレクトリを作成したところです．作業をするときは，必ずシステムの中のどこかにいてそこから作業をしているということを忘れないでください．

1.5.4　ディレクトリの移動

ということで，現在は，自分の居場所について知り，そして，そこにディレクトリを作成したところでした．今度は，そのディレクトリに入ってみようとおもいます．

Listing 1.3: 先ほど作成したディレクトリが erste だった場合

```
$ cd erste↵
$
```

cd[8]は，自分の作業をするディレクトリを変えるコマンドです．もう一度pwdすると，作成したディレクトリに移動していることがわかります．

1.5.5　ファイルを作る

さて，ここでlsしてみてください．おそらくまだ何もないと思います．そこで，新たにファイルを作ってみようと思います．

```
$ touch test↵
$
```

もう一度，単にlsと打ってみましょう．

test というファイルが追加されているはずです．

では今度は拡張子付きのファイルを作ってみましょう．今回は最も一般的な拡張子である.txt のテキストファイルを作ってみたいと思います．作り方は先ほどと同じで、ファイル名に拡張子をつけるだけです．

```
$ touch test.txt↵
$
```

コマンドが実行できたら, これでまたlsと打ってみましょう．

test.txt というファイルが追加されているはずです．

さらにいろんなコマンドを下で解説していますが，まだよく理解していない機能を試すときは，重要なファイルを誤って消さないように，こうしてテスト用のディレクトリを作って行うとよいでしょう．

8）Change the working Directory の略です

1.6 コマンドリスト

今紹介したものを含め，UNIX にはたくさんのコマンドがあります．それらを表 1.1 にまとめました．各コマンドの詳細は，man コマンドを用いるなどして調べてください．

表 1.1: コマンドリスト

コマンドの機能	コマンド名
(ファイルシステム関連)	
ディレクトリを移動します.	cd
ディレクトリスタックを操作します.	pushd, popd
ファイルのパーミッションを変更します.	chmod
ファイルのコピーをします.	cp
ディスク使用量を表示します.	du
ファイルのリンクをします.	ln
ディレクトリにあるファイル名を表示します.	ls
ディレクトリを作成します.	mkdir
ディレクトリを削除します.	rmdir
ファイルの移動，ファイル名の変更をします.	mv
ファイルを削除します.	rm
カレントディレクトリを表示します.	pwd
ディレクトリスタックの現在の状態を表示します.	dirs
ファイルの圧縮・展開をします	tar
(一般的な作業コマンド)	
テキストファイルの内容を表示します.	cat，more，lv
日付・時刻を表示します.	date
カレンダーを表示します.	cal
他の計算機でコマンドを実行します.	ssh
ネットワーク上のホスト間でファイルをコピーします.	scp
プログラムの実行時間を計測します.	time
(「フィルタ」として使われるもの)	
ファイル内の，指定した文字列を含む行を抜き出します.	grep，egrep
ファイルの内容を，先頭から指定した行数だけ表示します.	head
ファイルの内容を，末尾から指定した行数だけ表示します.	tail
ファイルの内容を，行単位でソートして表示します.	sort
(プロセスに関するもの)	
プロセスにシグナルを送ります.	kill

表 1.1: コマンドリスト

コマンドの機能	コマンド名
計算機で動いているプロセスの情報を表示します.	ps，top，pstree
(プリント関連のコマンド)	
印刷をします.	lp
プリンタへの印刷の要求を取り消します.	cancel
プリンタへ登録されているジョブの確認をします.	lpstat
(その他のコマンド)	
コマンドの説明を表示します.	man, info
その計算機にログインしている人を表示します.	who，finger
自分のユーザ名を表示します.	whoami
指定したユーザのユーザ ID やグループ ID などを表示します.	id
指定したユーザのグループ ID を表示します.	groups
その計算機で誰が何をしているかを調べます.	w，last
ファイルやディレクトリの所属グループを変更します.	chgrp
システムのアーキテクチャを表示します.	arch
ウィンドウに表示されている画像をファイルに保存します.	xwd
グラフを書きます.	gnuplot
図を書きます.	tgif
絵を書きます.	xpaint，gimp
画像形式の変換や，拡大縮小などの編集をします.	convert, sips

1.7　ファイルに関するコマンド

1.7.1　ls コマンド

自分が現在作業をしているディレクトリ (カレントディレクトリといいます) にあるファイルの名前を表示します.

```
$ ls↵
dir1 file1 file2
$ ls /↵
V-Boot   cdrom   home   lib32   lost+found   opt   run   srv   tmp   vdsk
bin   dev   host   lib64   media   proc   sbin   swapfile   usr
boot   etc   lib   libx32   mnt   root   snap   sys   var
$
```

この例では，まず引数を与えずに ls コマンドを実行し，3 個のファイルおよびサブディレクトリの名前が表示されました．次に，引数としてディレクトリを与えることで，そのディレクトリにあるファイルやサブディレクトリの名前を表示しました．このように ls コマンドの引数としてディレクトリを与えることで，そのディレクトリの中身を表示できます.

ls コマンドに，「-l」(小文字の L) というオプションを付けることにより，それぞれのファイル・ディレクトリについての詳しい情報を見ることができます.

```
$ ls -l↵
合計 3
drwxr-xr-x 2 s2011077 ugrad  2  4月  5 17:54 dir1
-rw-r--r-- 1 s2011077 ugrad  6  4月  5 17:54 file1
-rw-r--r-- 1 s2011077 ugrad 11  4月  5 17:55 file2
$
```

ここで，ファイルのパーミッションの表示について説明します．左から，「ファイルのパーミッション情報」「リンク数」「ファイル所有者」「ファイル所有グループ」「ファイルサイズ」「ファイルの最終更新日時」「ファイル名」を示しています．例として，上の file1 に関して見てみます.

```
-rw-r--r-- 1 s2011077 ugrad 11  4月  5 17:55 file1
```

いちばん左に`-rw-r--r--` と表示されていますが，これがそのファイルのパーミッションの表示です．この 10 桁の文字列のうち，一番左の 1 文字は，そのファイルの種類を示しています．これが「-」なら通常のファイル，「d」ならディレクトリ，「l」なら，シンボリックリンクです[9]．残りの 9 桁が，実際のアクセス許可情報を持っています．この 9 桁の表わす情報は，表 1.2 のようになっています.

9)　他にもありますが，それらは man で調べてみてください.

表 1.2: 9 桁の文字列の意味

左側 3 桁	ファイルのオーナーの持つ権利
中央 3 桁	グループの持つ権利
右側 3 桁	その他の人の持つ権利

それぞれの 3 桁が，「読み」「書き」「実行」についての許可を表わしています (表 1.3) [10)]．

表 1.3: 文字の意味

左の桁が r	読むことが許可されている．
中央の桁が w	書き込みが許可されている．
右の桁が x	実行 (ディレクトリに対しての場合は，探索) が許可されている．
-の表示	許可されていない．

これらの表から，先程の例のファイルは，「オーナーは読み・書きができ，グループの人とその他の人は読むことのみできる」ということがわかります．

今までの例では，「ドットファイル[11)]」を見ることができません．ドットファイルを見るときには，「-a」オプションを付けます．また，「-F」オプションをつけると，ディレクトリ名の末尾に/が付いて判別しやすくなります．日本語のファイル名が正しく表示されないときは「-v」をつけると正しく表示されるかもしれません[12)]．これらの他，ls コマンドには多数のオプションがありますが，それらの詳細は「man ls」として，ls コマンドのマニュアルを参照してください．

1.7.2 mkdir コマンド

mkdir コマンドは，新しくディレクトリを作成するコマンドです．`mkdir` ディレクトリ名 のようにして使います．

```
$ ls -l↵
合計 3
drwxr-xr-x 2 s2011077 ugrad  2  4月  5 17:54 dir1
-rw-r--r-- 1 s2011077 ugrad  6  4月  5 17:54 file1
-rw-r--r-- 1 s2011077 ugrad 11  4月  5 17:55 file2
$ mkdir dir2↵
$ ls -l↵
合計 3
drwxr-xr-x 2 s2011077 ugrad  2  4月  5 17:54 dir1
drwxr-xr-x 2 s2011077 ugrad  2  4月  5 17:56 dir2
-rw-r--r-- 1 s2011077 ugrad  6  4月  5 17:54 file1
-rw-r--r-- 1 s2011077 ugrad 11  4月  5 17:55 file2
```

10） Windows などでは.COM や.EXE という拡張子が実行ファイルになりますが，UNIX ではそのファイルが実行ファイルかどうかはこの実行パーミッションによって識別されます．

11） 「.」記号ではじまるファイル・ディレクトリのことです．設定ファイルなどによく使われます．

12） それでも正しく表示されないときはターミナルの設定で文字セットエンコーディングを変更してみてください．

```
$ 
```

1.7.3 rmdir コマンド

rmdir コマンドは，空のディレクトリを削除するコマンドです．rmdir ディレクトリ名 のようにして使います．

```
$ ls -l↵
合計 3
drwxr-xr-x 2 s2011077 ugrad  2  4月  5 17:54 dir1
drwxr-xr-x 2 s2011077 ugrad  2  4月  5 17:56 dir2
-rw-r--r-- 1 s2011077 ugrad  6  4月  5 17:54 file1
-rw-r--r-- 1 s2011077 ugrad 11  4月  5 17:55 file2
s2011077@azalea13:~/erste$
$ rmdir dir2↵
$ ls -l↵
合計 3
drwxr-xr-x 2 s2011077 ugrad  2  4月  5 17:54 dir1
-rw-r--r-- 1 s2011077 ugrad  6  4月  5 17:54 file1
-rw-r--r-- 1 s2011077 ugrad 11  4月  5 17:55 file2
$ 
```

1.7.4 cp コマンド

cp コマンドは，ファイルをコピーするためのコマンドです．あるファイルを，別のファイル名にコピーしたい場合，cp コピー元ファイル名 コピー先ファイル名 のようにします．

```
$ ls↵

$ cp file1 file2↵
$ ls↵
file1 file2
$ cp file2 file3↵
$ ls↵
file1 file2 file3
$ 
```

コピー先ファイル名にディレクトリ名を指定すると，そのディレクトリの中にファイルのコピーが作られます．またこの場合には，コピー元ファイル名を複数指定できます．

```
$ mkdir dir1↵
$ ls dir1↵
$ ls -F↵
dir1/ file1 file2
$ cp file1 file2 dir1↵
$ ls dir1↵
file1 file2
```

```
$ mkdir dir2↵
$ cd dir2↵
$ cp ../file1 .↵          ←
    コピー先ディレクトリにドットを指定するとカレントディレクトリにコピーできる
$ ls↵
file1
$
```

1.7.5 mv コマンド

mv コマンドは，cp コマンドとは異なり，「ファイルの移動」「ファイル名の変更」を行なうコマンドです．mv [移動元ファイル名 (複数可)] [移動先ディレクトリ名] あるいは mv [旧ファイル名] [新ファイル名] のような書式で使います．

```
$ ls -F↵
dir1/ file1            ←ファイル1つに，ディレクトリ1つ．
$ ls dir1↵
   ←dir1 の中にファイルはない．
$ mv file1 file2↵
$ ls -F↵
dir1/ file2                  ←ファイル名が変わった．
$ mv file2 dir1↵
$ ls -F↵
dir1/
$ ls dir1↵
file2                        ←ファイルが移動した．
$
```

1.7.6 rm コマンド

rm コマンドは，ファイルの削除をするためのコマンドです．rm [ファイル名 (複数可)] のようにして使います．また，rm -r [ディレクトリ名] のようにすると，指定したディレクトリの中のファイルすべて，および，そのディレクトリ本体をすべて削除します．「-i」オプションをつけて実行すると，削除するファイルについて，それぞれ本当に削除するかどうか聞いてきます．この質問には，y (yes) か n(no) で答えてください．

```
$ ls -F↵
dir1/ file1      ←ファイル1つに，ディレクトリ1つ．
$ ls dir1↵
file2             ←ディレクトリ内に，ファイル1つ．
$ rm -r dir1↵
$ ls↵
file1              ←ディレクトリが削除された．
$ rm -i file1↵
rm: 通常ファイル 'file1' を削除しますか?      ← y と答えると
$ ls↵ ←ファイルが削除された．
$
```

rm コマンドを使用する際は，誤って必要なファイル・ディレクトリを削除しないように注意しましょう．特に， `rm -r *`[13]のようなコマンドは要注意です．このコマンドはカレントディレクトリ，およびそれより下にあるすべてのファイルおよびディレクトリを消去するコマンドです．つまり，ディレクトリ単位でざっくり消してしまえるので便利といえば便利ですが，消してしまった後に必要となると，復元できなくなってしまうこともあります．削除する前に本当に削除してしまっても大丈夫なのかチェックしてから削除しましょう．

1.7.7 cd コマンド

カレントディレクトリを移動するコマンドです．cd [ディレクトリ名] のようにして使います．ディレクトリ名を省略した場合は，「自分のホームディレクトリへの移動」となります．また，「cd -」とすると移動する前のディレクトリに戻ります．

1.7.8 pwd コマンド

今いるディレクトリの絶対パス名を確認できるコマンドです．cd コマンドであっちこっちに行っているとシンボリックリンクなどに惑わされて，今自分がどこにいるか判らなくなることがよくあります．そのようなときに pwd コマンドを使うと，自分のいるディレクトリがわかります．

```
$ pwd↵
/home/ugrad/99/s99xxxxx
$
```

1.8 ファイルの中身に関するコマンド

1.8.1 cat コマンド

ファイルの内容を画面に表示するコマンドです．cat [ファイル名] のようにして使います．そのファイルがテキストファイル (通常の文字からなるファイル) ならば，その内容を読むことができます．ファイル名を複数指定すると，指定した順番にファイルを続けて表示します．これを利用して，複数のファイルを連結する際にも使うことができます．

```
$ cat file1↵
This is test 1.
$ cat file2↵
This is test 2.
$ cat file1 file2 > file3↵
$ cat file3↵
This is test 1.   ←file1, file2の内容がfile3に入っている.
This is test 2.
$
```

13） * はワイルドカードで 0 文字以上の任意の文字列にヒットします．

1.8.2 lv コマンド

cat コマンドでは，1 画面を越える長さのファイルを読もうとすると，ファイルの最初の方がスクロールして画面から消えてしまい，内容が読めません．長いファイルを読むときに，1 画面ずつ読むためのコマンドが lv[14]です．ここでは，lv のキー操作を簡単に表 1.4 にまとめておきます．

表 1.4: lv コマンドのまとめ

キー	機能
q	lv を終了します.
j	1 行読み進みます (ファイルの後方に向ってスクロールします).
k	1 行戻ります (j の逆です).
f,(space)	1 画面読み進みます.
b	1 画面戻ります (f の逆です).
g,>	先頭に移動します.
G,>	末尾に移動します.
/ パターン	パターンで示された文字列の検索を，ファイル後方に向って行ないます.
? パターン	パターンで示された文字列の検索を，ファイル前方に向って行ないます.
n	直前の検索を後方に向かって繰り返します.
N	直前の検索を前方に向かって繰り返します.

lv は，「文書・文章の入ったファイルを表示し，その内容を (人間が) 読む」ことを目的としたコマンドですが，cat については，ファイルを表示するためだけでなく，ファイルの内容を連結するためにも使われます.

1.8.3 head コマンド

head コマンドは，指定したファイルや標準入力の最初の 10 行を表示するコマンドです．`-n 数字` オプションを付けることにより，先頭から任意の行数を表示できます.

```
$ cat file1↵
line 1.
line 2.
line 3.
line 4.
line 5.
line 6.
$ head -n 3 file1↵
line 1.
line 2.
line 3.
```

14) こういうコマンドを「ページャ」ともいいます.

```
$ 
```

1.8.4 tail コマンド

tail コマンドは，指定したファイルや標準入力の最後の 10 行を表示するコマンドです．head コマンドと同様に，-n オプションを付けることにより，最後から任意の行数を表示できます．

```
$ tail -n 4 file1↵
line 3.
line 4.
line 5.
line 6.
$ 
```

1.8.5 grep コマンド

grep コマンドは，ファイルや標準入力に対して指定した文字列を検索し，その文字列が含まれる行を表示するコマンドです．

```
$cat file2↵
tsukuba
kenkyugakuen
bampakukinenkoen
midorino
miraidaira
moriya
$ grep ba file2↵
tsukuba
bampakukinenkoen
$ 
```

1.8.6 sort コマンド

sort コマンドは，ファイルや標準入力に対して，行単位でのソートを行うコマンドです．

```
$ sort file2↵
bampakukinenkoen
kenkyugakuen
midorino
miraidaira
moriya
tsukuba
$ 
```

1.8.7 chmod コマンド

chmod[15]コマンドは，ファイルのパーミッション 1.3.2 を変更するためのコマンドです．chmod パーミッション変更シンボル ファイル名 という書式で使います．パーミッション変更シンボルは，「パーミッションをどのように設定するか」を表わす文字列で，「誰に対してのパーミッションを変更するのか (u,g,o あるいは a の記号を用います)」「許可を与えるのか，許可を取り消すのか (+,- あるいは= の記号を用います)」「書きこみ，読みこみ，実行許可のうちのどれに対してか (r,w あるいは x の記号を用います)」を指定する 3 つの部分からなります．表 1.5 に，例を挙げます．

表 1.5: 管理者権限の変更

chmod u+w file1	file1 に対して「オーナーによる書き込み許可」を出す．
chmod g+r file2	file2 に対して「グループによる読み込み許可」を出す．
chmod o-x file3	file3 に対して「他の人による実行許可」を取り消す．
chmod a+rx file4	file4 に対して「すべての人に対する読み込み・実行許可」を出す．
chmod u=rw file5	file5 に対するオーナーの権利は「読み・書きは許可，実行は不許可」とする．

自分に対するアクセス許可がないファイルをアクセスしようとしてもできません．以下の例のように，エラーになります．

```
$ ls -l⏎
total 245
drwxr-xr-x 2 johotaro ugrad         48  1 25 00:45 dir1
-rw-r--r-- 1 johotaro ugrad       7377  1 25 00:44 file1
-rw------- 1 johotaro ugrad     239914  1 25 00:44 file2
$ chmod u-r file1⏎
$ ls -l⏎
total 245
drwxr-xr-x 2 johotaro ugrad         48  1 25 00:45 dir1
--w-r--r-- 1 johotaro ugrad       7377  1 25 00:44 file1
-rw------- 1 johotaro ugrad     239914  1 25 00:44 file2
$ touch file1⏎
file1: Permission denied
$
```

ファイルのパーミッションを，記号でなく数値（8 進数）で直接設定するやり方もあります．これは $r = 4, w = 2, x = 1$ として，オーナー，グループ，他の人の順にその和を並べるやりかたです (表 1.6 参照)．慣れると一度にすべてのパーミッションの設定ができて便利です．

```
$ ls -l⏎
total 245
```

15） CHange MODe の略です．

```
drwxr-xr-x 2 johotaro ugrad          48  1 25 00:45 dir1
-rw-r--r-- 1 johotaro ugrad        7377  1 25 00:44 file1
-rw------- 1 johotaro ugrad      239914  1 25 00:44 file2
$ chmod 666 file1↵
$ ls -l↵
total 245
drwxr-xr-x 2 johotaro ugrad          48  1 25 00:45 dir1
-rw-rw-rw- 1 johotaro ugrad        7377  1 25 00:44 file1
-rw------- 1 johotaro ugrad      239914  1 25 00:44 file2
$
```

file1 に注目すれば，わかりやすいと思います．ここで途中（6 行目）で出てきた「666」という数値について説明を加えておきます．左の数値は所有者の権限を，真ん中の数値はグループの権限を，右の数値はその他（所有者でもグループでもない）の人の権限を表します．数値は 4 が読み出し許可，2 が書き込み許可，1 が実行許可を表し，例えば 6 なら 4+2 で「読み出し許可+書き込み許可」という意味になります．つまり，今回出てきた 666 であれば「所有者にもグループにもその他にも，読み出しと書き込みを許可する」ということになります．数値の詳細に関しては表 1.6 を参照してください．

表 1.6: 管理者権限を表す数値の詳細

数値	内容
4	読み出し許可
2	書き込み許可
1	実行許可
7 (4+2+1)	読み出し許可＋書き込み許可＋実行許可
6 (4+2)	読み出し許可＋書き込み許可
5 (4+1)	読み出し許可＋実行許可
3 (2+1)	書き込み許可＋実行許可

1.8.8 open コマンド

open コマンドとは，ファイル・ディレクトリ・URL などを開くコマンドです．「ファイル」などの GUI 環境における，ファイルのダブルクリックに相当します．

```
$ open file1↵
↑ これでfile1というファイルが開かれた.
$ open .↵
↑ これでカレントディレクトリが「ファイル」
で開かれた
```

1.9 システムに関するコマンド

1.9.1 quota コマンド

自分がどれくらいディスクを使用しているかを調べるコマンドです.

```
$ quota -v↵
Disk quotas for user s1311350 (uid 5621):
   Filesystem  1K blocks    quota    limit  grace  files  quota  limit  grace
        /home    2456883  3072000  3584000         84863      0      0
$ ■
```

この例だと，使用量 (blocks) が 2456883 キロバイト，制限 (quota) が 3072000 キロバイトとなっています.

一般的な Linux 端末では quota コマンドが利用できるのですが，COINS ではシステムの都合で quota コマンドが利用できません．quota の上限・現在の使用量・残り容量などは，Windows にログインして，H: ドライブのプロパティで確認してください．詳細は，https://www.coins.tsukuba.ac.jp/ce/?quota を参照してください.

1.9.2 du コマンド

ディスクの使用量を表示するコマンドです．指定したファイルと，それをルートとする階層中にある全ディレクトリのディスク使用量を表示します．ここでは，カレントディレクトリ下の各ファイル，または，ディレクトリごとのディスク使用量の一覧を表示しています．「-h」によってディスク使用量を人間の読みやすい単位に変換し，「-s」によって各ディレクトリのディスク使用量の総計のみを表示しています.

```
$ du -h -s *↵ ← '*' は後述するワイルドカード
12K     dir1
44M     dir2
200M    dir3
$ ■
```

1.10 画像に関するコマンド

1.10.1 convert コマンド

COINS にはインストールされていませんが，convert コマンドは，画像のサイズや形式を変換するコマンドです．非常に多くの機能を持っていて，ここですべては網羅できないので，ぜひ各自で調べてみてください.

```
$ convert piture1.png picture2.eps↵ ← PNG形式をEPS形式に変換
```

```
$ convert -resize 50% piture1.png picture3.png⏎ ← サイズを縦横ともに50%に縮小
$ convert -resize 512x512 piture1.png picture4.png⏎
↑ 縦横比を保ったまま，512x512のサイズに収める
$ convert -resize 512x512 piture1.png picture5.png⏎
↑ 縦横比を保たずに，512x512のサイズに変換
$
```

1.11 プロセスを扱うコマンド

Unix でプログラムを作ったり作業をしたりしていると，「プログラムが止まらなくなった」「他のウィンドウからプログラムを停止させたい」という状態になることがよくあります．そのようなときに用いるコマンドを説明します．

1.11.1 ps コマンド

ユーザが今どのようなプロセス (プログラム) を動かしているのか調べるためのコマンドです．

```
$ ps⏎
  PID TTY          TIME CMD
 5978 pts/8    00:00:00 bash
29522 pts/8    00:00:00 ps
$
```

ps [-U] [ユーザ名] のようにすると，そのユーザが動かしているプロセスの一覧が表示されます．一番右に表示される文字列が実行されているコマンドの名前で，一番左の数字がそれぞれのプロセスに与えられている**プロセス** *ID* といわれる番号です．UNIX で同時に動いている多くのプログラムは，このプロセス ID という番号で識別されます．`ps a`とした場合，全ユーザが動かしているプロセスの一覧を表示できます．

他にも，`ps l`とすると，プロセスについてのより詳しい情報が得られます．`ps u`とすると，プロセスの実行開始時間を同時に得ることができます．そして，`ps x`とすると，端末の直接の制御下にないプロセスも表示できます．

1.11.2 kill コマンド

主に，プロセスを強制終了させるために使われるコマンドです．`kill` [プロセス ID] と実行すると，そのプロセスを強制終了します．これで終了できないプロセスには， `kill` [-KILL] [プロセス ID] が有効かもしれません．

```
$ ps⏎
  PID TTY          TIME CMD
 5978 pts/1    00:00:00 bash
29621 pts/1    00:00:00 sleep
29522 pts/1    00:00:00 ps
$ kill 29621⏎
```

```
$ ps↵
  PID TTY          TIME CMD
 5978 pts/8    00:00:00 bash
29522 pts/8    00:00:00 ps
$
```

1.11.3 man コマンド

本手引きに書かれている説明はごく簡単かつ基本的なもので，細かいコマンドまでは書かれていません．疑問点などがあったり，使用法を忘れてしまった場合や，そのコマンドの詳しい使い方や仕様を知りたいときのために，man というコマンドが用意されています．これは，

```
$ man ls↵
```

のように用います．これを実行すると，そのコマンドのマニュアルが画面に表示されます．このマニュアルは，表 1.7 のように章が分けてあります． 同じ名前の別のマニュアルが，2 つ以上の章に

表 1.7: 章建て

1 章	コマンド
2 章	システムコール
3 章	ライブラリ関数
4 章	デバイスファイル
5 章	ファイルフォーマット
6 章	ゲーム
7 章	その他
8 章	システム管理コマンド
9 章	カーネルルーチン
10 章	その他 (2)

入っていることがあります．その場合は，man **章番号** **コマンド名** のように，章を指定して，自分の見たいマニュアルを見てください[16]．

また，man -k **キーワード** のようにすると，キーワードに関係のあるマニュアル (manual) の項目名が表示されます．man コマンドのより詳しい使用法は，

```
$ man man↵
```

のように入力すれば，見ることができます．UNIX を使いこなすためには，*man* **を見る習慣をつける**ことが大切です[17]．

16） マニュアル内でよくある，「ls(1)」などのようなコマンド名のあとに括弧内に数字が書かれた表記は，括弧内の数字が章を表わしています (この場合は「1 章の ls」コマンド).

17） 英語で画面が埋め尽されてうんざりすることがあると思いますが，そう難しい英語ではないので頑張って読みましょう．

1.11.4 info コマンド

info コマンドでもコマンドについての説明を見ることができます．一部のコマンドの man コマンドの説明では，info コマンドも参照することを指示される場合があるのでそういった場合に info コマンドを使うと良いでしょう．

```
$ info ls↵
```

のように用います．

1.12 ワイルドカード

ワイルドカード[18]は様々なパターンにマッチし，適宜置換されます．似たような名前のファイルや特定の規則に基づいた名前のファイルを扱うときに便利です．

```
$ ls↵
a.c   b.c   c.c   d.c   a.java   b.java   c.java   d.java   report.tex
$ ls ?.c↵
a.c   b.c   c.c   d.c
$ ls a.*↵
a.c   a.java
$ ls [ab].c↵
a.c   b.c
$ ls [^ab].java↵
c.java   d.java
$ ls [^ab].*↵
c.c   d.c   c.java   d.java
$ ls [c-z]*.[d-z]*↵
c.java  d.java  report.tex
$
```

- *
  0 文字以上の任意の文字列にマッチします．
- ?
  任意の一文字にマッチします．
- [*pat*]
  pat の中のいずれかの 1 文字にマッチします．

18） 正式には “ファイル名の置き換え”，あるいは “パス名の展開” ですが，一般に “ワイルドカード” という名称が浸透しています．

表 1.8: *pat* の例

abc	a，b，c のいずれか
a-c	上記と同じ
a-z	小文字のアルファベット 1 文字
A-Z	大文字のアルファベット 1 文字
0-9	数字 1 文字
a-zA-Z	大小を問わずアルファベット 1 文字
0-9a-zA-Z	すべての半角英数字
^0-9a-zA-Z	すべての半角英数字以外

1.13 tab キーによる補完

入力の途中などに tab キーを押すことで，コマンド名やファイル名などを自動で補完してくれます．

例えば`gatherheaderdoc`というコマンドを実行しようと“gather”まで入力したが，それ以降を覚えていなかった場合，tab キーを押してみます．

```
$ gather[tab]
$ gatherheaderdoc
```

残りの部分が補完されました．

また，ファイル名の補完もできます．`ls b`まで入力して tab キーを押すと，b から始まるファイル名が候補として出力されます．

```
$ ls b[tab]
bin/        bitbucket/ bookmarks/  b.xx
$
```

候補が一意に定まらない場合，候補を出力するだけにとどまります．

1.13.1 カーソル移動

入力位置を示すカーソルは，十字キーで移動することができます．また，ショートカットキーもあるので以下に提示します．

- Ctrl+→ 単語単位で右にカーソルを移動
- Ctrl+← 単語単位で左にカーソルを移動
- Home 行の先頭にカーソルを移動
- End 行の終端にカーソルを移動

1.14　ログインしている人を確認する

ログインしている人を確認するコマンドとして，who コマンドがあります．

```
$ who ⏎
s2011077 tty2         2023-04-05 17:00 (tty2)
s2011077 pts/1        2023-04-05 18:45 (2001:2f8:3a:1712:15d6:18e4:5b52:9c23)
```

これらのうち，1 行目に表示された

```
s2011077 tty2         2023-04-05 17:00 (tty2)
```

は，ユーザ s2011077 が表示された IP アドレスからその端末を利用していることを表します．また，2 行目に表示された

```
s2011077 pts/1        2023-04-05 18:45 (2001:2f8:3a:1712:15d6:18e4:5b52:9c23)
```

は，ユーザ s2011077 が ssh コマンドによるリモートログインで接続していることを表します．

なお，**who コマンドで表示できるのは ssh コマンドでログインしているユーザのみです．ssh コマンド以外でログインしているユーザは表示できない**ので，注意してください．

1.15　コマンドをさらに使いこなす

1.15.1　標準入力，標準出力

通常，シェル上で実行するコマンドの入力はキーボードから行い，ディスプレイに出力されます．これをそれぞれ**標準入力**，**標準出力**といいます．

1.15.1.1　リダイレクト

リダイレクトとは，標準入力先または標準出力先を変更する機能です．記号 > で，標準出力を指定したファイルに切り替えて，指定したプログラムを実行します．記号 < で，標準入力を指定したファイルに切り替えて，指定したプログラムを実行します．

```
$ ls -l > file1⏎
$ cat file1⏎
total 9
drwxr-xr-x  2 johotaro ugrad  4096  2 15 00:10 dir1
-rw-r--r--  1 johotaro ugrad     0  2 17 01:13 file1
-rw-r--r--  1 johotaro ugrad    12  2 15 00:10 file2
$
```

この例では，`ls -l`の実行結果を，画面ではなく file1 に出力しています．

1.15.1.2　パイプ

パイプとは，複数のプログラムを同時に実行したり，プログラムの標準出力を，次に実行するプログラムの標準入力につないだりする機能です．記号 | で複数のコマンドをつないで使います．左側に記述したコマンドが先に実行されます．

```
$ ls -l | head -n 3↵
total 9
drwxr-xr-x  2 johotaro ugrad  4096  2 15 00:10 dir1
-rw-r--r--  1 johotaro ugrad     0  2 17 01:13 file1
$
```

`ls -l`はファイル・ディレクトリの情報を出力するコマンドで，`head -n 3`は入力されたデータのうち，先頭 3 行を表示するコマンドです．`ls -l`の出力先を `head -n 3` の入力先につなぐことにより，ファイル・ディレクトリの情報のうち，先頭 3 行を表示しています．

1.15.2　メタキャラクタのエスケープ

シェルには

< > | $ { } ( ) [ ] & ; ^ " * ? ~ ’ ‘ \ 空白　タブ　#

等のシェルが解釈し，コマンドとして渡されない**メタキャラクタ**という文字が存在します．それらをシェルに解釈させるのではなくコマンドの文字として渡したい場合はエスケープをする必要があります．表 1.9 にエスケープの使い方を示します．

表 1.9: エスケープの使い方

形式	エスケープ対象	エスケープできないメタキャラクタ
\x	「\」の直後の一文字	なし
’str’	「’」でくくられた文字列	「’」
"str"	「"」でくくられた文字列	「$」，「‘」(バッククォート),「"」(ダブルクォート),「\」,「!」

```
$ echo i\'m\ happy ↵ ← ' と（空白）をエスケープする例
i'm happy
$ echo "<><>" ↵ ← < と > をエスケープする例
<><>
$ echo '<><>' ↵ ← ' でもエスケープできる
<><>
$ echo \\ ↵ ← バックスラッシュ自身をエスケープする場合は2つ重ねる
\
$
```

1.15.3　シェル変数と環境変数

シェルでは**シェル変数**と**環境変数**の 2 種類の変数を使用できます．環境変数は子プロセスに変数が引き継がれますが，シェル変数は引き継がれません．

1.15.3.1 シェル変数の代入と参照

シェル変数の定義は 変数名=値 で行います.

```
$ X=100 ⏎ ← X という名前のシェル変数に 100 を代入する
$ echo $X ⏎ ← 値を参照したい場合は$変数名のようにする.
100
$ ■
```

1.15.3.2 環境変数の代入と参照

環境変数の定義は export 変数名=値 で行います. あらかじめ定義したシェル変数を export シェル変数 で環境変数に変えることもできます.

```
$ export Y=200 ⏎ ← X という名前の変数に 200 を代入する
$ echo $Y ⏎ ← 値を参照したい場合は$変数名のようにする
200
```

```
$ Z=300 ⏎
$ bash ⏎ ← 子プロセスとして bash を起動する
$ echo $Z ⏎ ← シェル変数である Z
    は子プロセスに引き継がれないため何も表示されない

$ exit ⏎ ← 子プロセスとして起動した bash を終了する
$ export Z ⏎ ← シェル変数を作ってから環境変数に変えることも可能
$ bash ⏎ ← 子プロセスとして bash を起動する
$ echo $Z ⏎ ← Z を環境変数にしたので子プロセスに引き継がれた
300
$ ■
```

1.15.3.3 変数の削除

変数の削除は環境変数, シェル変数共に unset 変数名 で行います.

```
$ X=100 ⏎
$ echo $X ⏎
100
$ unset X ⏎ ← X という名前のシェル変数を削除
$ echo $X ⏎ ← X という変数は削除されたため何も表示されない
```

```
$ export Y=200 ⏎
$ echo $Y ⏎
200
$ unset Y ⏎ ← Y という名前の環境変数を削除
$ echo $Y ⏎ ← Y という変数は削除されたため何も表示されない

$ ■
```

1.15.3.4　設定されている変数の確認

シェル変数を確認したい場合は set コマンドを，環境変数を確認したい場合は export コマンドをそれぞれ引数なしで入力します．シェル変数や環境変数にはあらかじめ設定されているものがあり，現在の環境の状態や設定を保持しています．

1.15.4　ディレクトリスタック

ディレクトリを移動する方法には cd コマンドのようにパスを直接指定する方法の他に，**ディレクトリスタック**を用いる方法があります．ディレクトリスタックはディレクトリのパスを後入れ先出しの形式で記憶してくれます．これにより，遠いディレクトリ間の移動が楽に行えます．

ディレクトリスタックの一番先頭にはカレントディレクトリのパスが入っています．後入れ先出しで記録するため，ディレクトリを取り出すときは最後に入れたディレクトリパスが最初に取り出されます．

ディレクトリスタックを操作するコマンドには pushd と popd があります．pushd コマンドは引数に渡したディレクトリパスをスタックに入れて，そのディレクトリに移動するコマンドです．popd コマンドはディレクトリスタックの先頭のディレクトリパスを取り出して，次のディレクトリに移動する命令です．dirs コマンドを用いるとディレクトリスタックの現在の状態を確認できます．

以下にディレクトリスタックの使い方の例を示します．

```
$ dirs ⏎ ← 現在のディレクトリスタックの状態を確認する
~
$ pushd /proc ⏎ ← /proc をディレクトリスタックに入れて /proc に移動
/proc ~
$ pushd /var/log ⏎ ← /var/log をディレクトリスタックに入れて /var/log に移動
/var/log /proc ~
$ pushd /usr/bin ⏎ ← /usr/bin をディレクトリスタックに入れて /usr/bin に移動
/usr/bin /var/log /proc ~
$ popd ⏎ ← ディレクトリスタックの一番先頭のディレクトリパスを取り出して移動
/var/log /proc ~
$ pwd ⏎ ← ディレクトリスタックの一番先頭が /var/log
    になったのでカレントディレクトリ
/var/log    も /var/log に変わった
$ popd ⏎ ← ディレクトリスタックの一番先頭のディレクトリパスを取り出して移動
/proc ~
$ pwd ⏎ ← ディレクトリスタックの一番先頭が /proc
    になったのでカレントディレクトリも /proc に変わった
/proc
$
```

また，～数字 とするとディレクトリスタック内のディレクトリパスを表すことができます．

```
$ pushd /var/log ⏎
/var/log /proc ~
$ pushd /usr/bin ⏎
```

```
/usr/bin /var/log /proc ~
$ dirs -p -v ⏎ ← dirs コマンドに -p -v
    とオプションを渡すと見やすい形式で出力される
 0 /usr/bin
 1 /var/log
 2 /proc
 3 ~
$ cd ~2 ⏎ ← ディレクトリスタックの（0から数えて）2番めに移動
$ pwd ⏎
/proc
$ dirs ⏎ ← cd
    コマンドで移動したためカレントディレクトリのパスを示す先頭以外は変わらない
 0 /proc
 1 /var/log
 2 /proc
 3 ~
$
```

1.16 その他有用なコマンド

1.16.1 alias コマンド

このコマンドを用いると，良く使うコマンドやオプションに別な名前を付けることが出来ます．alias 自分で付けたい名前 = 'コマンド列' で定義します．

```
$ ls⏎
dir1  file1  file2
$ alias lls='ls -l -F'⏎
$ lls⏎
total 245
drwxr-xr-x  2 johotaro ugrad        48  1 25 00:45 dir1
-rw-rw-rw-  1 johotaro ugrad      7377  1 25 00:44 file1
-rw-------  1 johotaro ugrad    239914  1 25 00:44 file2
$
```

決まったオプションでしか使わないコマンドや名前の長いコマンドなどを alias で別名にすることで，使いやすくなります．ただし，ターミナル上で設定したエイリアスは，1 度ターミナルを閉じると消えてしまいます．

より高度な技術を身につけたい人は，シェルとは何か，また，.bashrc や.zshrc とはどのようなファイルかを学ぶと，ターミナルを開くたびにこのエイリアスをロードするようにできるでしょう．このように，いくつかのヒントを元に，自分の環境をカスタマイズしていくと，少しずつ上達していきます．たとえば，ホームディレクトリにある.bashrc というファイルにコマンドを書き加えておくと，そのコマンドはターミナルでウィンドウを開くたびに実行されます，といったことから，自分で環境をカスタマイズするヒントを得られるようになるでしょう．

以下に，便利な alias の例を示します．

```
alias ll='ls -l'
alias lla='ls -al'
alias lat='ls -tal'
alias lt='ls -tl'
alias la='ls -A'
alias l='ls -CF'
alias u='cd ..'
alias uu='cd ../../'
alias uuu='cd ../../../'
```

1.16.2 nkf コマンド

文字コードと改行コードを変換するコマンドです．Windows などと相互にファイルのやりとりをすると，時々 UNIX から持っていったテキストファイルが読めないことがあります．または自分で作った Web ページが文字化けしてしまって正しく表示されなかったりすることがあります．そういうときにテキストファイルの文字コード・改行コードを変換してくれるのが nkf コマンドです．

書式は nkf [オプション] [変換前ファイル名] > [出力先ファイル名][19]となります．オプションは，出力したい文字コードによって-e（日本語 EUC），-j（JIS コード），-s（シフト JIS コード），-w（UTF-8）のどれかから選び，改行コードも-Lu (UNIX)，-Lw (Windows)，-Lm (Mac OS 9 以前の Macintosh) の中から選びます．以下は，Windows で作成したテキストファイルを UTF-8 に変換する例です[20]．

```
$ ls↩
file1
$ locale↩
LANG="ja_JP.UTF-8"
LC_COLLATE="ja_JP.UTF-8"
LC_CTYPE="ja_JP.UTF-8"
LC_MESSAGES="ja_JP.UTF-8"
LC_MONETARY="ja_JP.UTF-8"
LC_NUMERIC="ja_JP.UTF-8"
LC_TIME="ja_JP.UTF-8"
LC_ALL=
$ cat file1↩
śńォ çÁĽŕ¡eťğÁyĽçhã ☒☒ ć ☒☒ òð] ☒☒ B
u ☒ A ☒ v
@ewşţłçãðUèÁAŷÁãê ☒ éðàĽţłÁ¡BA ¿ćłç
u ¡Á¡çAéN 愚 éñůàAćŕ ☒ źñív
@ś¿tv¿ ☒ è
użêţàĺOxćśÁ¡çv
```

19） この 2 つのファイル名を同じにしてしまうとそのファイルが消えてしまうので注意してください．

20） 岡本かの子著：岡本かの子全集 5 ちくま文庫 筑摩書房 1993 年「快走」 http://www.aozora.gr.jp/cards/000076/card50618.html

```
@ğðćÅ

$ nkf -w file1 > file2↵
$ ls↵
file1 file2
$ cat file2 | fold -78↵
このとき後から追っかけて来た父親は草原の中に立って遥
かに堤防の上を白い塊が飛ぶのを望んだ。
「あれだ、あれだ」
  父親は指さしながら後を振り返って、ずっと後れて駈け
て来る妻をもどかしがった。妻は、はあはあ言いながら
「あなたったら、まるで青年のように走るんですもの、追
いつけやしませんわ」
  妻のこの言葉に夫は得意になり
「それにしてもお前の遅いことったら」
  妻は息をついで
「これでも一生懸命だもんで、家からここまで一度も休ま
ずに駈けて来たんですからね」
「俺達は案外まだ若いんだね」
「おほほほほほほほほほ」
「あはははははははは」
二人は月光の下を寒風を切って走ったことが近来にない喜
びだった。
二人は娘のことも忘れて、声を立てて笑い合った。
$ ■
```

ただし，fold -78 は，半角 78 文字分のところで折り返すためのコマンドです．

基本的に UNIX では自動的に文字コードは判別してくれるので，普段あまり用は無いかもしれませんが，他のシステムとファイルのやりとりをする場合は注意が必要です．

1.16.3 xclock，xcalc コマンド

それぞれ時計[21)]，電卓です．

21） 時々狂っている時があるかもしれませんがその時は管理者まで連絡をお願いします．

第 2 章　VSCode

VSCode(Visual Studio Code) とは，Microsoft 社が提供しているコードエディタです．ここでは，Windows を用いて説明します．拡張機能が豊富で，様々な拡張子[1)]のファイルに対応しています．

2.1　インストールの方法

使うためにまず VSCode をインストールしましょう．次のインストールページ（図 2.1）から自分が使用しているパソコンに入っている OS のものをインストールします．

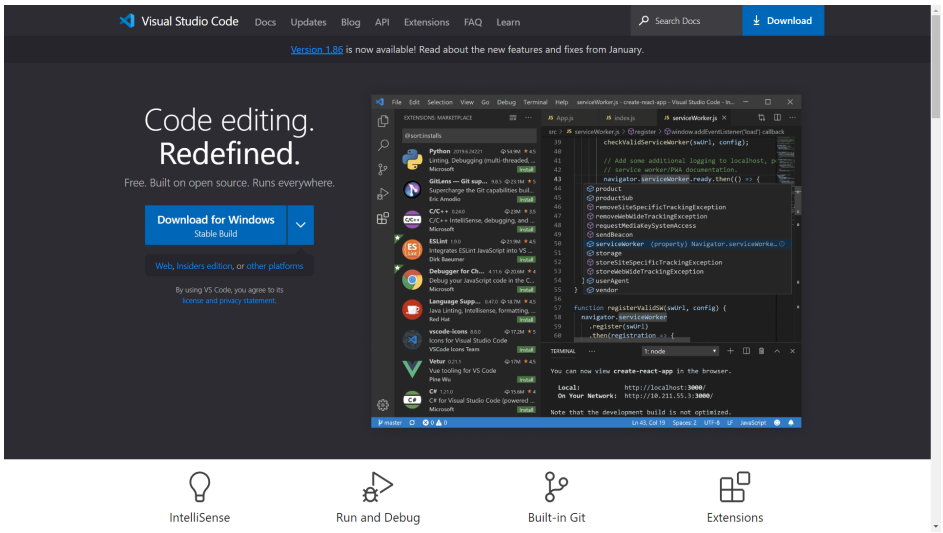


図 2.1: インストールページ

インストールができたら，インストーラーを起動します．すると次の画面が出てくるので，「同意する」を選択して，「次へ」をクリックします．

1)　1.3 節参照

図 2.2: 同意画面

「デスクトップ上にアイコンを作成する」に追加でチェックを入れます．（図 2.4）

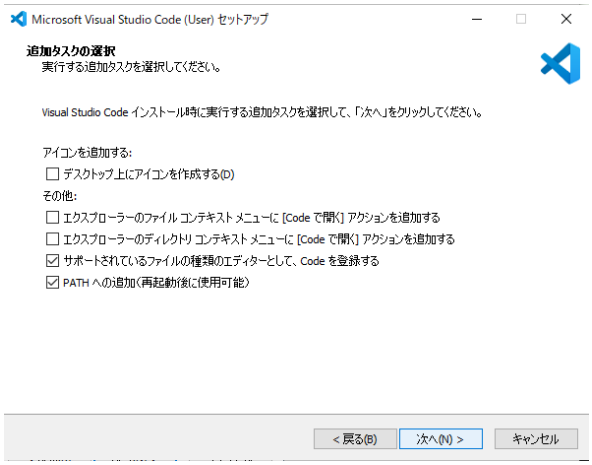


図 2.3: 追加タスクの選択①

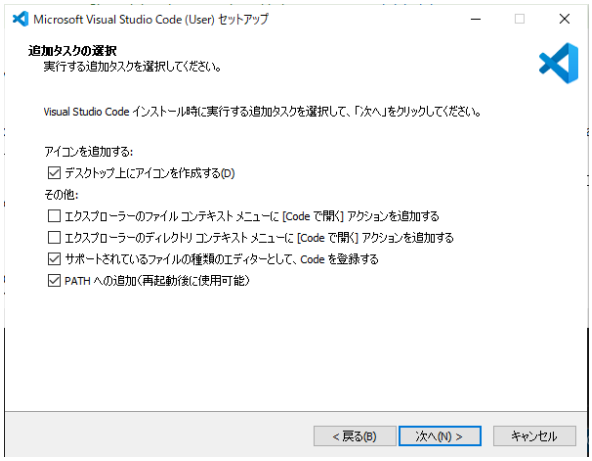


図 2.4: 追加タスクの選択②

「次へ」をクリックします.

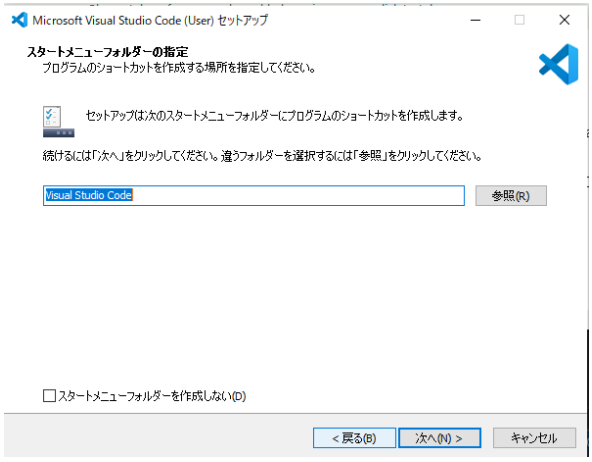


図 2.5: スタートメニューフォルダーの推定

インストール先のフォルダ等に何も問題がなければ「インストール」をクリックします.

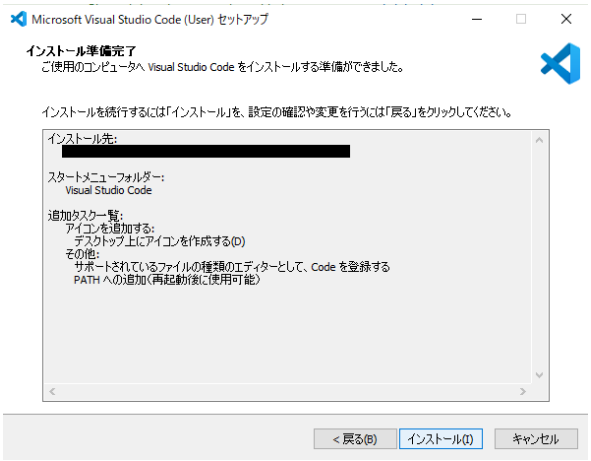


図 2.6: インストールの準備完了

インストールが終わると，自動で VSCode が起動します．

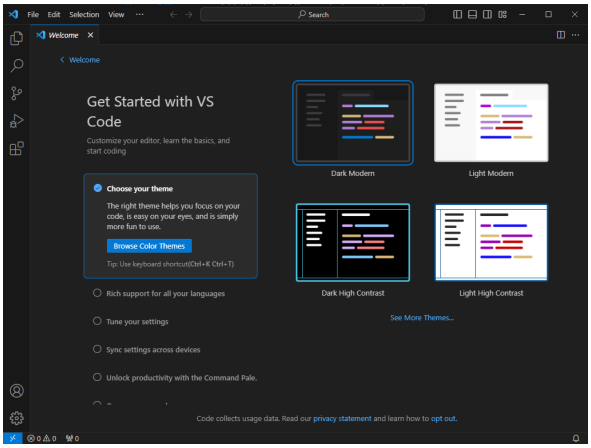


図 2.7: VSCode 起動画面

VSCode には表示言語を日本語にする拡張機能があるので，インストールします．以下，インストールの手順です．

1.「拡張機能」のアイコンをクリック
2. 左上の検索欄で「japanese」と入力
3.「Japanese Language Pack for Visual Studio Code」（図 2.8）をインストール
4. 右下にポップアップが出てくる（図 2.9）ので「Change Language and Restart」をクリック
5. 自動で再起動され，起動後は日本語になっている（図 2.10）

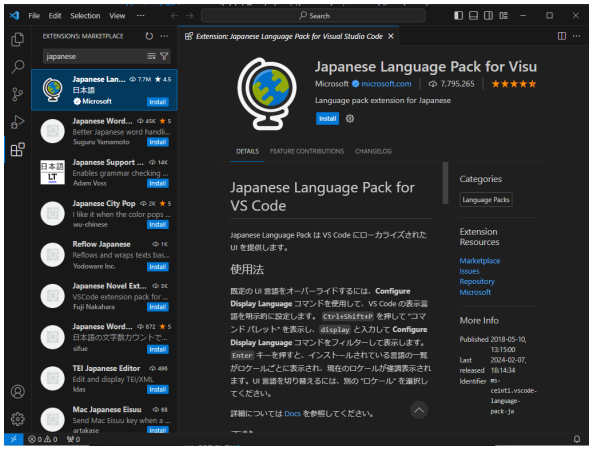


図 2.8: Japanese Language Pack のインストール画面

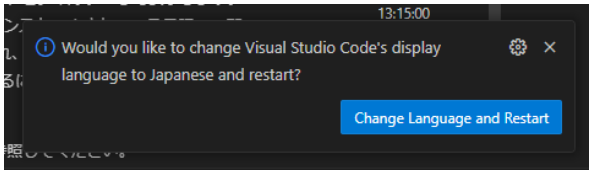


図 2.9: 言語変更と再起動確認のポップアップ

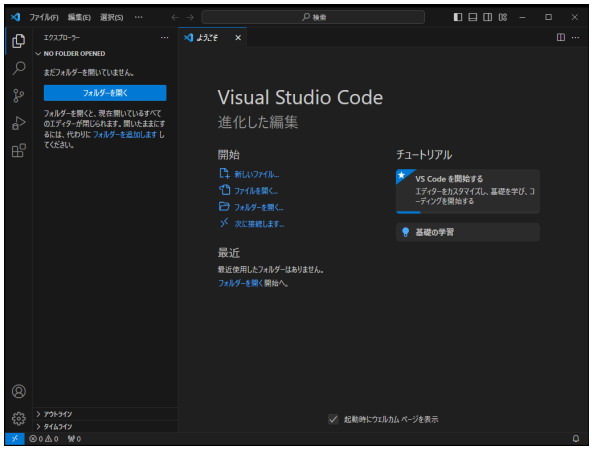


図 2.10: 再起動後の画面

2.2 VSCode の画面

図 2.11 が VSCode の実際の画面です．VSCode の画面は 5 つの画面から構成されています．

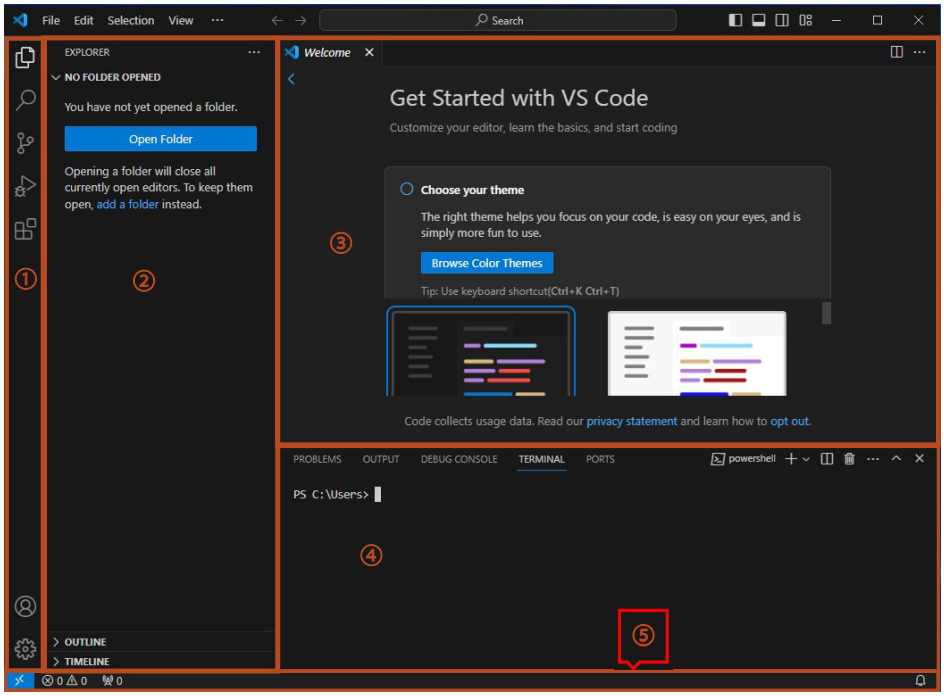


図 2.11: 実際の画面

2.2.1 ①：アクティビティバー

VSCode で主要な機能をアイコンから選択します．拡張機能を入れるとここにアイコンが出るものもあります．ここでは図 2.11 にあるものを上から順番に説明していきます．

1. エクスプローラー

   ファイルが 2 つ重なったような見た目のアイコンです．VSCode 上で開いているフォルダ・ファイルを一覧で表示してくれます．
2. 検索

   虫眼鏡のような見た目のアイコンです．ファイル内の任意の文字・文字列を検索してくれます．
3. git

   git のアイコンです．git と連携することができます．git に関してここで詳しくは扱いません．
4. デバッグ

   右向き三角に小さな虫がついているようなアイコンです．デバッグを行うことができます．
5. 拡張機能

   ブロックがつ組みあがっているようなアイコンです．VSCode の様々な拡張機能を検索・導入することができます．
6. アカウント

丸の中に人がいるようなアイコンです．アカウントの設定を指定できます．

7. 設定

歯車のようなアイコンです．VSCode に関する様々な設定ができます．

2.2.2 ②：サイドバー

アクティビティバーで選択しているアイコンによって表示が変わります．

- エクスプローラー

  ディレクトリ構造が表示されます．
- 検索

  検索結果が表示されます．
- git

  コミットしたりプルリクエストを送ったりすることができます．ターミナルでやる方が確実ですが，慣れないうちはこちらが便利です．
- デバッグ

  実行ファイルの詳しい情報を見ることができます．
- 拡張機能

  拡張機能の一覧を表示してくれます．

2.2.3 ③：エディタ

上部にタブのような形で開いているファイルが表示されます．開いているファイルの中身を表示してくれます．書いてあるコードをここで直接編集できます．ファイル名の横に白い丸がある場合，ファイルの変更内容が保存されていないことを示しています．保存するのを忘れないようにしましょう．

2.2.4 ④：パネル

ターミナルやデバッグ情報が表示されます．

2.2.5 ⑤：ステータスバー

文字コードや言語など，ファイルのステータスに関する情報を表示します．

2.3 ファイルの編集方法

VSCode でファイルを編集するためには，まず VSCode 上でフォルダを選択する必要があります．サイドバーで「フォルダーを開く (Open Folder)」をクリックして，開きたい任意のファイルを選択します．フォルダが開かれたら，サイドバーにディレクトリが表示されます．選択したフォ

ルダ名の右側にあるアイコンからファイルやフォルダを追加することができます.

図 2.12: ファイルやフォルダを追加するアイコン

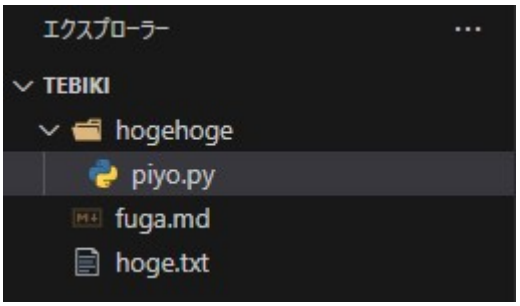


図 2.13: ファイルを追加した様子

2.3.1 ファイル編集のための操作コマンド

VSCode は GUI (Graphical User Interface) サポートが豊富なエディタですが，操作コマンドを駆使することによってファイル編集の効率が大幅に向上します．表 2.1 の操作コマンドは是非身につけましょう．ここでは，Windows 版のコマンドを説明します (Mac の場合は ctrl ではなく，⌘ を使用します).

表 2.1: ファイル編集のための操作コマンド (Windows 版)

編集系（コピー/切り取り/削除/保存等) コマンド	
コマンド	機能
ctrl + c	コピー
ctrl + x	切り取り
ctrl + v	ペースト
ctrl + shift + k	カーソルが当たっている行または選択行を削除
ctrl + s	ファイルの保存
移動系コマンド	
コマンド	機能
home / end	行の先頭/末尾にカーソルを移動
ctrl + → / ←	単語単位で左右にカーソルを移動
ctrl + home / end	ファイル先頭/末尾にカーソルを移動
alt + ↑ / ↓	選択行を上下に移動
alt + → / ←	前へ戻る/次へ進む
開く/閉じる系コマンド	
コマンド	機能
ctrl + w	タブを閉じる
ctrl + shift + t	閉じたタブを再び開く
ctrl + shift + w	VSCode を閉じる

2.4　VSCode でプログラムを実行する方法

VSCode で作成したプログラムは，VSCode から実行させることができます．ここでは，2.3 節の内容にしたがって作成した Python プログラムを実行する方法について説明します．

一番簡単な方法は，エディタ領域右上にある図 2.14 の実行ボタンをクリックすることです．そうすると，図 2.15 に示すように VSCode のパネル領域にターミナルが開き，そこで現在エディタ領域に表示されているファイルが Python インタプリターを使って実行されます．

これ以外の実行方法や Python 以外の実行方法については，公式ドキュメント[2)]を参照しましょう．

図 2.14: 実行ボタン

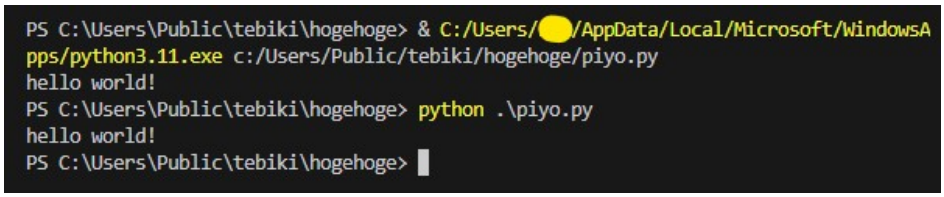


図 2.15: 実行結果

2.5　その他の機能

2.5.1　画面分割

VSCode では画面を分割し, 複数ファイルを同時に閲覧・編集することができます. タブのところで右クリックをすると, 以下の画面がでてきます.

ここから **split** から始まるものをクリックすると, その方向に画面が分割されます.

2.5.2　ブレイクポイント

ブレイクポイントはその行でプログラムの実行を一時中断できる機能です．デバッグの際によく用います.

下の画像で赤い丸がある行にブレイクポイントが設定されています. 設定の仕方は, 行番号の左側にカーソルをかざすと赤丸が出てくるので, そこでクリックしてください.

2)　https://code.visualstudio.com/docs/languages/overview

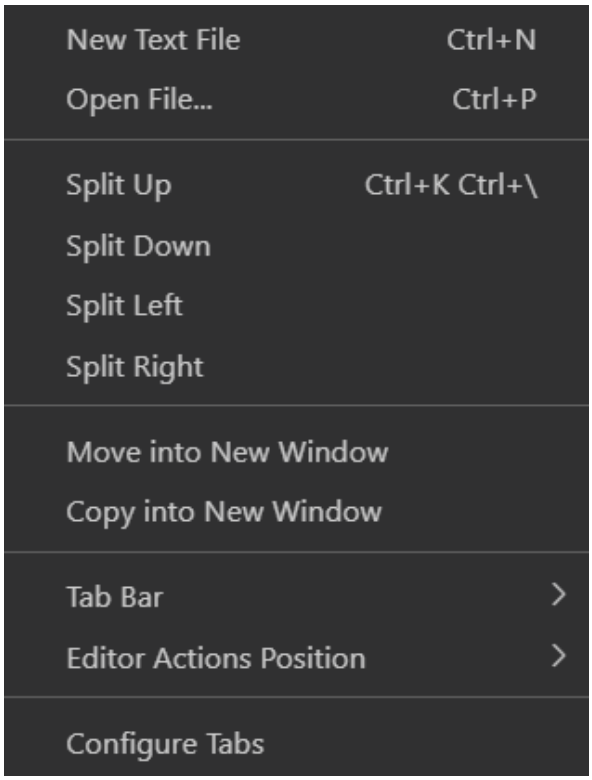


図 2.16: 画面分割の選択画面

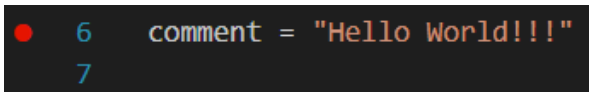


図 2.17: ブレイクポイント

第 3 章　LaTeX

本章では，組版処理ソフトウェアである LaTeX について紹介します．この章の目的は簡単なレポートから卒業論文まで書けるような LaTeX の記法を身につけることです．

3.1　LaTeX の概要

LaTeX[1)]は本やレポート，論文を作成するためのツールです．Microsoft の Word や Just System の一太郎などと同じような用途のソフトウェアです．最初は通常のワープロソフトに比べ，覚えることも多く面倒に感じるかもしれませんが，慣れてしまえば非常に有用なツールです．

LaTeX の特徴としては例えば次のようなものが挙げられます．

マークアップ言語である　LaTeX はマークアップ言語[2)]の一つです．この方式によって文書の体裁を論理的に記述できます．

数式や図を綺麗に出力できる　MS Word にも数式エディタが存在しますが，それよりも格段に綺麗な，かつ表現力の高い数式を出力できます．

無償で利用できる　LaTeX およびその関連ツールは無償で使うことができます．

複数の OS で動作する　Windows や macOS，Linux 系など様々な OS の上で動作します．

例えばこの手引きも LaTeX によって作成されています．

3.1.1　TeX と LaTeX

LaTeX を学習するにあたって，同時に TeX[3)] という言葉についても学習する必要があります．この 2 つの意味を正しくおさえておくことは，今後 TeX や LaTeX について学習したり Web などで情報を集めたりする際に役に立つでしょう．なお，LaTeX を利用して pdf ファイルを作成する具体的な方法は 3.4 節から説明します．まずは実際に LaTeX を使ってみたい，という方はここから取り組んでみてください．

3.1.1.1　TeX とは

TeX とは Donald Knuth という人が，1978 年に作成したプログラム言語です．プログラム言語ですから，Java や C 言語と同様に**処理系**と呼ばれるプログラムがあり，ソースコードから処理系を

1)　「らてふ」や「らてっく」と発音します．どちらで発音しても構いません．その他にも「れいてっく」などと発音することもあります．

2)　タイトルや見出しといった文書の構造と，フォントなどといった見た目の情報を文書の中に埋め込むという，文書を記述する方法の一つです．

3)　「てふ」もしくは「てっく」と発音します．

用いて何らかの成果物を得ます．先ほどの図 3.2 にあるように，TEX ファイル（ソースコード）は，ε - pTEX などの TEX の処理系を用いて DVI ファイルへと変換します．これは Java のソースコードを javac といった処理系を用いて，実行ファイルを得る関係と似ています．

TEX の処理系は Knuth が作ったもの以外にもたくさんあり，日本では主に ε - pTEX や ε-upTEX という処理系が使われています[4]．

つまり TEX とは厳密に言うならば，Knuth が作ったプログラム言語かあるいはその処理系を指すということになります．

3.1.1.2 LATEX とは

TEX が Knuth が作ったプログラム言語やその処理系を指すことに対して，LATEX とはもともとプログラム言語 TEX で書かれた**プログラム**のことです．1980 年に Leslie Lamport という人によって作られました．

LATEX は TEX をより便利に使えるようにするプログラムで，TEX の世界ではこれらを**マクロパッケージ**[5]と呼びます．Java など汎用言語の世界ではこのようなプログラムのことを**ライブラリ**と呼ぶことがありますが，その関係と似ています．

この TEX と LATEX の関係を図 3.1 にまとめました[6]．

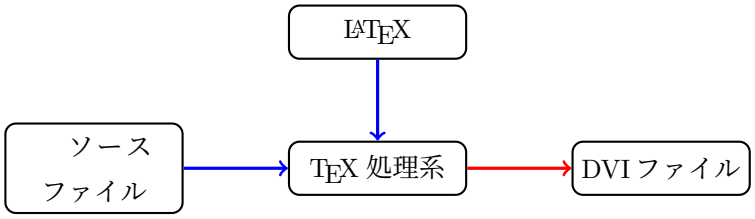


図 3.1: TEX 処理系とマクロパッケージ LATEX の関係

まず皆さんは VSCode などのテキストエディタを用いて文書を LATEX ファイル（ソースファイル）へ内容を書き込み，それを ε - pTEX など TEX の処理系に渡します．ε - pTEX など TEX の処理系はコンピュータの中から自動的に LATEX を探し出し，これを読み込んで最終的な文書を生成します[7]．

現在 LATEX は世界中で使われるようになり，少なくとも日本語圏においては多くの文書が LATEX を前提として作成されるようになったので，いつしか “LATEX” が “LATEX パッケージを前提とした TEX” という意味合いで用いられるようになりました．ですが厳密に言えば LATEX は TEX で実装されたプログラムです．

---

4) 有名な処理系については図 3.9 にまとめてあります．
5) あるいは “マクロ” や “パッケージ” と省略することもあります．
6) 正確に言えば，私たちが通常 LATEX を用いる際は LATEX パッケージを TEX 処理系によって一度コンパイルして得られた**フォーマットファイル**を用います．
7) 脚注 6)にあるように，正確には LATEX パッケージをコンパイルして得られるフォーマットファイルを読み込むことになります．また LATEX を読み込まずに TEX 処理系を使うこともできますが，この章では解説しません．

また，これに伴って，“TEX” という言葉が “LATEX を使っていない純粋な TEX” という意味に解釈されることがあります[8]．ただ，皆さんがマクロパッケージとしての LATEX を話題にすることは少ないでしょうから，この章では LATEX という言葉を，“LATEX パッケージを前提とした TEX” という意味で用います．

LaTeX2.09 と LaTeX2e　LATEX が TEX で実装されたマクロパッケージ（プログラム）であるということを前節では述べました．プログラムは “Windows 10” や “macOS 10.14 Mojave” といった OS のようにバージョンを持ちます．LATEX も同様にバージョンがあり，現在は LATEX 2$_\varepsilon$ というバージョンの LATEX が使われています．

LATEX 2$_\varepsilon$ が登場したのは 1993 年で多くの時間が経過しているので，現在 LATEX といった場合は LATEX 2$_\varepsilon$ のことを指します．この LATEX 2$_\varepsilon$ より前の LATEX のことを “LATEX2.09” と区別します．

3.1.2　ε - pTEX と dvipdfmx

TEX ファイルから最終的な PDF ファイルを得るためには，図 3.2 で示したように次の 2 つのプログラムを用いる必要があります．

ε - pTEX　LATEX ファイルを DVI ファイルへコンパイルするプログラム
dvipdfmx　DVI ファイルを PDF ファイルへコンパイルするプログラム

3.1.2.1　ε - pTEX

Knuth が作った TEX を改造して，日本語や縦書きができるようにした pTEX と，さらに NTS team によって拡張された ε-TEX をマージした処理系が ε - pTEX となります．Unicode に対応していない[9]という欠点はありますが，Web の資料や LATEX に関する日本語書籍の多くが ε - pTEX を前提としていることが多いので，本章では ε - pTEX を採用します．

情報科学類の計算機環境にて ε - pTEX を用いる場合は，Ubuntu のターミナルにて `platex` というコマンドを実行します[10]．すると次のような表示が得られるはずです．

```
$ platex↵
This is e-pTeX, Version 3.1415926-p3.4-110825-2.6 (utf8.euc) (TeX Live 2013)
 restricted \write18 enabled.
**
```

このコマンドによって，ε - pTEX（`e-pTeX`）が起動します．起動が確認できたら，ctrl+c でひとまず ε - pTEX を終了します．

もし `pTeX` という表示が出た場合，それは ε - pTEX ではなく pTEX という別の処理系です．恐ら

8）この問題を解決するために，LATEX などを用いていない純粋な TEX のことを “plain TEX” と呼び，処理系の話題と区別します．
9）文字コードが UTF-8 のファイルを扱うことはできますが，これは内部で `nkf` などを用いて一旦 EUC などの文字コードへ変換しているだけです．
10）情報科学類の計算機にインストールされた他の OS や，大規模計算機には TEX Live がインストールされていませんので，情報科学類の計算機を用いる場合には Ubuntu を使うのがよいでしょう．

く使用しているコンピュータにインストールされた LaTeX の環境が古いのだと思われます．3.24.1 節を参考するなどして最新の LaTeX 環境へアップデートしましょう．

3.1.2.2 dvipdfmx

dvipdfmx は DVI ファイルを PDF ファイルへコンパイルするプログラムです．かつては dvipdfm というプログラムが使われていましたが，それを改造して現在は dvipdfmx が TeX Live に収録されています．

現在はパッケージの相性という観点から dvipdfm の使用は非推奨ですし，TeX Live にも同梱されていません[11)]．

3.2 LaTeX による文書生成の流れ

LaTeX は後述する LaTeX ファイルというテキストベースのファイルから，DVI ファイルという形式のファイルを経て，PDF ファイルなどの最終的な文書へ変換されます．これは次の図 3.2 のようになります．

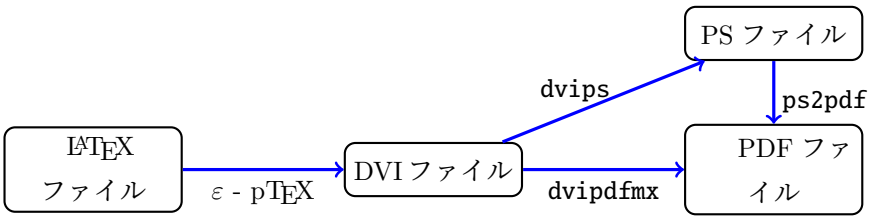


図 3.2: LaTeX ファイルから最終的な文書生成の流れ

従って，LaTeX ファイルの内容編集は VSCode（第 2 章参照）などといったテキストエディタで行い，それを ε - pTeX[12)] などのプログラムを用いて，後述する DVI ファイルへ変換し，さらにそれを dvipdfmx といったプログラムにて PDF へと変換することで，最終的な PDF ファイルの文書が得られます．

この変換のことを**コンパイル**と呼びます．つまり LaTeX のソースファイルをコンパイルすることで DVI ファイルが得られ，さらに DVI ファイルをコンパイルすることで PDF ファイルなどが得られます．

まずは各種ファイル形式について説明します．

11） dvipdfm コマンドを実行すると dvipdfmx が互換モードで起動します．
12） これは「いーぴーてっく」もしくは「いーぴーてふ」と発音します．

3.3 LaTeX に関連するファイルの形式

前節では “LaTeX ファイル” や “DVI ファイル” といった聞き慣れないファイル形式を経て最終的な文書になるという説明をしました．この節では LaTeX に関連するファイル形式を列挙して解説します．

LaTeX ファイル
: 文書の元となるテキストファイルです．これに文書や LaTeX の命令を書き込みます．詳しい文法などは後述します．

DVI ファイル
: DVI ファイルの DVI とは “**D**evice-**I**ndependent” のことで，文書のレイアウトが表示デバイスに全く依存しない形のバイナリデータです．LaTeX ファイルをコンパイルすることで生成されます．この DVI ファイルを直接閲覧・編集するということはあまりなく，別のファイルへ変換するための中間ファイルという役割が一般的です．

3.4 Hello LaTeX!

ソースコード 3.1 と同じ内容のテキストファイルを `test.tex` の名前で作成してください．詳細については後述しますので，今はとりあえず無視して転写してください．

Listing 3.1: Hello LaTeX!

```
\documentclass[a4j, dvipdfmx]{jsarticle}

\begin{document}

% この行はコメントです.
% 表示に影響を与えません.
\section{Hello \LaTeX!}
\LaTeX の文書は次のプログラムでコンパイルします.

\begin{itemize}
  \item platex
  \item dvipdfmx
\end{itemize}

\subsection{コンパイル}
以下のようにターミナルでコンパイルします.

\begin{verbatim}
platex -kanji=utf8 test.tex
dvipdfmx test.dvi
```

```
21 \end{verbatim}
22
23 \end{document}
```

この test.tex に対して以下のようにコンパイルを行います[13].

```
$ platex -kanji=utf8 test.tex↵
$ dvipdfmx test↵
$
```

このコマンドにより，LaTeX ファイル 3.1 を ε - pTeX と dvipdfmx でコンパイルした結果，様々なファイルと共に test.pdf という PDF ファイルが生成され，それは図 3.3 のようになります.

1 Hello LaTeX!

LaTeX の文書は次のプログラムでコンパイルします.

- platex
- dvipdfmx

1.1 コンパイル

以下のようにターミナルでコンパイルします.

```
platex -kanji=utf8 test.tex
dvipdfmx test.dvi
```

図 3.3: test.pdf

LaTeX で文書を作る最初の一歩を踏み出しました．もしコンパイルが上手くいかない場合は次の節を参考にしてください.

3.4.1 コンパイルに失敗したとき

Java の処理系（javac）へ文法的に誤ったプログラムを渡したときのように，エラーが発生してコンパイルに失敗することがあります．例えば，次のような状態で TeX が止ってしまうことがあります.

```
! Undefined control sequence.
l.362 \LATEX

?
```

13） このオプション-kanji=utf8 は LaTeX ファイルの文字コードを指定しています．ファイルの文字コードが EUC-JP の場合は euc，Shift-JIS の場合は sjis と指定します.

このような状態になった場合は，ひとまず x と入力してから ↵（return）キーで TEX を終了します．

```
! Undefined control sequence.
l.362 \LATEX

? x↵
```

TEX のエラーは!から始まります．ですのでこの場合は `! Undefined control sequence.`がエラーということになります．また次の行にある `l.362 \LATEX`とは，“362 行目の `\LATEX`の付近でエラーが発生した” ということを示しています．

LATEX のエラーコードはたくさんあります．もし皆さんがエラーと遭遇してしまった場合は TEX Wiki[14] にエラーメッセージの意味が多くまとめられていますので，こちらを参考に間違いを修正するとよいでしょう．

3.5 LATEX のコマンドと環境

前節ではサンプルを基に LATEX ファイルを実際にコンパイルして PDF ファイルを作成しました．本節では実際に LATEX ファイルを記述するための記法を紹介します．

3.5.1 コマンド

ソースコード 3.1 には \（バックスラッシュ）から始まる，例えば`\documentclass`や`\section`といったものがあります．これらのことを LATEX では次のように言います．

- コマンド
- マクロ
- コントロールシークエンス（Control Sequence, CS）
- 制御 綴（せいぎょつづり）

このうちどれを使っても間違いではありません．この章では \ から始まるものを，**コマンド**という用語で統一します．

このコマンドとは TEX 処理系へ組版に関する指示を与える命令のことです．コマンドの意味を理解することにより，LATEX で綺麗な文書を作ることができるようになります．

3.5.1.1 特殊文字

LATEX では一部の ASCII 文字を**特殊文字**として扱います．それらの文字を下に示します．

14） https://texwiki.texjp.org

特殊文字の一覧

$ % & _ { } \^~

これらの特殊文字は，そのまま書いても表示されず，命令として解釈されます．ただし，これらと同じ形をしている全角の文字は，特殊文字としては扱われません．特殊文字を文章に入れたい場合は，表 3.1 に示す方法でエスケープする必要があります．

表 3.1: 特殊文字のエスケープ

文字	コマンド	文字	コマンド	文字	コマンド
#	\#	$	\$	%	\%
&	\&	_	_	{	\{
}	\}	\	\textbackslash	^	\textasciicircum
~	\textasciitilde				

3.5.1.2 記号

TEX や LATEX には，記号を出力するコマンドがいくつかあらかじめ用意されています．一部を表 3.2 に示します．

表 3.2: 記号の一例

コマンド	出力結果	コマンド	出力結果	コマンド	出力結果
\dag	†	\ddag	‡	\copyright	©
\pounds	£	\oe	œ	\OE	Œ
\ae	æ	\AE	Æ	\aa	å
\AA	Å	\o	ø	\O	Ø
?‘	¿	!‘	¡	$\sim$	∼
‘‘	“	’’	”	--	–
\LaTeX	LATEX	\TeX	TEX	---	—
\textasciicircum	^	\textbar	|	\textless	<
\textgreater	>	\textregistered	®	\texttrademark	™
\textvisiblespace	␣	\textbackslash	\	\textasciitilde	~
\S	§	\P	¶		

3.5.1.3 引数

LaTeX のコマンドには**引数**といって，そのコマンドに何か情報を渡す必要がある場合があります．このコマンドに渡される情報を引数と呼び，`[]`や `{}`で囲んでコマンドの後に記述します．このうち，`[]`を使う引数を特に**オプション**と言い，こちらは書いても書かなくてもよいです．

例えば `\documentclass`コマンドは a4j, dvipdfm と jsarticle という 3 つの引数を受け取っていて，そのうちオプション a4j, dvipdfmx は省略可能です．

3.5.2 環境

LaTeX の “**環境**” について解説します[15]．

環境とは命令の作用する範囲のことであり，\begin と\end という 2 つのコマンドで囲むことにより指定します．例えばソースコード 3.1 の 3 行目と 23 行目を見ると，次のようになっています．

```
\begin{document}
....
\end{document}
```

`{}`で囲われた部分は，その環境の名前を表わしています．そして\begin は始まりを，\end は終わりを表わします．

例えば `\begin{document}`となっていたら，“document 環境の始まり” という意味になり，`\end{document}`は同様に “document 環境の終わり” を意味します．

3.6 クラスファイルの指定

LaTeX ではまず，どのような種類の文書であるのかを明示する必要があります．それを行うのが `\documentclass`コマンドです．

ソースコード 3.1 で提示したサンプルを見ると，最初の行が次のようになっています．

```
\documentclass[a4j, dvipdfmx]{jsarticle}
```

このコマンドにおいて，オプションに紙のサイズや文字の大きさなどを，引数に具体的な**ドキュメントクラス**を指定します．

3.6.1 ドキュメントクラス

日本語の文書では奥村晴彦氏が開発した jsarticle や jsbook が主に使われています．この 2 つの使い分けを説明します．

15） 環境という概念は純粋な TeX にはなく，LaTeX によって提供されているものです．

表 3.3: クラスファイルの一覧

クラスファイル	意味
jsarticle	一般的な文書，レポート
jsbook	書籍

授業のレポートなど多くの場合は jsarticle を用いれば大丈夫です．

3.6.2 jsarticle のオプション

オプションを指定することで，ドキュメントクラスに対して文書の形式等を指定できます．ソースコード 3.1 では a4j というオプションが指定してありました．a4j は “A4 用紙の大きさで文書を作成する” という意味です．

また，ソースコード 3.1 で指定してあったオプション dvipdfmx は，「DVI ファイルから PDF を得るために，dvipdfmx を用いる」ことを表すオプションです．このように，利用する DVI 処理系をオプションとして文章冒頭で渡しておいたほうが，エラー[16)]を未然に回避できます．

以下に jsarticle のオプションを記してありますので参考にしてください．

3.6.2.1 用紙サイズ

a4paper 余白の広い A4 サイズ
b5paper 余白の広い B5 サイズ
a4j 余白の狭い A4 サイズ
b5j 余白の狭い B5 サイズ

3.6.2.2 文字サイズ

オプションで文字サイズの指定がなければ，10pt が本文に適用されます．

- 10pt
- 11pt
- 12pt

3.6.2.3 段組

新聞のような文書の段組を指定できます．

onecolumn 段組なし
twocolumn 2 段組

3.6.2.4 ページの体裁

両面印刷を前提としたとき，奇数ページと偶数ページでページの体裁を変えるかどうかを決めることができます．

oneside 全て同じ体裁
twoside ページの体裁は偶数とページと奇数ページとで異なる

3.6.2.5 組版の確認

draft 行の右端がはみでたときに印を付ける
final 最終版．何も印はつかない

16) 具体的には，各種パッケージのデフォルトの DVI 処理系と，実際に利用される DVI 処理系が異なることによるエラー．

この他のオプションについて知りたい方は jsarticle などが収録されている jsclasses パッケージ[17]のドキュメントを参照してください.

3.7 本文

文書の記述を始めるにあたり，ドキュメントクラスを記述しました．次に記述するのは document 環境です．document 環境は，どこが文書の本文なのかを明示します．つまり，文書に出力されるのは document 環境内のみで，\begin{document}の前や\end{document}以降に文章を記述しても無視されます.

3.8 表題と著者

論文やレポートを書くうえで表紙は大事です．これらの文書では自分の名前や学籍番号などを記述することが求められます．表紙ではタイトル，名前等を書きますので，本節ではそれらの書き方を紹介します.

表題 \title{タイトル}のようにして，文書の表題を記述します.

著者 \author{山田 太郎}のように文書の著者を記述します．複数著者がいる場合は，\andコマンドで区切ります.

日付 \date{2011/3/11}のように，文書にかかわる日付（作成日，提出日など）を記述します．\date{}とすると日付は入力されず空となり，\date\todayとすると作成日の日付が自動的に入力されます.

ここまでで表題や著者の書き方を紹介しました．しかしこれらだけでは表題ページは生成されません．そこで使用するのが \maketitle コマンドです．これを使用することで，タイトル等が整形された形で生成されます.

例えばまとめると以下のソースコード 3.2 のようになります.

Listing 3.2: 表題ページの例

```
1  \documentclass[a4j, dvipdfmx]{jsarticle}
2
3  \title{日本語入力SKKについて}
4  \author{\text{情報 雅彦} \and Yukiyoshi KAGAKU}
5  \date\today
6
7  \begin{document}
8
9  \maketitle
10
```

17） http://www.ctan.org/pkg/jsclasses

```
11 日本語入力SKKとは……
12
13 \end{document}
```

これをコンパイルすると次の図 3.4 となります．

図 3.4: 表題ページを出力する

3.9 見出し

よい文章を書くためには文章の中の章立てを作ることが重要です．新聞などを見ると，まず大きな文字で書かれた**見出し**を用いてその後の本文が一体何について論じているのかを一言で述べ，本文では端的にこのことについて論じたいという目標を明らかにします．見出しという文章の目標があることで，何について意識して読めばよいのかが明瞭になり，読者にとって文章がより理解しやすくなります．

LaTeX における見出しの書き方を紹介します．見出しの中でもっとも大きいのは `\part`（部）になりますが，大きいものから始めなければいけないということはありません[18]．レポートなどの場合は `\section`（節）から始めればいいでしょう．表 3.4 は見出しの表です．

表 3.4: 見出しの種類

見出し	意味
`\part`	部
`\chapter`[19]	章
`\section`	節
`\subsection`	小節
`\subsubsection`	小々節
`\paragraph`	段落
`\subparagraph`	小段落

18） ただし，例えば `\chapter` の後に `\subsection` を使うなど，途中の見出しを飛ばすのはよくありません．

以下のソースコード 3.3 は見出しの例です．

Listing 3.3: 見出しの例

```
\documentclass[a4j, dvipdfmx]{jsbook}

\begin{document}
\part{地球}
地球は太陽系の惑星の一つで太陽から三番目に近く，生物が生存している星である．

\chapter{アジア}
アジアはヨーロッパを除いたユーラシア大陸全般を指す．

\section{日本}
日本は日本列島を主な領土とする東アジアの国家である．

\subsection{茨城}
日本の県の一つで関東地方の北東に位置し，東は太平洋に面する．

\subsubsection{つくば}
茨城県南部に位置し，筑波研究学園都市はつくば市全域を区域とする．

\paragraph{天王台}
つくば市の東部に位置し，地域内のほぼすべてが筑波大学や研究施設である．

\subparagraph{1-1-1}
筑波大学天王台キャンパスがある．
\end{document}
```

これは次の図 3.5 のようになります．

19） \chapterコマンドは jsarticle クラスには含まれていません．jsbook に含まれています．

第 I 部

地球

地球は太陽系の惑星の一つで太陽から三番目に近く，生物が生存している星である．

第 1 章

アジア

アジアはヨーロッパを除いたユーラシア大陸全般を指す．

1.1　日本

日本は日本列島を主な領土とする東アジアの国家である．

1.1.1　茨城

日本の県の一つで関東地方の北東に位置し，東は太平洋に面する．

つくば
茨城県南部に位置し，筑波研究学園都市はつくば市全域を区域とする．

■天王台　つくば市の東部に位置し，地域内のほぼすべてが筑波大学や研究施設である．
1-1-1　筑波大学天王台キャンパスがある．

図 3.5: 見出しのコンパイル結果

この例では説明のため各見出しが一度しか用いられていませんが，本来このような文章の構造は避けるべきです．

3.10　書体・文字サイズ

見出し（3.9 節）のコマンドを用いて文書を章立し文章の構造を作ることが重要であるように，**一部だけ書体を変更する**などして，文章の中で特に注目して欲しい部分を明示的にするというのも文章を良くするうえで重要です．

3.10.1　書体の変更

3.10.1.1　和文書体

和文（ひらがな・カタカナ・漢字）に関する書体を変更する場合には次のコマンドを用います．

`\textgt`
　引数の文字列を**ゴシック体**にします．

`\textmc`
　引数の文字列を明朝体にします．

`\textbf`
　引数の文字列を**太字**にします．

和文では**ゴシック体**にすることで文章の中で強調を表わすことが多いです．ただ，強調をする場合は `\emph`コマンドを用いるほうが意味も分かりやすく欧文にも使えるので便利です．

`\textbf`コマンドで**ボールド**にしたり，あるいは `\underline`コマンドで下線を引いたりといった方法による強調は，ページにおける黒色の割合が増えて密度が高く見えてしまうなどの理由から，安易な多用はお勧めしません．

3.10.1.2　欧文書体

主に欧文の書体を変更します．

`\textbf`
　引数の文字列を**ボールド（Bold）**にします．基本的には見出しや表題の文字列に使います．

`\textit`
　引数の文字列をイタリック（*Italic*）にします．強調するときに使います[20)]．

`\texttt`
　引数の文字列をタイプライタ（`Typewriter`）にします．ユーザの入力部分などに用います．

`\textsc`
　引数の文字列をスモールキャピタル（Small Capital）にします．

20）　強調する際には `\texttt`コマンドよりも，`\emph`コマンドの方が直感的でかつ和文にも使えます．

前にも述べましたが \textbfの多用は，ページに文字が詰っている印象を与えるので使用はほどほどにしましょう．

3.10.2 文字サイズ

文字サイズを変更する際は次のコマンドを使います．

表 3.5: 文字サイズを変更するコマンド

コマンド	結果
\tiny	tiny
\scriptsize	scriptsize
\footnotesize	footnotesize
\small	small
\normalsize	normalsize
\large	large
\Large	Large
\LARGE	LARGE
\huge	huge

注意するべきは，これらのコマンドは引数を取ってその文字列の大きさを変更するというわけではないことです．例えば，\large{large text}とするのは誤りです．これらのコマンドは**グループ**という，{と }で囲んだ部分や環境の中全体に作用します．つまり，{\large large text}などとするのが正しい使い方です．

ただ，これらのコマンドを用いて安易に文字の大きさを変更するべきではありません．多くの場合は見出し（3.9 節）のコマンドなどによって適切な文字サイズになります．本文の文字サイズを無闇に変更すると，本文の高さが各行で異なってしまって見た目の印象がよくありません．文字サイズの変更は例えば表（3.14 節）の中など，本文ではない部分について行うのがよいでしょう．

3.11 改行・改ページ

LaTeX においてはただ改行しただけでは改行されません．これは改行や改ページといったことは TeX の処理系などが適切と判断した場所で行われるため，LaTeX のソースコードを編集する私たちが気にすることではないという考えによります．

次の例を見てください．

```
1 ソースコードでは，この直後に改行するものの，
2 改行されません．
```

結果は次のようになります．

ソースコードでは，この直後に改行するものの，改行されません．

一見直感とは異なり不自由に思うかもしれないですが，これにより，1 つの段落が非常に長くなってしまったとしても，ソースコードでは複数の行に分けて記述できます．従って，テキストエディタを横にスクロールする必要がなくなりより快適な編集ができます．

ただ TEX の処理系による改行や改ページの判断が必ずしも適切とは限りません．止むを得ず，編集者が改行や改ページを行わなければならない局面があるので，ここでは改行や改ページの方法を説明します．

3.11.1 改行

改行するためにはどうしたらいいかと言うと \\を使います．

```
この直後に改行する． \\
改行されます．
```

結果は次のようになります．

この直後に改行する．
改行されます．

このように改行されますが，この方法は先にも説明したとおり止むを得えない場合にのみ使うべきです．普段の改行については TEX の処理系に任せるべきです．

3.11.2 改ページ

次に改ページの方法を紹介します．改ページは\newpageコマンドや\clearpageコマンドで行うことができます．この 2 つのコマンドは二段組など段組の中で用いられた場合の挙動が異なります．

\newpage
: 次の段がある場合は次の段へ行き，ない場合は改ページを行います

\clearpage
: 段組に関わらず改ページを行います

ここでは実行例を示しませんので各自でやってみてください．

3.12 箇条書き

箇条書きの方法を紹介します．LATEX の箇条書きは 3 つの種類があります．

3.12.1　順序なし箇条書き

itemize 環境は**順序なし**の箇条書きを作ります.

```
\begin{itemize}
  \item 鍋
  \item フライパン
\end{itemize}
```

結果は次のようになります.

- 鍋
- フライパン

3.12.2　順序あり箇条書き

enumerate 環境は**順序あり**の箇条書きとなります.

```
\begin{enumerate}
  \item フライパンに油を敷く
  \item 卵を入れる
\end{enumerate}
```

結果は次のようになります.

1. フライパンに油を敷く
2. 卵を入れる

3.12.3　定義リスト

定義リストとは単語とその意味を列挙する際に用います. LaTeX では description 環境を用いて次のように書きます.

```
\begin{description}
  \item[つくば]
    筑波大学があります.

  \item[土浦]
    つくば国際大学があります.
\end{description}
```

このようになります.

つくば　筑波大学があります.
土浦　つくば国際大学があります.

3.13 図

これまでは文字列を扱う記述について紹介してきましたが，本節ではレポートでは必須の画像の取り扱い方法を紹介します．

3.13.1 graphicx パッケージ

画像を挿入するにあたり，\documentclass コマンドと\begin{document}の間[21]に\usepackage[dvipdfmx]{graphicx}と記述し，graphicx パッケージ[22]を読み込んでいます．graphicx パッケージは画像を LaTeX で使うために必要なパッケージです．このパッケージは画像の他にも，図形などに関する様々な機能を提供しますが本章では解説しません．詳細を知りたい方は脚注 22)の URL を参照してください．

この graphicx パッケージによって，画像を表示するための \includegraphics コマンドが使えるようになります．

3.13.2 画像の表示

\includegraphics コマンドを用いた例を以下のソースコード 3.4 に示します．

Listing 3.4: 画像の挿入

```
1  \documentclass[a4j, dvipdfmx]{jsarticle}
2  \usepackage[dvipdfmx]{graphicx}
3  \usepackage{float}
4
5  \begin{document}
6
7  \begin{figure}[H]
8    \centering
9    \includegraphics[width=5cm]{example-image-a.pdf}
10   \caption{画像の挿入}
11   \label{fig:testpicture}
12 \end{figure}
13
14 \end{document}
```

実行結果は図 3.6 のようになります．

21） このことを**プリアンブル**といいます．
22） http://ctan.org/pkg/graphicx

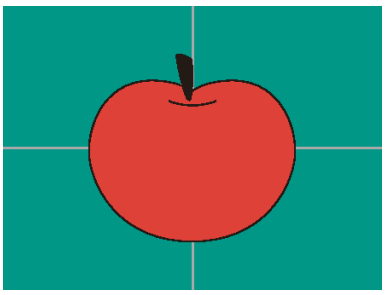


図 3.6: 画像の挿入

3.13.2.1 float パッケージ

まず 3 行目にて，float パッケージを読み込んでいます．このパッケージは画像を配置を直感的にするものですので，導入をお勧めします．

3.13.2.2 figure 環境

figure 環境は環境内にあるものが図であるということを示す環境です．\begin{figure}[H] の [H]の部分は表示位置に関する設定を表わしています．H は float パッケージによって提供されるものですが，LaTeX の標準では次のような指定ができます．

表 3.6: figure 環境などで使える位置指定子

位置指定子	意味
h	環境が使用された位置に出力します
t	環境が現れるページの上端に出力します
b	環境が現れるページの下端に出力します
p	表や図からなるページを作成しそこに出力します

これらは [htbp]といった具合に併記でき，そうした場合は書かれた順に優先度を持ちます．例えば [htbp]の場合はまず，“h” に従い環境が使用された位置に出力しようとします．しかしそれが何らかの理由によってできない場合，TeX 処理系は次に “t” に従い，ページの上端に表示しようとします．というように，最終的に可能なものを用います．

ただ，多くの場合は float パッケージを用いて H のみを指定すればよいので，これらを覚える必要はあまりないでしょう．

3.13.2.3 \centering コマンド

これはその名の通り， \centering コマンド以降の文章や画像などを中央寄せにします．このコマンドは現在の環境，あるいはグループの中で終了します．

3.13.2.4 \includegraphics コマンド

また 9 行目では [width=5cm]にて，画像の大きさを 5cm と指定しています．そして，example-image-a.pngという画像を読み込んでいます[23]．実際に使うときは，LaTeX ファイルからの相対パス名で画像を指定します．

3.13.2.5 \caption コマンド

そのあとに来る \captionコマンドは画像に対する簡単な説明（キャプション）を書きます．一般的に図のキャプションは図の下に記述します．このコマンドは省略できます．

3.13.2.6 \label コマンド

\labelコマンドは，画像に本文などから参照するための**ラベル**を付加します．このコマンドは省略できます．このラベルの使用目的，方法は後述の参照（3.16 節）にて紹介します．

3.14 表

LaTeX においては tabular 環境を用いて表を作成し，その表の配置やキャプションを table 環境やその中で指定します．表は図とは異なり，プリアンブルには何も追記する必要はありません[24]．

実際に利用する際には以下のソースコード 3.5 のように記述します．

Listing 3.5: 表の例

```
\begin{table}[H]
  \caption{25マス計算（掛け算）}
  \label{table:tableexample}

  \centering
  \begin{tabular}{|r||c|c|c|c|c|}
  \hline
  掛け算 & 2  & 3  & 5  & 7  & 1 \\ \hline \hline
  1     & 2  & 3  & 5  & 7  & 1 \\ \hline
  4     & 8  & 12 & 20 & 28 & 4 \\ \hline
  9     & 18 & 27 & 45 & 63 & 9 \\ \hline
  3     & 6  & 9  & 15 & 21 & 3 \\ \hline
  8     & 16 & 24 & 40 & 56 & 8 \\ \hline
  \end{tabular}
\end{table}
```

結果は次の表 3.7 のようになります．

23） あらかじめ画像 example-image-a.png を用意する必要があります．
24） ただし figure 環境のように表示位置として H を用いる場合は float パッケージが必要となります．

表 3.7: 25 マス計算（掛け算）

掛け算	2	3	5	7	1
1	2	3	5	7	1
4	8	12	20	28	4
9	18	27	45	63	9
3	6	9	15	21	3
8	16	24	40	56	8

図と同様に\begin{table}の直後に [H]があります．figure 環境と同様に表 3.6 にある表示位置を指定できます．ですが，特に理由がない限り [H]を用いれば問題ありません．

table 環境の中には figure 環境と同様にキャプション（ \captionコマンド），ラベル（ \label コマンド）を書くこともできます．

表は図とは異なり，一般的にキャプションを表の上に記述します．図のソースコード 3.4 と表のソースコード 3.5 を見比べてもらえばわかりますが，それぞれのキャプションの位置はソース上のその位置と同じです．つまり，これらを逆にすることも可能ということですが，一般的にはこの順番を用います．

3.14.1 tabular 環境

tabular 環境は表に特有な，まるで数学の行列式のような構造を作る環境です．この tabular 環境は LaTeX の内部でとても複雑な処理をしています．そのお陰で，私たち一般の LaTeX 利用者は容易に表を作ることができます．

3.14.1.1 セル内の文字位置

\begin{tabular}の直後に， |r||c|c|c|c|c|という記述がありますが，これは表の線と内容の配置を示しています．各アルファベット（r や c）は各列の文字の配置を示し，一列につき 1 つしか指定できません．それぞれの意味は以下のようになります．

表 3.8: セル内の位置指定子

位置指定子	意味
l	文字列を左寄せにします
c	文字列を中央寄せにします
r	文字列を右寄せにします

3.14.1.2　縦線

また，各々の間に書かれている `|` は表の列間の線を表しており，これは書いた本数分だけ線が引かれます．ここでの例では，一列目と二列目の間には二本の線が引かれていますが，それ以外の場所は一本です．

3.14.1.3　セルと横線

続いて tabular 環境の中身を見てみます．各列は `&` によって分けられます．また，各列は `\\` によって行の終わりを表現します．そして `\hline`コマンドは水平線を引くためのものです．例のように 2 つを連続で記述することにより二重線を記述できます．

3.14.1.4　複数の列をまたぐセル，複数の行ををまたぐセル

また，表を利用する際には 2 つ以上のセルにまたがる表を作成したい，ということがあります．このような書き方は以下のソースコード 3.6 のようにすることで可能になります．

Listing 3.6: 複数セルにまたがる表

```
\begin{table}[H]
  \caption{複数セルにまたがる表}
  \label{table:multicolumn}

  \centering
  \begin{tabular}{|l|c|r|r|}
    \hline
    \multicolumn{4}{|c|}{メンバ} \\ \hline \hline
    主戦力 & Aさん & Bさん & Cさん \\ \cline{2-3}
    & Dさん & Eさん & Fさん \\ \hline
    副戦力 & Gさん & Fさん & Gさん \\ \cline{3-4}
    & Hさん & Iさん & Jさん  \\ \hline
    補欠 & \multicolumn{3}{|c|}{新人}\\ \hline
  \end{tabular}
\end{table}
```

実行結果は表 3.9 のようになります．

表 3.9: 複数セルにまたがる表

メンバ			
主戦力	A さん	B さん	C さん
	D さん	E さん	F さん
副戦力	G さん	F さん	G さん
	H さん	I さん	J さん
補欠	新人		

ここで用いているのは \multicolumn コマンドです．これは直後の引数で記述した数の列をまとめて 1 つの列とし，その次の引数で当該セル内の配置を指定し，最後の引数にセル内に入るコンテンツを記述します．

ここまでは複数の列にまたがるセルが対象でしたが，次に見るのは複数の行にまたがるセルです．これは \cline コマンドを用いることで表現できます．表の 2 行目，3 行目を見てみると 2 列目と 3 列目に上下を分けるような線が入っています．これは \cline の直後の {} によって範囲を指定できます．2，3 行目と同様に 4，5 行目も以下のように記述することで，3 列目と 4 列目に上下を分けるような線が引かれています．これを用いることで，上下を分けるような線の引かれていない場所を結合セルとして扱うことができます．

3.15 脚注

脚注とは本文に書くほどのことではない補足を書くために用いるものです．

3.15.0.1 \footnote コマンド

LaTeX では \footnote コマンドを用いて，次のように行うことができます．

```
\TeX\footnote{Knuthが作ったプログラム言語．}は優れた組版能力があり……
```

実行すると次の図 3.7 のようになります[25]．

TEX[a]は優れた組版能力があり……

a Knuth が作ったプログラム言語.

図 3.7: 脚注の例

3.15.0.2 \footnotemark，footnotetext コマンド

\footnotemark コマンドと \footnotetext コマンドを用いることで，脚注番号と脚注本文を別々に書くことができます．次のようにします．

```
1 \TeX\footnotemark は優れた組版能力があり……
2
3 \footnotetext{Knuthが作ったプログラム言語．}
```

\footnotemark が脚注番号を出力し，\footnotetext が脚注本文を構築します．上記の例の結果は図 3.7 と同じです．これは脚注本文がとても長くなってしまって，\footnote で書くとソースコードが読み難くなってしまうのを防ぐ効果があります．

25） 実行結果では脚注番号が “a” になっています．これはサンプルの都合によるもので，通常は算用数字が振られます．

3.16　参照

論文やレポートにおいて画像や表を挿入した場合，参照を付けなければならないことがあります．ここでは LaTeX の便利な点である図表の参照方法について述べます．図表の番号を自分で付け，それへの参照を自分でつけた番号をもとに行うといった作業が LaTeX では必要ありません．LaTeX は参照に必要な番号を自動で割り振ってくれるのです．

参照の仕方ですが，`\ref`コマンドと `figure` 環境（3.13 節）などで紹介された `\label`コマンドを用いることで実現できます．3.13 節で用いた図と，3.14 節で用いた表を例としますと以下のように書くことができます．

```
図\ref{fig:testpicture}と表\ref{table:tableexample}
```

次のようになります[26]．

> 図 3.6 と表 3.7

このように `\ref`コマンドの中に，`\label`コマンドでラベルとして記述した文字列を入力することで参照できます．注意していただきたいのは，別の図表に同じラベルを付けると後に出てきたラベルの方が優先されてしまいます．ラベルは重複させてはいけません．そこで，重複を防ぐためにも図のラベルは `fig:` から，表のラベルは `table:` から始めるなどという慣習があります．

3.16.1　見出しの参照

`\label`コマンドは図や表以外にも，`\section`コマンドといった見出しや脚注にも付けることができます．例えばこの節には次のようにラベルが付けられています．

```
\section{参照}
\label{sec:reference}
```

次のように参照します．

```
第\ref{sec:reference}節
```

どのようになるのかは，本章にたくさんある参照を見ると分かります．

3.16.2　脚注の参照

`\footnote`コマンド内で `\label`コマンドを用いれば，脚注を参照できます．次のようにします．

```
\LaTeX\footnote{\label{fn:latex}Lamportが作ったマクロです．} ..... \\
..... \\
\LaTeX については脚注\ref{fn:latex}を参照してください．
```

26）この例では参照に PDF のリンクが貼られています．これは `hyperref` パッケージの機能を使って行なっているので，これを読み込んでいない場合は参照の番号が表示されるだけでリンクは貼られません．`hyperref` パッケージを用いたリンクの貼り方は `https://texwiki.texjp.org/?hyperref` をご覧ください．

次のようになります．

LaTeX[a] .....

.....

LaTeX については脚注 a を参照してください．

---

a Lamport が作ったマクロです

3.17 数式

さて，ここまでのところで画像や表を用いたレポートや論文は書けるようになりました．しかし，コンピュータサイエンスの分野にいる以上は数式とは切っても切れない縁があります．そこで，文書内の数式をきれいに書く技術が必要になってきます．MS Word などにも数式エディタが存在するように，もちろん LaTeX にも数式を書くための方法が用意されています．例えば，以下のような式を記述できます．

$$\int \frac{1}{x^2-1}\mathrm{d}x = \frac{1}{2}\log\left|\frac{x-1}{x+1}\right| + C$$

通常の数式エディタでは指数をうまく表示できなかったり，複雑になると全体の形が崩れたりしてしまうことが多々ありますが，LaTeX は世の中のほとんどの数式に対応しています．なぜなら世の中の数学者が論文を書く際には多くの場合 LaTeX を使い，もし既存の LaTeX にはない新しい数学的表現を導入したならば，それは LaTeX の機能で実装されます．従って，LaTeX は世の中にある多くの数学的な表現に対応していると言えるのです．

また，この節では LaTeX による数式の表現力を示すために，やや専門的な数式が例として書かれています．この節はあくまで LaTeX の数式能力に関する解説なので，例にある数式の意味が分からなくとも問題はありません．

3.17.1 amsmath パッケージを利用して数式を書く

LaTeX には数式を扱うためのさまざまなコマンドや環境が提供されていますが，現在数式を扱う論文はほとんどが amsmath パッケージ[27] を用いていると言われています．つまり業界のデファクトスタンダードということで，本節でも amsmath を前提とした数式の書き方を説明します．

amsmath パッケージは graphicx パッケージ（3.13 節）と同じく，`\usepackage` コマンドによって，`\usepackage{amsmath}` という記述で読み込むことができます．

実際に amsmath パッケージを用いて数式を書いてみましょう．次のソースコード 3.7 をコンパイルしてみてください．

27） http://www.ctan.org/pkg/amsmath

Listing 3.7: Hello amsmath!

```
\documentclass[a4j, dvipdfmx]{jsarticle}
\usepackage{amsmath}

\begin{document}

\begin{align}
\left(
  \begin{array}{cc}
    2 & -1\\
    -3 & 4 \\
  \end{array}
\right)
\left(
  \begin{array}{c}
    1 \\
    2 \\
  \end{array}
\right) = \left(
  \begin{array}{c}
    0 \\
    5 \\
  \end{array}
\right)
\end{align}

\end{document}
```

次の式（3.1）のようなります．

$$\begin{pmatrix} 2 & -1 \\ -3 & 4 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 5 \end{pmatrix} \tag{3.1}$$

amsmath パッケージをより詳しく知りたい方へ

ここで利用した amsmath パッケージは $\mathcal{AMS}$-LaTeX の一部です．$\mathcal{AMS}$-LaTeX とは American Mathematical Society[a]によって開発された，LaTeX 数式機能を強化するマクロパッケージ amsmath と，American Mathematical Society の組版に合わせた amscls というクラスファイル，さらに数式に用いる記号類のフォントをまとめたものをいいます．このクラスファイルとは，jsarticle（3.6 節）などと同じく文書のレイアウトを定義するものです．これは American Mathematical Society へ論文などを投稿するときに利用されています．

[a] http://www.ams.org/home/page

さて，肝心な書き方ですが数式を記述する方法が大きく分けて**ディスプレイ数式**と**インライン数**

式の 2 つが存在します．

3.17.2 ディスプレイ数式

ディスプレイ数式とは，数式を本文とは別の行にして表示する方法です．ディスプレイ数式のために，次のようなコマンドや環境が提供されています[28)]．

- align 環境
- align*環境
- \[コマンドと \]コマンド

これらの違いを順に説明します．

3.17.2.1 align 環境

ソースコード 3.7 に用いられていた環境です．この align 環境は**番号付き**ディスプレイ数式を提供します．気付いた方もいるでしょうが，ソースコード 3.7 を実行すると“（3.1）”のような番号が式の右端に自動で付与されます．これが番号付きということの意味です．

また，この番号は複数の行にわたる数式を記述した場合も，自動で各行に振られます．次のソースコードを見てください．

Listing 3.8: 複数行にわたる数式

```
\begin{align}
  \overrightarrow{dd} &= \overrightarrow{rd} + \left(\vec{d} \cdot
      \vec{n}\right)\vec{n} \label{eq:vectordd1} \\
  &= \vec{d} - \left(\vec{d} \cdot \vec{n}\right)\vec{n} \label{eq:vectordd2}
\end{align}
```

次のようになります[29)]．

$$\overrightarrow{dd} = \overrightarrow{rd} + \left(\vec{d} \cdot \vec{n}\right)\vec{n} \tag{3.2}$$

$$= \vec{d} - \left(\vec{d} \cdot \vec{n}\right)\vec{n} \tag{3.3}$$

数式内で改行をしたい場合は，\\で改行場所を指示する必要があります．これは数式の改行位置を TEX の処理系が判断できないからです．

また，式（3.2）と式（3.3）は =の位置で整列しています．これは揃えたい場所に &を置くことで実現できます．

ソースコード 3.8 を見て気付いた方もいるかもしれませんが，数式に対してもラベルを用いることができます．つまり，他の場所からの参照（3.16 節）が可能ということです．書き方は通常の参

28） equation 環境や equarray 環境，$$など，この他にも LATEX が提供する数式用の環境がありますが，AMS-LATEX で提供される align 環境などを使う方が良いとされています．

29） この例では番号が（3.2）から始まっていますが，これは先ほどの式（3.1）の番号を 1 としているためです．

照と同様に \refコマンドを用います．

一部の数式にだけ番号を振る　最初の例では両方の数式に数式番号が振られています．しかし 1 つの数式が複数の行にわたっており，数式番号をその全てには付けたくないということが考えられます．そのような場合は \nonumberコマンドを用いて，番号の付与を抑制できます．次のようにします．

```
\begin{align}
\left(\lambda r.r\right)\left(\lambda x.\lambda y.x~y\right) & \rightarrow_{\eta}
  \left(\lambda r.r\right)\left(\lambda x.x\right)\nonumber \\
  & \rightarrow_{\beta} \lambda x.x
\end{align}
```

結果はこのようになります．

$$\begin{aligned}\left(\lambda r.r\right)\left(\lambda x.\lambda y.x\ y\right) &\rightarrow_{\eta} \left(\lambda r.r\right)\left(\lambda x.x\right) \\ &\rightarrow_{\beta} \lambda x.x\end{aligned} \tag{3.4}$$

このようにある行には番号を付けず，ある行には付けるということが可能です．

3.17.2.2　align*環境

こちらは align 環境とは異なり，**番号なし**ディスプレイ数式を提供します．align 環境では原則全ての行に番号が振られていましたが，align*環境は数式に番号を振りません．従って，番号がないので \labelでは参照できません．番号が振られない以外に align 環境と違いはありません．次の例を見てください．

```
\begin{align*}
\tau &::= \alpha\, |\, int\, |\, bool\, |\, \tau_1 \rightarrow \tau_2 \\
\sigma &::= \tau\, |\, \forall \alpha. \sigma
\end{align*}
```

実行すると次のようになります．

$$\begin{aligned}\tau &::= \alpha \,|\, int \,|\, bool \,|\, \tau_1 \rightarrow \tau_2 \\ \sigma &::= \tau \,|\, \forall \alpha.\sigma\end{aligned}$$

このように，行に番号が振られていません．

3.17.2.3　\[コマンドと\] コマンド

\[コマンドと \]コマンドで囲まれた部分は**一行のみ**のディスプレイ数式となります．align 環境や align*環境のように \\を用いて複数行の数式を書くことはできません．次のようになります．

```
\[
  Fun\left(x, \protect\underbrace{Let%
  \left(f, \protect\overbrace{Fun\left(y, x\right)}^{\forall\alpha_2.\alpha_2
      \rightarrow \alpha_1}, f\right)}_%
  {\forall\alpha_2.\alpha_2 \rightarrow \alpha_1}\right)
  : \forall\alpha_2.\alpha_1 \rightarrow \left(\alpha_2 \rightarrow
      \alpha_1\right)
\]
```

実行すると次のようになります．

$$Fun\left(x, \underbrace{Let\left(f, \overbrace{Fun\,(y,x)}^{\forall\alpha_2.\alpha_2\rightarrow\alpha_1}, f\right)}_{\forall\alpha_2.\alpha_2\rightarrow\alpha_1}\right) : \forall\alpha_2.\alpha_1 \rightarrow (\alpha_2 \rightarrow \alpha_1)$$

さて，これまでの例で LaTeX はとても複雑な数式を表現できるということが分かったのではないでしょうか．

3.17.3　インライン数式

先に述べたディスプレイ数式（3.17.2 節）は数式と本文を完全に分離していました．ですが，レポートなどでは文章の中に数式を挿入したいということがあります．そのようなときには `$`を用いて本文に数式を埋め込むことができます．次に例を示します．

```
離散フーリエ変換$X_k\left(k = 0, 1, \dots , N - 1\right)$は級数
$X_k = \sum^{N-1}_{n=0}x_n\mathrm{e}^{-i\frac{2\pi kn}{N}}$となり，
この計算量は$\mathcal{O}\left(N^2\right)$になる．
```

次のようになります．

> 離散フーリエ変換 $X_k\,(k = 0, 1, \ldots, N-1)$ は級数 $X_k = \sum_{n=0}^{N-1} x_n \mathrm{e}^{-i\frac{2\pi kn}{N}}$ となり，この計算量は $\mathcal{O}\left(N^2\right)$ になる．

このように `$`で囲まれた部分が数式として解釈されます．

3.18　ソースコード

情報科学類の皆さんは課題のレポートや論文などでソースコードを出力したいということがあるでしょう．LaTeX では `listings` パッケージ[30]という伝統的なマクロパッケージによって，この本で用いられているようなソースコードの出力を簡単に行えます．

30）`http://www.ctan.org/pkg/listings`

3.18.1 listings パッケージの読み込み

graphicx パッケージ（3.13 節）と同様に，\usepackage コマンドを用いてプリアンブルに \usepackage{listings} と記述します．

3.18.2 設定

listings パッケージは大変高性能なパッケージなのですが，初期設定ではあまり美しい出力が得られません．設定は次の 2 つのコマンドで設定できます．

\lstdefinestyle
: 新しい**スタイル**を定義する命令です．

\lstset
: 全てのソースコードに対して設定を適用します．

3.18.2.1 スタイルの定義

例えば 1 つのレポートに Java のソースコードと C 言語のソースコードが混在する，という局面を考えるとします．そして，Java のソースコードの設定と C 言語のソースコードの設定は別々にしたいという時に，“スタイル” という設定を編集者が定義し，ソースコードに応じて柔軟に見た目を切り替えることができます．

\lstdefinestyle は次のように使います．

```
\lstdefinestyle{java}{
  language=java,
  morekeywords={lambda}
}

\lstdefinestyle{c}{
  language=c,
  numbers=left
}
```

このようにして，新しいスタイル java と c を作ることができます．

3.18.2.2 全てのソースコードに関する設定

統一したい設定については \lstset コマンドにて設定します．例えばこの手引きでは次のような設定をしています[31]．

Listing 3.9: 手引きの listings パッケージの設定

```
\definecolor{solarized@base03}{HTML}{002B36}
\definecolor{solarized@base02}{HTML}{073642}
```

31） 手引きでは color パッケージを用いて文字や背景の色を設定しています．http://www.ctan.org/pkg/color

```
\definecolor{solarized@base01}{HTML}{586e75}
\definecolor{solarized@base00}{HTML}{657b83}
\definecolor{solarized@base0}{HTML}{839496}
\definecolor{solarized@base1}{HTML}{93a1a1}
\definecolor{solarized@base2}{HTML}{EEE8D5}
\definecolor{solarized@base3}{HTML}{FDF6E3}
\definecolor{solarized@yellow}{HTML}{B58900}
\definecolor{solarized@orange}{HTML}{CB4B16}
\definecolor{solarized@red}{HTML}{DC322F}
\definecolor{solarized@magenta}{HTML}{D33682}
\definecolor{solarized@violet}{HTML}{6C71C4}
\definecolor{solarized@blue}{HTML}{268BD2}
\definecolor{solarized@cyan}{HTML}{2AA198}
\definecolor{solarized@green}{HTML}{859900}

\lstset{
  basicstyle=\small\ttfamily\color{solarized@base00},
  rulesepcolor=\color{solarized@base03},
  numberstyle=\scriptsize\color{solarized@base01},
  keywordstyle=\color{solarized@blue},
  stringstyle=\color{solarized@cyan}\ttfamily,
  commentstyle=\color{solarized@base01},
  emphstyle=\color{solarized@red},
  backgroundcolor=\color{solarized@base3},
  sensitive=true,
  breaklines=true,
  breakatwhitespace=true,
  framerule=0pt,
  frame=l
  showstringspaces=false,
  tabsize=2,
  basewidth={0.57em, 0.52em},
}
```

設定できる項目は `\lstdefinestyle` コマンドと `\lstset` コマンドの間に違いはありません．設定は膨大にあるので，詳細は脚注 30)にあるドキュメントを読んでいただくしかないのですが，その中でも特に重要なものを書き出しておきます．

language ソース中に書いてあるものが，何言語なのかを記述します[32)]．

style `\lstdefinestyle` にて定義した設定を指定します．

basicstyle ソース内の普通の文字のスタイルを決めます．

keywordstyle キーワードの文字スタイルを決めます．

breaklines `breaklines=true` とすると，自動改行が有効になります．見た目上の行と実

32) 記述の仕方が独特なので注意が必要です．詳しくは listings パッケージのドキュメント（脚注 30)）を参照してください．

際の行は区別されます．自動改行がないと listings の枠からはみ出る可能性があるので，有効にするほうがいいでしょう．

numbers　行番号の位置に関する設定です．値は次のものを取ります．

- none
- left
- right

firstnumber　行番号の初期値です．これに数値を設定するとその番号から始まるのですが，auto または何も指定しないと，前回の番号の次の値から始まります．

frame　枠に関する設定です．引数はいろいろあるのですが，とりあえず tbrl を入れておけば，四角で囲われます．

tabsize　タブの幅がスペース何個分かを数値で与えます．

xleftmargin, xrightmargin　左右のマージンを決めます．

3.18.3　listings パッケージの使い方

ようやく listings パッケージを使うための準備が整いました．使い方は次の 3 種類があります．

- LaTeX ソース中に直接埋め込む（lstlisting 環境）
- ファイルから読み込む（\lstinputlisting コマンド）
- 本文中に挿入する（\lstinline コマンド）

順に解説します．

3.18.3.1　lstlisting 環境

ソースコードを LaTeX ファイルに直接埋め込む際は lstlisting 環境を用います．次のようにします．

```
\begin{lstlisting}[language=java, numbers=left]
public class HelloWorld {
  public static void main (String[] args) {
    System.out.println("Hello World !!");
  }
}
\end{lstlisting}
```

設定 3.9 を用いて実行すると次のようになります．

```
public class HelloWorld {
  public static void main (String[] args) {
    System.out.println("Hello␣World␣!!");
  }
}
```

このように行番号が自動で振られ，文字列や予約語の色が自動で変更されています．

3.18.3.2 \lstinputlisting コマンド

\lstinputlisting コマンドはファイルを指定してそのファイルをソースコードとして読み込み，整形して表示します．

次のようにします．

```
\lstinputlisting[style=java]{src/example.java}
```

まず，[style=java]にて，\lstdefinestyle コマンド（3.18.2.1 節）にて定義した設定を呼び出し，それに基いて src/example.java を読み込み表示します．

3.18.3.3 \lstinline コマンド

これは本文の中にプログラムを埋め込む際に利用します．次のようになります．

```
この時，変数\lstinline[language=java]|x|の型は
\lstinline[language=java]|int|となる．
```

次のようになります．

この時，変数 xの型は intとなる．

情報科学類の学生は綺麗なソースコードを出力するために，秘伝のタレとなった listings の設定を持っているほうが少なくありません．他の人の書いたレポートなどのソースコードがとても美しく出力されていたら，その人に声をかけて設定を教えてもらうのもよいでしょう．

3.19 書いたとおりに出力する

\verb コマンドや verbatim 環境を利用するとテキストを書いた通りに出力ことができます．これらを利用した場合，特殊文字をエスケープせずに使用することができます．特殊文字については 3.5.1.1 の特殊文字の項を参照してください．

3.19.1 出力したいテキストが複数行の場合

複数行にわたるテキストを書いた通りに出力したい場合，verbatim 環境を利用します．

```
\begin{verbatim}
verbatim環境を使うと，
書いた通りに出力することができます!#^^#
\end{verbatim}
```

結果は次のようになります．

```
verbatim 環境を使うと，
書いた通りに出力することができます!#^^#
```

3.19.2　出力したいテキストが 1 行の場合

1 行のテキストを書いたとおりに出力したい場合，\verb コマンドを利用します．以下の例のように，書いた通りに出力したい文字列を*以外の 1 文字で挟みます．

```
\verb"書いた通りに出力したい文字列を*以外の1文字で挟む"
```

結果は次のようになります．

書いた通りに出力したい文字列を*以外の 1 文字で挟む

3.20　BibTEX を用いた参考文献

ここまででほとんどのレポートは問題なく書けるようになりました．しかしまだ論文（主に卒論）は書けません．ほとんどの卒論は他の文献を読まずには成り立ちません．つまり，何らかの参考文献が存在します．そのような論文はほぼ必ず参考文献を掲載します．ここでは参考文献を載せるための方法を紹介します．

3.20.1　BibTEX を用いない

まず BibTEX[33] を用いる前に，BibTEX が自動生成しているプログラムについて説明します．ソースコード 3.10 を例に見ていきましょう．

今回は LaTEX の第一人者である奥村晴彦氏の著書，“[改訂第 6 版] LaTEX 2ε 美文書作成入門” を例にします．

Listing 3.10: nobib.tex

```
\documentclass[a4j, dvipdfmx]{jsarticle}
\begin{document}
奥村先生の本\cite{奥村_黒木201310}を参考とした.
\begin{thebibliography}{10}
    \bibitem{奥村_黒木201310}%
        奥村晴彦,黒木裕介. [改訂第6版] LaTeX2ε美文書作成入門. 技術評論社,
            改訂第6,10 2013.
\end{thebibliography}
\end{document}
```

まずこれをコンパイルしてみます．

```
$ platex -kanji=utf-8 nobib.tex↵
$ platex -kanji=utf-8 nobib.tex↵
$ dvipdfmx nobib↵
$
```

33)　「びぶてふ」もしくは「びぶてっく」と読みます．http://ctan.org/pkg/bibtex

2 回 platex コマンドを実行しているのは，1 度目に後述の\citeによる参照情報の更新を行い，2 度目に最終的な出力ファイルを生成しているからです．

以下のような PDF ファイルが得られると思います．

図 3.8: nobib.pdf

奥村先生の本 [1] を参考とした．

参考文献

[1] 奥村晴彦, 黒木裕介. [改訂第 6 版] LaTeX 2ε 美文書作成入門. 技術評論社, 改訂第 6,10 2013.

3.20.1.1 thebibliography 環境

ソースコード 3.10 の 4 行目から 6 行目は thebibliography 環境となっており，ここで文献情報を書いていきます． \begin{thebibliography}{LONGEST-LABEL}～ \end{thebibliography}の間に\bibitem[LABEL]{KEY} ENTRYを並べていきます．

LABEL

参考文献のラベル（ソースコード 3.10 の赤文字部分）を指定します．このオプションは省略可能で，省略した場合には [1]，[2]，[3]，..... のように 1 から連番となります．

もちろん数字以外にも英数字や空白，日本語などが指定できます．

KEY

文献を参照するための目印（ソースコード 3.10 の緑文字部分）を記述します．これを後述の\citeコマンドにより参照します．他の目印と重複しないようにしてください．

ENTRY

文献の内容（ソースコード 3.10 の青文字部分）を記述します．ここには任意の LaTeX コードが記述できます．

LONGEST-LABEL

LABEL の中で一番長い文字と同じ数だけ数字を入力します（ソースコード 3.10 の黄文字部分）．

たとえば LABEL を一切変更せず参考文献の数が 2 桁だった場合，**22** とか **00**，**72** など適当な 2 桁の数字を入力しておきます．2 桁の数字ならなんでも構いません．

また LABEL を変更し，≺changed label≻ という ラベル（\bibitem[≺changed label≻]{なんとか}.....）が最長だった場合，文字数は 15（記

号が 2 文字，ローマ字が 12 文字，空白が 1 文字）なので **123456789012345** など任意の 15 桁の数字を入力します．

3.20.1.2 \cite コマンド

\citeコマンドは文献の参照に用います．ソースコード 3.10 では “奥村_黒木 201310” という指定をしています．

\labelコマンドと \refコマンドによる参照方法（3.16 節）とは少し違うということがお分かりいただけたでしょう．参照する際の命令が違うとはいえ，基本的にはその性質は変わりません．いちいち参照するたびに数字を書き換える必要もありません．

3.20.2 BibTEX を用いる

参考文献を掲載するには thebibliography 環境を用いる方法もありますが，LaTeX にまかせている処理はラベルに番号を振っている程度です．そこで文献内容などのレイアウトを一括して自動でおこなってくれるのが BibTeX です．

BibTeX は参考文献に関する**書誌情報**と呼ばれる，その参考文献の著者やタイトルなどという情報を決められたフォーマットで記述されたファイルを読み込みます．そして BibTeX はその書誌情報を処理して LaTeX で扱える形にし埋め込んでくれるソフトウェア（処理系）です．これにより，参考文献のデータベースを単一の LaTeX ファイルのみならず，他のファイルとも共有できます．

3.20.3 書誌情報の入手

参考にした論文や資料の書誌情報は多くの場合 Web で入手できるので，皆さんが自力で作成する必要はありません．BibTeX の書誌情報を提供している Web サイトは，TeX Wiki の BibTeX 関連ツール[34]にて紹介されています．また，CiNii[35]など論文検索サービスや学会の Web サイトでは論文の BibTeX 書誌情報を公開している場合が多いです．

先程と同様，“[改訂第 6 版] LaTeX 2ε 美文書作成入門” の BibTeX 書誌情報[36]を例にします．BibTeX の書誌情報は次のようなテキストファイルとなっています．

Listing 3.11: bibunsho.bib

```
1 @BOOK{奥村_黒木201310',
2 title={[改訂第6版] LaTeX2ε 美文書作成入門},
3 author={奥村 晴彦 and 黒木 裕介},
4 publisher={技術評論社},
5 year={2013},
6 month={10},
7 edition={改訂第6},
```

34） https://texwiki.texjp.org
35） 日本の論文検索サービス．http://ci.nii.ac.jp/
36） この書誌情報は Lead2Amazon（http://lead.to/amazon/jp/）という 書誌情報検索サービスから取得したものを一部改変したものです．

```
isbn={9784774160450},
url={http://amazon.co.jp/o/ASIN/4774160458/},
price={￥ 3,360},
totalpages={432},
timestamp={2014.03.13},
}
```

このファイルを “bibunsho.bib” という名前で保存します．

3.20.4 LaTeX ファイル側の書式

BibTeX の書誌情報を利用する LaTeX ファイルを用意する必要があります．次のソースコード 3.12 のようにします．

Listing 3.12: bibunsho.tex

```
\documentclass[a4j, dvipdfmx]{jsarticle}

\bibliographystyle{junsrt}

\begin{document}

奥村先生の本\cite{奥村_黒木201310'}を参考とした．

\bibliography{bibunsho}
\end{document}
```

このソースコード 3.12 には，\bibliographystyle コマンドと \cite コマンド，さらに \bibliography コマンドの 3 つが利用されています．これらについて解説します．

3.20.4.1 \bibliographystyle コマンド

これは参考文献の表示方法を設定するコマンドです．指定できる**スタイル**[37]とその意味を表 3.10 にまとめました．

表 3.10: 使用できるスタイルの例

スタイル	意味
jplain	もっとも標準的なスタイル
jalpha	文献の番号が著者名と出版年をあわせたものになる
jabbrv	できるだけ短かくする
junsrt	本文で引用した順に並べる

37) BibTeX のスタイルファイルはプログラム言語 TeX ではなく，特殊なプログラム言語にて記述してあります．普通はスタイルファイルの実装について知る必要はありませんが，興味のある方は http://tug.ctan.org/info/bibtex/tamethebeast/ttb_en.pdf をご覧ください．

3.20.4.2 \cite コマンド

ソースコード 3.12 では “奥村_黒木 201310’” という指定をしています．これは BibTeX 書誌情報 bibunsho.bib（3.11）の 1 行目で指定している**文献名**です．

3.20.4.3 \bibliography コマンド

最後に \bibliography コマンドを用いて参考文献の一覧を出力します．ソースコード 3.12 にて，\bibliography コマンドに bibunsho という文字列を渡しています．これは読み込む BibTeX 書誌情報ファイルを示しています．ただし，書誌情報ファイルの拡張子は除きます．今回の例では bibunsho.bib というファイル名ですので，拡張子を取り除き bibunsho という文字列を与えます．

3.20.5 BibTeX を用いた LaTeX ファイルのコンパイル

コンパイルするときの手順が増えます．BibTeX を用いたコンパイルのやり方は以下のように行います．以下の例では先に述べた bibunsho.tex をコンパイルしています．

```
$ platex -kanji=utf8 bibunsho.tex↵
$ pbibtex -kanji=utf8 bibunsho↵
$ platex -kanji=utf8 bibunsho.tex↵
$ dvipdfmx bibunsho↵
$
```

pbibtex という，BibTeX の処理系を日本語へ対応させたプログラムを用います．

thebibliography 環境を用いた場合と同じように，参照情報の更新と出力ファイル生成のために platex を 2 回実行します．

コンパイルが無事に終わると次のようになるはずです．

> 奥村先生の本 [1] を参考とした．

また巻末には次のような参考文献の一覧が出力されています．

参考文献

[1] 奥村晴彦，黒木裕介．[改訂第 6 版] LaTeX 2ε 美文書作成入門．技術評論社，改訂第 6,10 2013.

3.21 文書の余白

文書の余白とは文書の上下左右の隙間のことです．ここでは文書の余白を調整する方法を紹介します．

さまざまな方法がありますが，geometry パッケージ[38]を用いる方法がもっとも簡単でかつ安全です．これは以下のように使用します．

```
\usepackage{geometry}
\geometry{left=25mm, right=25mm, top=30mm, bottom=30mm}
```

左右の余白を 25mm にし，上下の余白を 30mm に設定しています．

この他にも geometry パッケージは特定のページだけ余白を設定するなど便利な機能が多数あるので，脚注 38)の URL にあるドキュメントを読むとよいでしょう．

3.22 目次の生成

目次は見出しをまとめたもののことで主に各章節への参照として用いられます．本文を書く際にはほとんど気にすることなく，目次に必要な情報は，\section などの見出し（3.9 節）から自動で取得されるので，皆さんは目次を表示するコマンドを記述するだけです．

目次を表示したいところに，\tableofcontents コマンドを使用すれば目次が表示されます．

3.23 LuaLaTeX を使ってみよう

以上で，LaTeX の使い方はひととおり説明してきました．以上での説明で前提としていたのは，pLaTeX です．しかし，最近では日本でも LuaLaTeX が用いられるようになってきました．LuaLaTeX は，今まで説明してきた pLaTeX に匹敵する組版能力を持ちます[39]．そのうえ，pLaTeX にはない，次のようなメリットを持ちます．

- DVI ファイルを経由することなく，直接 PDF が得られる．
- 組版時に Lua 言語を実行できる．
- フォント周辺の扱いが大幅に簡単になっている．
- LuaLaTeX 専用のパッケージが利用できる．

その反面，組版速度は pLaTeX に軍配が上がります[40]．また，2022 年 3 月現在では，学会のクラスファイルはまだ pLaTeX が中心のようです[41]．そのため，LuaLaTeX を活用できる機会は，日常で作成する文書（レポートなど）に限られてしまいます．

とはいえ，今まで説明してきたことは無駄にはなりません．pLaTeX で組版できる文書から LuaLaTeX で組版できる文書を作成するためには，次のような点を変更するだけで良いのです．逆に言えば，数式の組み方や図表の取り入れ方などは，LuaLaTeX でも変わりません．

38） http://www.ctan.org/pkg/geometry

39） このような高度な組版能力は，LuaTeX-ja（http://sourceforge.jp/projects/luatex-ja/）というプロジェクトによる開発の成果です

40） かつてよりかは高速に組版ができるようですが，それでも pLaTeX のほうが高速です．

41） 情報科学類の卒業研究論文のスタイルファイルの様式も，pLaTeX が前提になっているようです．

- 文書クラスを変更する．
- オプションの dvipdfmx を削除する．
- LuaLaTeX 用のパッケージを読み込む．

それぞれ，以下で順番に説明していきます．

3.23.1 前提：文字コードについて

LuaLaTeX では，デフォルトの文字コードは UTF-8 です．そのため，最初から UTF-8 でファイルを記述しておくことを推奨します．

3.23.2 LuaLaTeX で使える文書クラス

LuaLaTeX では，次のような文書クラスを利用できます．なお，ここでは日本語文書が組版できるクラスファイルのみ紹介します．

3.23.2.1 ltjs 系クラス

3.6 節で説明したような，LuaLaTeX で使えるデフォルトのクラスファイルが存在します．それが，ltjs 系のクラスファイルです（pLaTeX でのクラスファイル名の頭に “lt” をつけることで，LuaLaTeX 用のクラスファイル名になるため，ここではこのように呼びます）．lsjsarticle, ltjsreport, ltjsbook の 3 つが用意されています．

3.23.2.2 jlreq

LuaLaTeX では，jlreq というクラスファイルが用いられることも多いです．これは，「日本語組版処理の要件」を満たすような組版を実現できるように設計されたクラスファイルです．jlreq は，LuaLaTeX だけでなく pLaTeX などでも利用できます．

3.23.2.3 bxjs 系クラス

bxjs 系クラスファイルも，ltjs 系クラスファイルと同じように使える汎用クラスファイルです．通常の pLaTeX でのクラスファイル名の頭に “bx” をつけた，bxjsarticle, bxjsreport, bxjsbook の 3 つが用意されています．jsarticle, jsreport, jsbook などのクラスファイルに改良が加えられたクラスファイルです．

3.23.3 オプションについて

今まで説明してきたオプションの中で，DVI 処理系を指定するオプション（dvipdfmx など）は，忘れずに削除しておきましょう．LuaLaTeX では DVI 処理系を経由せずに PDF を出力するため，完全に意味のないオプションです．それだけでなく，誤作動を起こす可能性もあります．

同様にして，各パッケージに渡してあるオプションで dvipdfmx があれば，これも削除しておきましょう．

3.23.4 LuaLaTeX 用のパッケージを読み込む

LuaLaTeX 用のパッケージとして忘れずに読み込んでおきたいのは，luatexja パッケージ[42]です．このパッケージでは，LuaLaTeX における基本的な和文組版機能を提供します．ただし，ltjs 系クラスを使う場合は，自動でパッケージが読み込まれます（追加で読み込んでも副作用はありません）．

また，LuaLaTeX 向けに再設計されたパッケージもあります．例えば，otf パッケージは LuaLaTeX では利用できず，代わりに luatexja-otf を利用する必要があります．さらに，当たり前ですが pLaTeX 専用のパッケージは LuaLaTeX では利用できません．このようなパッケージを利用している場合は，適宜置き換え・削除が必要ですので，注意してください．

なお，pLaTeX 専用の記述を使っている場合も，ここで置換が必要です．例えば，スペースを空けるなどの目的で pLaTeX で zw と書いていたところは，LuaLaTeX では \zw とする必要があります．

3.23.5 LuaLaTeX で組版してみよう

LuaLaTeX で組版するためには，

```
$ lualatex ファイル名
```

のコマンドを実行します．すると，DVI 処理系を走らせることなく PDF ファイルが得られるはずです．

3.24 発展的な TeX の話題

この節では他の節に比べてやや発展的な TeX の話題について述べます．

3.24.1 LaTeX をインストールする

皆さんが自身のコンピュータで LaTeX の編集をしたいことがあるでしょうから，この節では macOS，Windows，Linux 系 OS について LaTeX のインストール方法を紹介します．いずれも TeX Live と呼ばれる，ありとあらゆる LaTeX の周辺ツールを同梱したものをインストールするだけです．

Windows の場合 http://www.tug.org/texlive/acquire-netinstall.html から install-tl.exe をダウンロードし，EXE ファイルを実行します．

macOS の場合 macOS の場合は，一番よいのは TeX Live をフルインストールすることです．https://tug.org/texlive/acquire-netinstall.html から，tar.gz 形式の圧縮ファイルをダウンロードし，ここに含まれている実行ファイル install-tl によりインストールします．または，macTeX と呼ばれる，macOS 環境に最適化された TeX Live をインストールしても構いません．この場合は，**必ず Homebrew 経由で，かつ --cask オプションを伴って**インス

42） https://www.ctan.org/pkg/luatexja

トールしてください．アンインストールが困難になってしまうため，**macTeX をインストーラ経由でインストールすることは推奨しません**．

Linux の場合　パッケージマネージャーで `texlive` と検索し，ヒットしたものを調べてインストールします．例えば Ubuntu では `sudo apt-get install texlive-full` とします．

TEX Live は後の節（例えば 3.13 節など）にて解説される `graphicx` パッケージといった各種のマクロパッケージ（後述）を全て含みます．この章はいくつかのマクロパッケージを紹介しますが，紹介するものは全て TEX Live に収録されているものです．従って，TEX Live のインストール後に皆さんが何かパッケージをインストールする必要はありません．

3.24.2　ε - pTEX 以外の TEX 処理系

TEX 処理系を図 3.9 にまとめました．図中の矢印は派生を示して，例えば ε - pTEX は pTEX と ε-TEX の派生であるという意味です．興味のある方はこれらについて Web などで調べてみると良いでしょう．

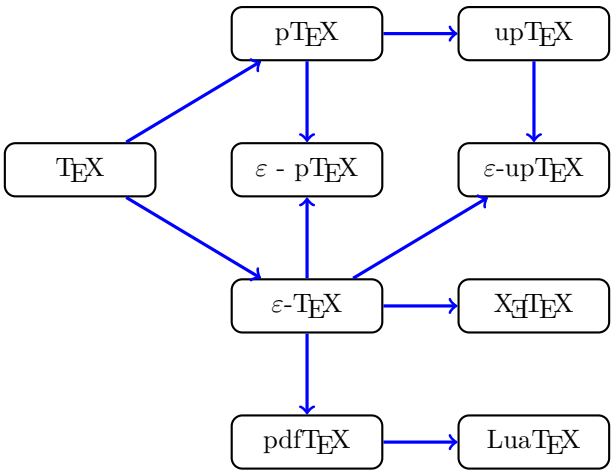


図 3.9: TEX 処理系の一部

3.24.2.1　ε-upTEX

ε - pTEX の派生として日本でよく利用されている ε-upTEX[43] という処理系があります．これは内部の文字コードを Unicode へ対応させ「髙（はしごたか）」といった特殊な文字や韓国語などを扱えるようにしてあります．

43）「いーゆーぴーてっく」もしくは「いーゆーぴーてふ」と発音します．http://sourceforge.jp/projects/eptex/

3.24.2.2 pdfTEX

TEX の派生で特に有名な pdfTEX[44]は，DVI ファイルを介さず直接 PDF を生成します．DVI ファイルを通らないので dvipdfmx を必要とせず，PDF の強力な表現力を利用できることから欧文圏では高い人気があります．ただ pdfTEX は日本語に関する組版処理が ε - pTEX と比べて劣るなどの理由から，日本ではあまり用いられていません．

注意するべきは，海外のフォーラムなどで LATEX について論じられている場合，この pdfTEX を前提としている可能性が極めて高いということです．するとソースコードをコピーしたにも関わらず，ε - pTEX では正しく動作しないということがありえます．

3.24.2.3 XETEX

XETEX はコンピュータにインストールされたフォントを直接使うことで，面倒な設定をすることなく，フォントの高度な情報を利用できます．日本語のフォントを想像するとよく分からないかもしれませんが，例えばアラビア語といった複雑な文字を美しく表現したい場合，TEX の処理系を改造するなどして対応するのは大変なことです．そこで XETEX はフォント情報を読み込むことで，処理系を改造することなく複雑な文字をも容易に扱えます．

3.24.3 ConTEXt

LATEX 以外にも純粋な TEX を強化するためのマクロパッケージはいくつか存在します．最近では海外を中心に，ConTEXt[45]という LATEX 以外のマクロパッケージを用いた文書も増えつつあります．LATEX とは違い，次のような文法となっています．

Listing 3.13: ConTEXt による文書の例[46]

```
\setuphead[title][style={\ss\bfd},
    before={\begingroup},
    after={John Doe, the author\smallskip%
           \currentdate\bigskip\endgroup}]

\starttext

\title{\CONTEXT}

\section{Text}
\CONTEXT\ is a document preparation system for the
\TEX\ typesetting program. It offers programmable
desktop publishing features and extensive
facilities for automating most aspects of
typesetting and desktop publishing, including
numbering and cross-referencing (for example to
equation \in[eqn:famous-emc]), tables and figures,
```

44） http://ctan.org/pkg/pdftex
45） 「こんてくすと」と発音します．http://wiki.contextgarden.net/

```
page layout, bibliographies, and much more.

It was originally written around 1990 by Hans
Hagen. It could be an alternative or complement
to \LATEX.

\section{Maths}
With \CONTEXT\ we could write maths. Equations
can be automatically numbered.

\placeformula[eqn:famous-emc]
\startformula
    E = mc^2
\stopformula
with
\placeformula[eqn:def-m]
\startformula
    m = \frac{m_0}{\sqrt{1-\frac{v^2}{c^2}}}
\stopformula

\stoptext
```

3.24.4 その他の組版処理系

ここでは，LaTeX 以外の組版処理系のうち，無償で使えるもの（ないし使えると予想されるもの）を紹介します．

3.24.4.1 SATySFi

SATySFi[47] とは，諏訪敬之により開発された組版処理システムです[48]．OCaml というプログラミング言語をベースに開発され，静的な型がついていることが特徴です．

3.24.4.2 Twight

Twight とは，和田優斗により発表された組版処理システムです．強力なグラフィック機能を備える組版処理系です．2022 年 3 月現在では，まだ一般公開されていない処理系ですが，オープンソースで公開する計画はあるようです[49]．

3.24.4.3 CSS 組版処理系

Web 技術を利用した組版処理系もあります．Vivliostyle[50] がその典型例です．

46） http://en.wikipedia.org/wiki/ConTeXt#Example_of_code
47） 英単語の satysfi と同じ読みをします．
48） https://github.com/gfngfn/SATySFi
49） https://note.com/ipsj/n/n6f6961254850
50） https://vivliostyle.org/ja/

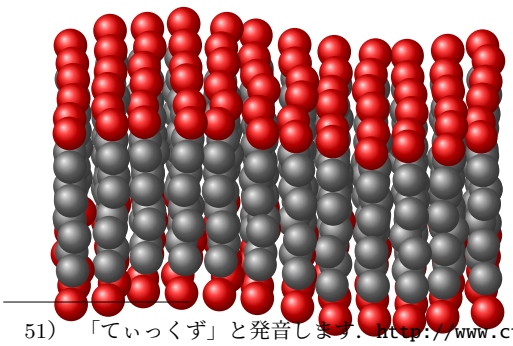
3.24.5 TikZ による図

TikZ[51] とは PGF と呼ばれる TEX 用の図形描画を目的としたマクロパッケージのフロントエンドです．TikZ を用いることで，さまざまな図をテキスト形式で記述できます．本章の図 3.2 や図 3.1 も TikZ により生成されています．

例えば図 3.2 は次のようなソースコードで作られます．

```
\begin{tikzpicture}[scale=1.5,
  block/.style = {rectangle, draw=black, thick, text width=6em, align=center,
      rounded corners, minimum height=2em},
  textblock/.style = {rectangle, text width=6em, align=center, minimum
      height=2em}]

  \draw (-1, -1) node[block] (A) {ソースファイル};
  \draw (2, -1) node[block] (B) {DVIファイル};

  \draw[->, very thick, blue] (A) -- (B);
  \draw (0.5, -1.2) node (Bd) {\epTeX};

  \draw (5, -1) node[block] (C) {PDFファイル};
  \draw (5, 0.2) node[block] (D) {PSファイル};

  \draw[->, very thick, blue] (B) -- (C);
  \draw (3.5, -1.2) node (Cd) {\dvipdfmx};

  \draw[->, very thick, blue] (B) -- (D);
  \draw (3.25, -0.3) node (Dd) {\texttt{dvips}};

  \draw[->, very thick, blue] (D) -- (C);
  \draw (5.45, -0.4) node (Dd) {\texttt{ps2pdf}};
\end{tikzpicture}
```

TikZ を説明するのはとても大変で紙面の都合もあるので，この節では TikZ によって作られた美しい図（次頁に掲載します）を TEXample.net[52] よりいくつか拝借してきました．興味のある方はぜひ脚注 52)へアクセスして，素晴しい図のソースコードを眺めてみましょう．



51) 「てぃっくず」と発音します．http://www.ctan.org/pkg/pgf

52) http://www.texample.net/tikz/examples/

図 3.10: http://www.texample.net/tikz/examples/dipolar-magnetic-field/

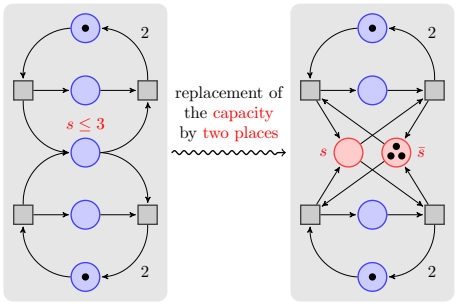


図　3.11:　http://www.texample.net/tikz/examples/nodetutorial/

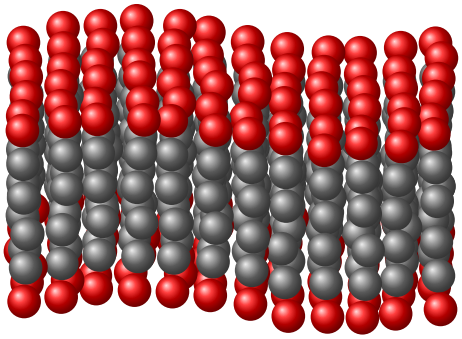


図　3.12:　http://www.texample.net/tikz/examples/membrane-surface/

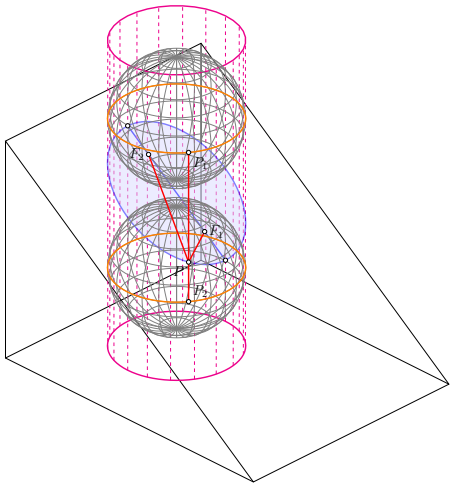


図　3.13:　http://www.texample.net/tikz/examples/dandelin-spheres/

TikZ はこのような高度な図を Illustrator といった高価なソフトウェアを用いずに，あくまでテキストベースで作ることができます．

また TEX - LATEX Stack Exchange[53] という Web サイトにて，**天下一科学系図コンテスト**[54] というスレッドが立ち，TikZ に限らずさまざまなソフトウェアを用いて作られた，素晴らしい科学系の図が投票により順位付けされています．上記の図や TEXpample.net を見て感動した方はこちらを見るのもよいでしょう．

3.24.6　古いが有益な情報

ここには，現在ではすでに古くなっているが，いまだに有益と認められる情報を置きます．過去の資料などを探す際に参考にしてください．**ここで説明する事柄は，すでに古くなっている事柄ですので**，ここでの説明の通りに操作することは，現在では推奨しません．注意してください．

53）http://tex.stackexchange.com/

54）原題は “Nice scientific pictures show off” です．http://tex.stackexchange.com/questions/158668/nice-scientific-pictures-show-off

3.24.6.1 バウンディングボックス

LaTeX で画像を用いる前に，**バウンディングボックス**という用語について理解する必要があります．LaTeX におけるバウンディングボックスとは画像の**大きさ**を示すものです．画像の大きさとは縦と横の**長さ（距離）**になります．これがなければ，LaTeX は画像の大きさが分からず画像を文書に埋め込めません．

残念なことに TeX は PNG や JPEG，PDF といったバイナリファイルから画像のバウンディングボックスを取得できません[55]．そこで PNG や JPEG，PDF といったバイナリファイルの画像を文書で用いる場合，次の 2 つの方法のうちいずれかを用いてバウンディングボックスを TeX へ渡す必要があります．

- `\includegraphics` コマンドの `bb` オプションを用いる
- `extractbb` というプログラムで生成する

bb オプションは非推奨　LaTeX について Web の資料などを調べたことがある方は，`bb` オプションを用いて `\includegraphics` コマンドへ直接バウンディングボックスを指定する方法を知っているかもしれません．

ですが，この方法は画像の“物理的な長さ”を手動で計測する必要があります．例えば皆さんが画像の情報としてよく用いる**ピクセル**という単位は，実は物理的な長さではなく，画素が何個という**数**を表す単位です．

ある物理的な長さに，どれだけの画素が詰め込まれているかという密度を表す量を解像度とよび，解像度を表す単位としては *DPI*（dots per inch: 1 インチあたりの画素数）などが存在します．

`bb` オプションを用いる場合，ピクセルといった数の情報と，解像度という画像の密度を表す情報を用いて物理的な長さを計算する必要があり，この作業は大変な労力でかつ人間の計算ミスもありえます．従って，次に説明する `extractbb` というプログラムによって算出するほうが，手作業によるミスもなく簡単でよいとされています．

extractbb によるバウンディングボックスの生成　PNG，JPEG の画像や PDF を貼る場合は，`extractbb` コマンドを用いて次のようにすることでバウンディングボックス情報ファイルが生成されます．例えば `example.png` のバウンディングボックスを生成する際は，ターミナルで次のようにします．

```
$ extractbb example.png↵
$
```

すると，画像と同じディレクトリに `example.xbb` というファイルが生成されます．このファイルにバウンディングボックスが記述されています．`\includegraphics` コマンドを用いると，画像の名前と同じ `xbb` ファイルを検索して，`xbb` ファイルがあれば利用するので，PNG，JPEG の画像や PDF を用いる際はあらかじめ `extractbb` コマンドを使ってバウンディングボックスを生成しておきましょう．

55）反対に，テキストファイルで記述されている EPS であれば，ファイルの先頭に書かれているバウンディングボックスの情報を読み込んで使うことができます．

現在はどうなっているか　現在では，バウンディングボックスを指定したり，xbb ファイルを作成したりすることは，誤動作のもとになるため非推奨になっています．最新の TEX 処理系では，バウンディングボックスを指定しなくても図版を取り込められます．すでに xbb ファイルが存在する場合は，削除しておくことをおすすめします．

参考文献

第 4 章　言語処理系

この章では COINS の計算機環境上での Python，Java，C，C++，MATLAB によるプログラムの開発方法や，実行方法を説明します．

4.1　言語処理系とは

コンピュータが動作するためには，CPU が解釈できる**機械語**の形式でのプログラムが必要になります．しかし機械語は人間には解釈しづらいため，一般的には C など人間が読みやすい**プログラミング言語**を使ってプログラムを書きます．

この言語を解釈して直接 CPU などの処理系が解釈できる言語に変換するためのプログラムがあります．機械語，あるいはそれに準ずる形式の実行ファイルを出力するタイプの処理系を**コンパイラ**（*Compiler*） といい，ソースファイルを読み出したり人間の入力を解釈してその都度命令を実行するタイプのものを**インタプリタ**（*Interpreter*）といいます．コンパイラは，**ソースファイル**（*Source File*）を入力して**実行ファイル**（*Execution File*）を生成します．このとき，ソースファイル中のプログラムのことを**ソースコード**（*Source Code*） ，コンパイラを使って変換することを**コンパイル**（*Compile*）と言います．インタプリタもソースコードを機械語に変換して実行しますが，普通は機械語のファイルを出力しないところがコンパイラと異なります．

4.2　Python

情報科学類では，プログラミング入門で学習する言語が，2019 年度より Java から Python になりました．そこで，この章では一番最初に Python について説明します．また JupyterNotebook と JupyterLab の使い方についても解説します．

4.2.1　Python プログラムの実行

Python は，**オブジェクト指向言語** *(Object Oriented Language)* です．オブジェクト指向言語とは，プログラムをオブジェクトという機能の単位で構成しようとするプログラミングスタイルのための言語です．

4.2.2　Python プログラムの作成と実行

Python プログラムのソースファイルの拡張子は “.py” です．

試しに hello.py というファイルを作成し、標準出力に “Hello, world!!” と表示する Python プログラムを作成してみましょう．

まず hello.py を作成するために、コマンドを用いて以下のようにファイルを作成しましょう.

```
$ touch hello.py↵
```

ファイルを作成できたら, 以下のプログラムをファイルに記述してみましょう.

Listing 4.1: hello.py

```
1 print("Hello,␣world!!")
```

このプログラムを実行するためには， python コマンドを用いて以下のようにコマンドを入力します.

```
$ python hello.py↵
```

これで標準出力に “Hello, world!!” と表示されます.

また対話的にも Python コマンドを実行できます. 引数に何も渡さずに python コマンドを実行すると，対話環境が起動します.

```
$ python3↵
Python 3.11.5 (default, Aug 24 2023, 14:38:34)
[MSC v.1936 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

この状態で Python の式を評価できます.

```
>>> 1 + 2↵
3
>>> print("Hello")↵
Hello
```

quit() と打ち込むか， Ctrl-D で終了できます.

```
>>> quit()↵
$
```

4.2.3 JupyterNotebook の使い方

JupyterNotebook は Python の実行環境を含んだドキュメント作成を目的とするアプリケーションです. JupyterNotebook を用いることによって，手軽に Python コードを実行し，出力を確認できます.

COINS 計算機システムにも実行環境が用意されています. ここでは使い方を解説します.

4.2.3.1 起動して`Hello`を表示してみる

Ubuntu 環境で JupyterNotebook を実行する方法を記します. 端末で次のように入力します.

Listing 4.2: notebook-start

```
1 $ mkdir $HOME/prog
```

```
$ cd $HOME/prog
$ jupyter-notebook
```

すると JupyterNotebook が起動し，ブラウザ上で JupyterNotebook のページが開きます．

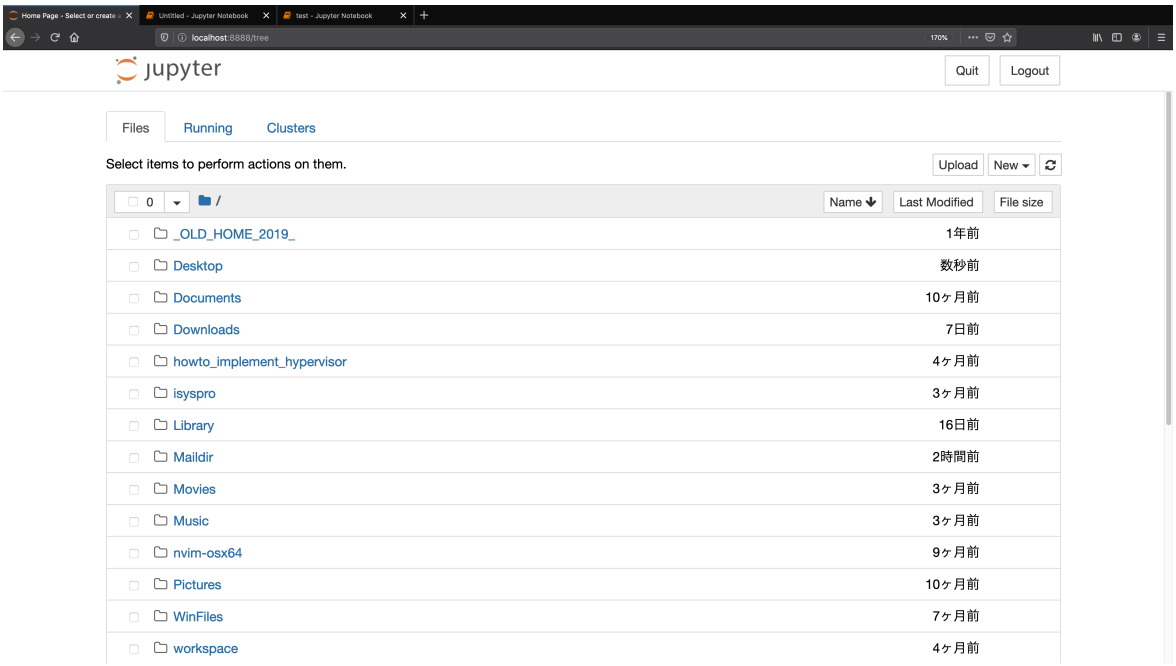


図 4.1: JupyterNotebook のホーム画面

それではノートブックを作成してみましょう．右上の`New`ボタンから`Python 3`を選択しましょう．新しいノートブックが作成されます．

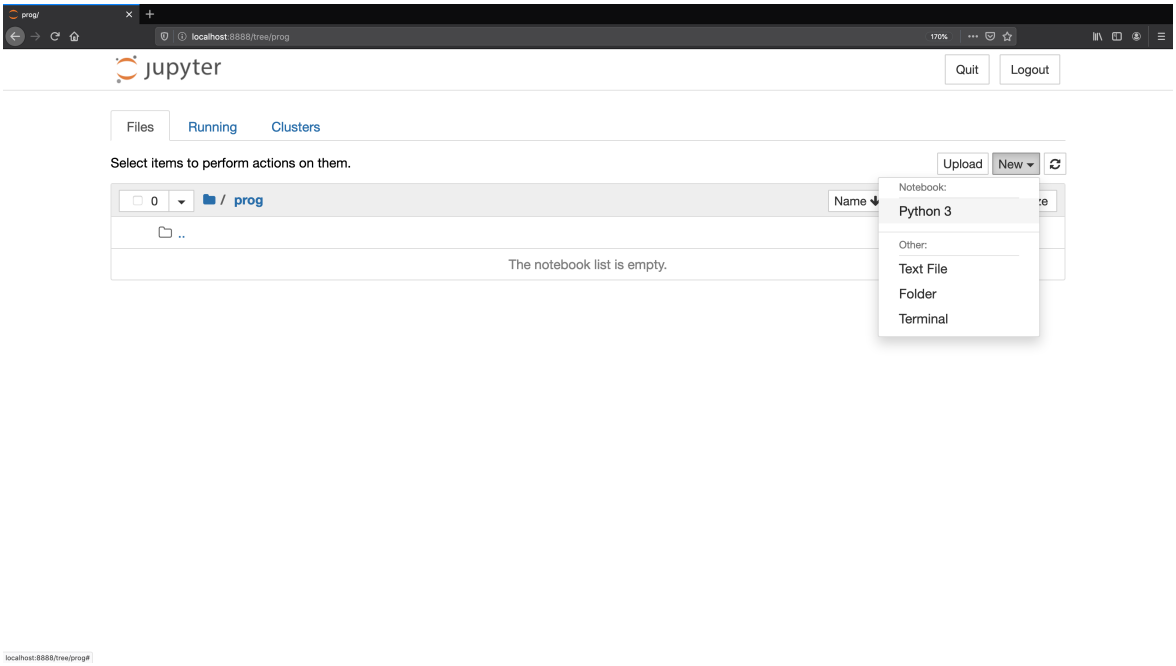


図 4.2: 新規ノートブックの作成

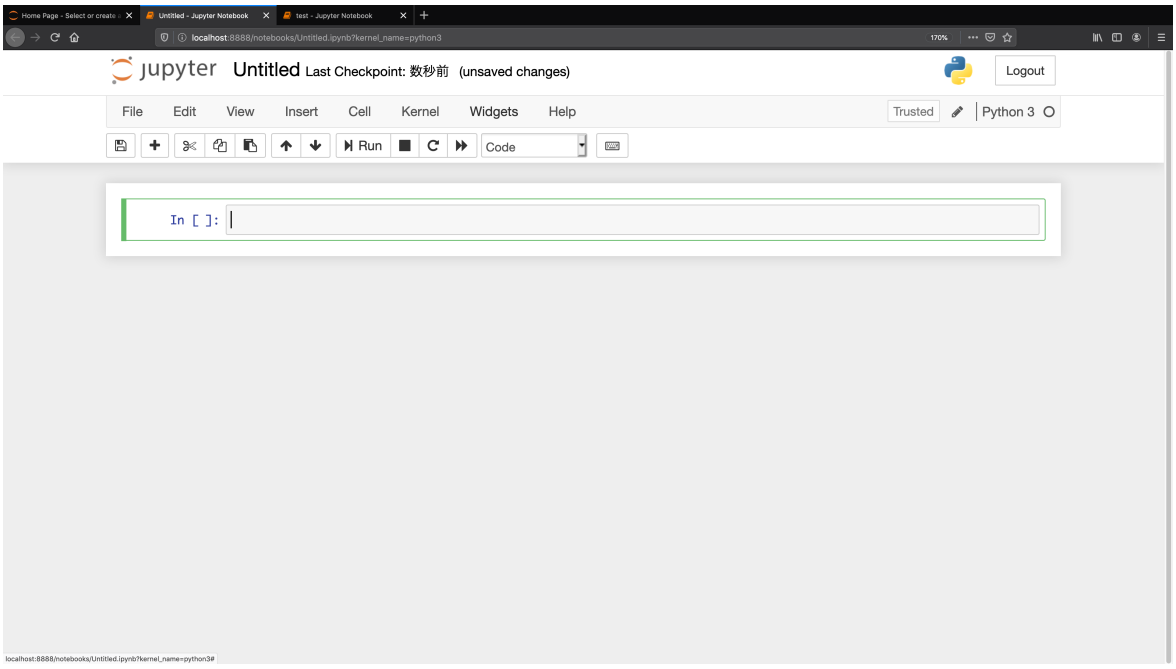


図 4.3: 新規ノートブックの画面

それではノートブックで Python プログラムを実行してみましょう．`Hello, jupyter!!`と表示するプログラムを書きます．

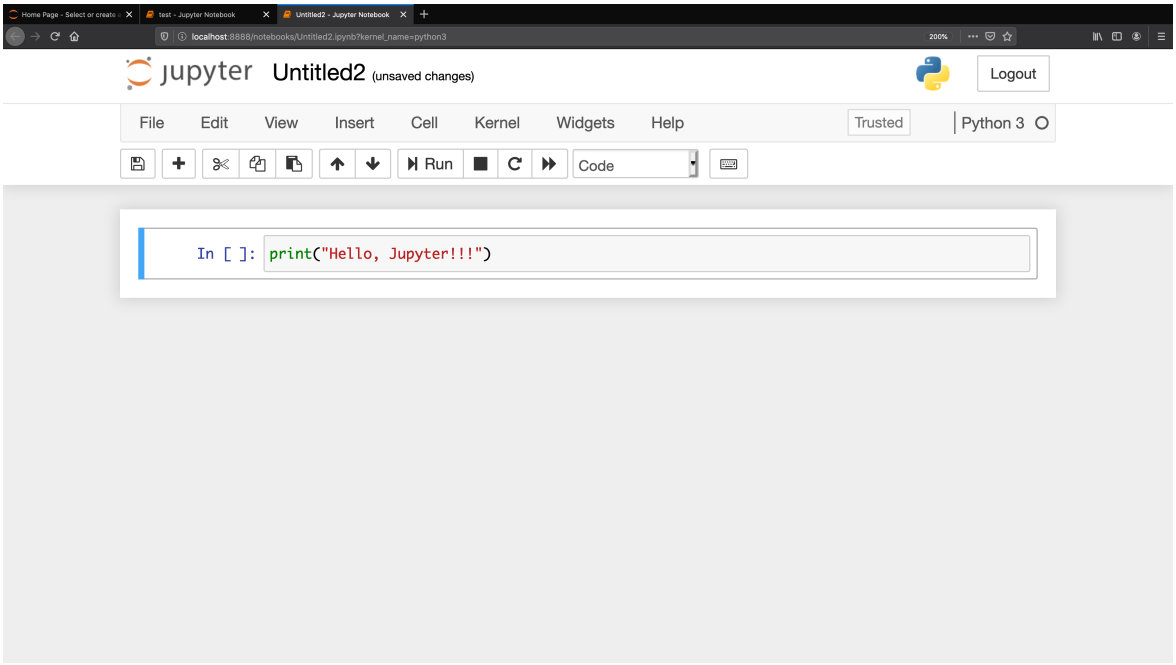


図 4.4: ノートブックにコードを書く

コードを書いたら`Ctrl + Enter`で実行します．

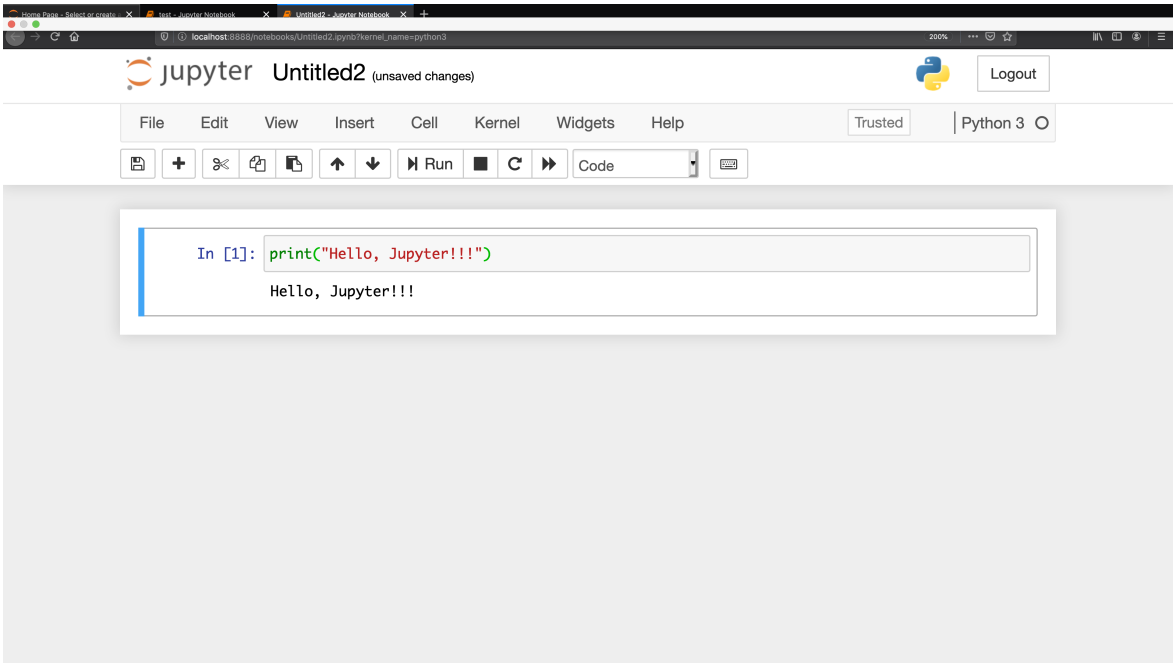


図 4.5: ノートブックのコードを実行する

コードの下に`Hello, jupyter!!`と表示されました.

4.2.3.2 グラフを表示してみる

`matplotlib`などを用いてグラフなどを表示できます.

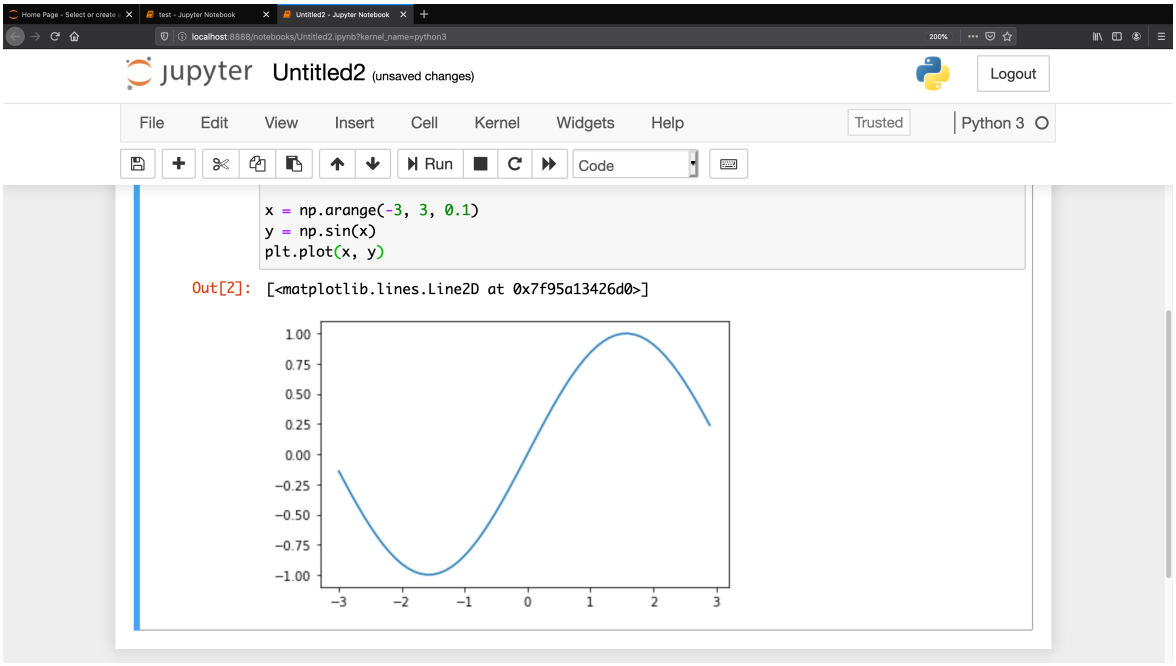


図 4.6: matplotlib のグラフを表示する

4.2.3.3 Markdown を書いてみる

Markdown とは手軽に文章構造を明示できる文章記法です．簡単で覚えやすいので幅広く用いられています．

JupyterNotebook はドキュメント作成のためのアプリケーションなので Markdown で文書作成ができます．

まずはセルのタイプを Markdown にしましょう．

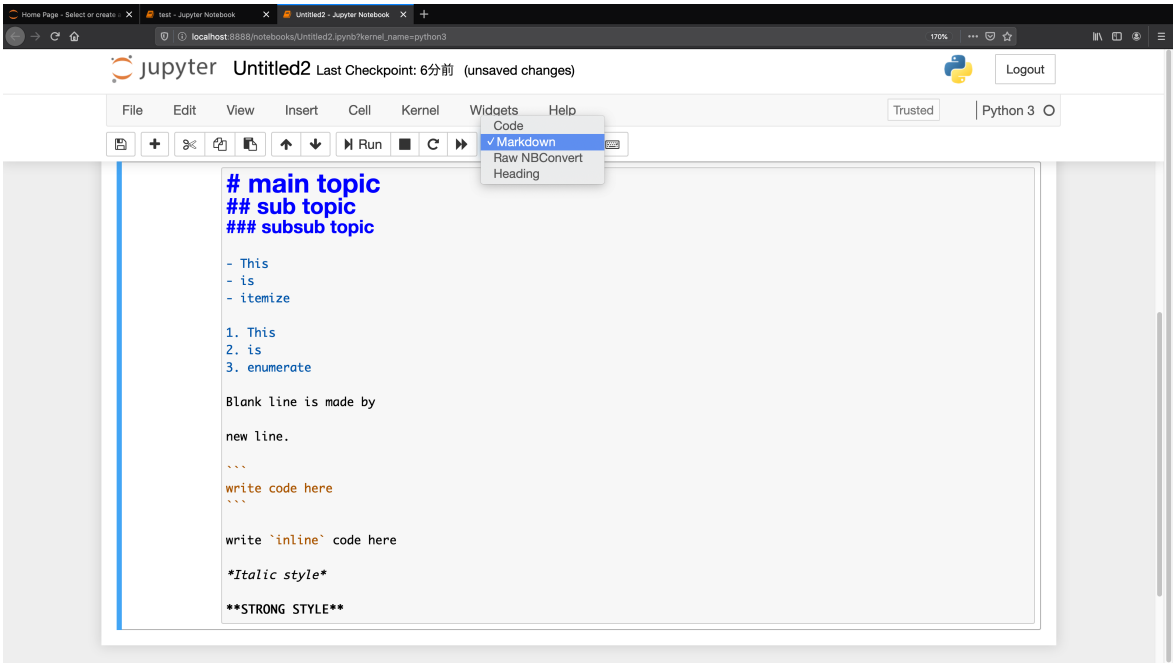


図 4.7: セルのタイプを Markdown にする

コード 4.3 を入力し，`Ctrl + Enter`を押すと，図 4.8 が生成されます．

`#`で見出しを明示します．`#`の数が多いほど見出しの重要度が下がっていきます．

`-`で箇条書きを明示します．`数字.`で番号付き箇条書きを明示します．

空行で改行を明示します．

‘‘‘で文章を囲むとコードブロックを明示できます．‘で文章を囲むとインラインコードを明示できます．

`*`で文章を囲むとイタリック体を明示できます．`**`で文章を囲むと強調を明示できます．

Listing 4.3: Markdown を書いてみる

```
# main topic
## sub topic
### subsub topic

- This
- is
- itemize

1. This
2. is
3. enumerate

Blank line is made by
```

```
14
15 new line.
16
17 ‘‘‘
18 write code here
19 ‘‘‘
20
21 write ‘inline‘ code here
22
23 *Italic style*
24
25 **STRONG STYLE**
```

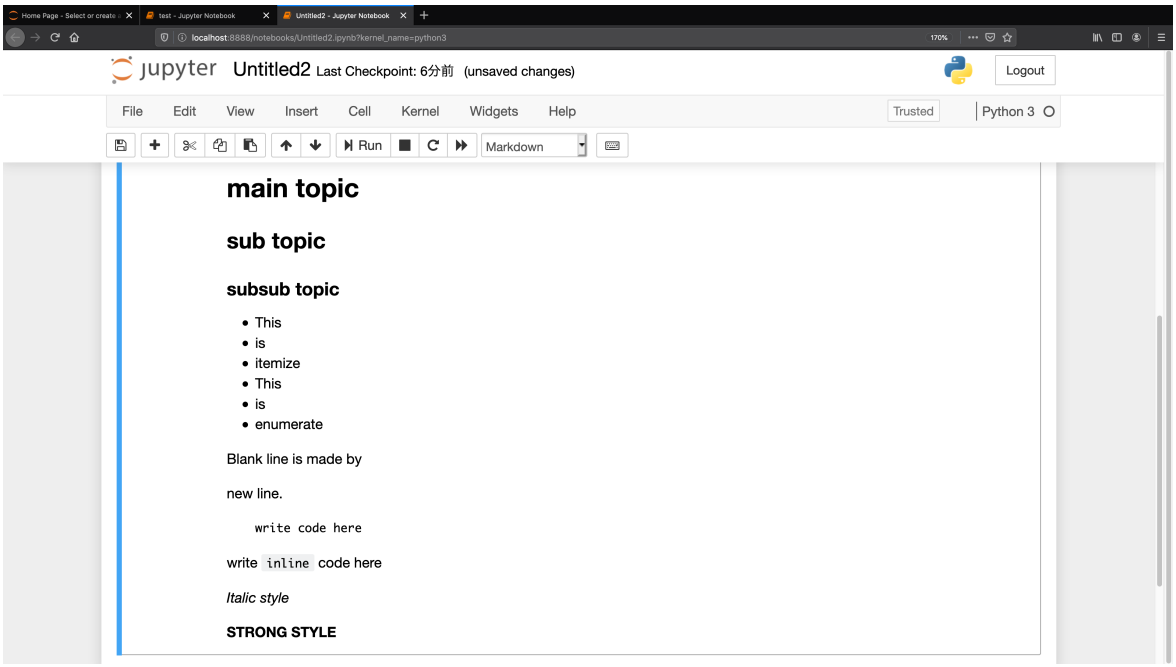


図 4.8: セルのタイプを Markdown にする

4.2.3.4 ファイル名の修正・保存・終了

ファイル名の修正・保存・終了方法を記します．まず左上の file のボタンをクリックします．

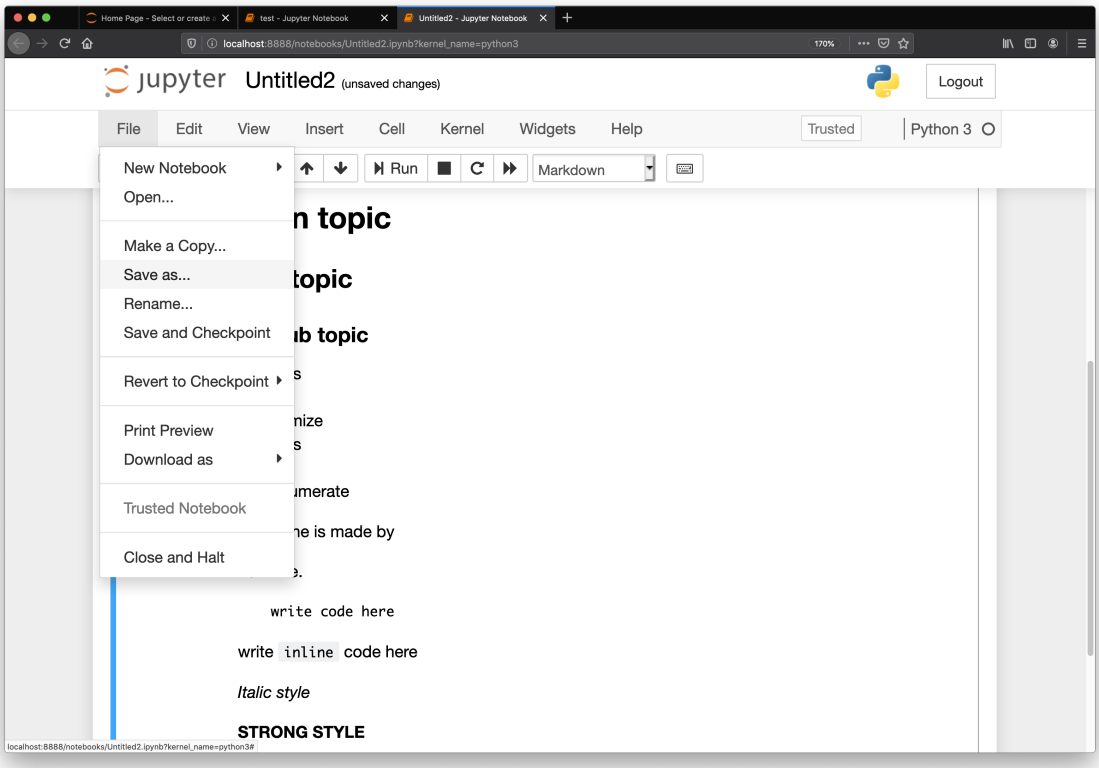


図 4.9: ファイルメニューを開く

ここで

- Rename でファイル名を修正
- Save and Checkpoint でファイルを保存
- Close and Halt で終了

ができます.
他にも Save as をクリックすることで名前を付けて保存ができます.

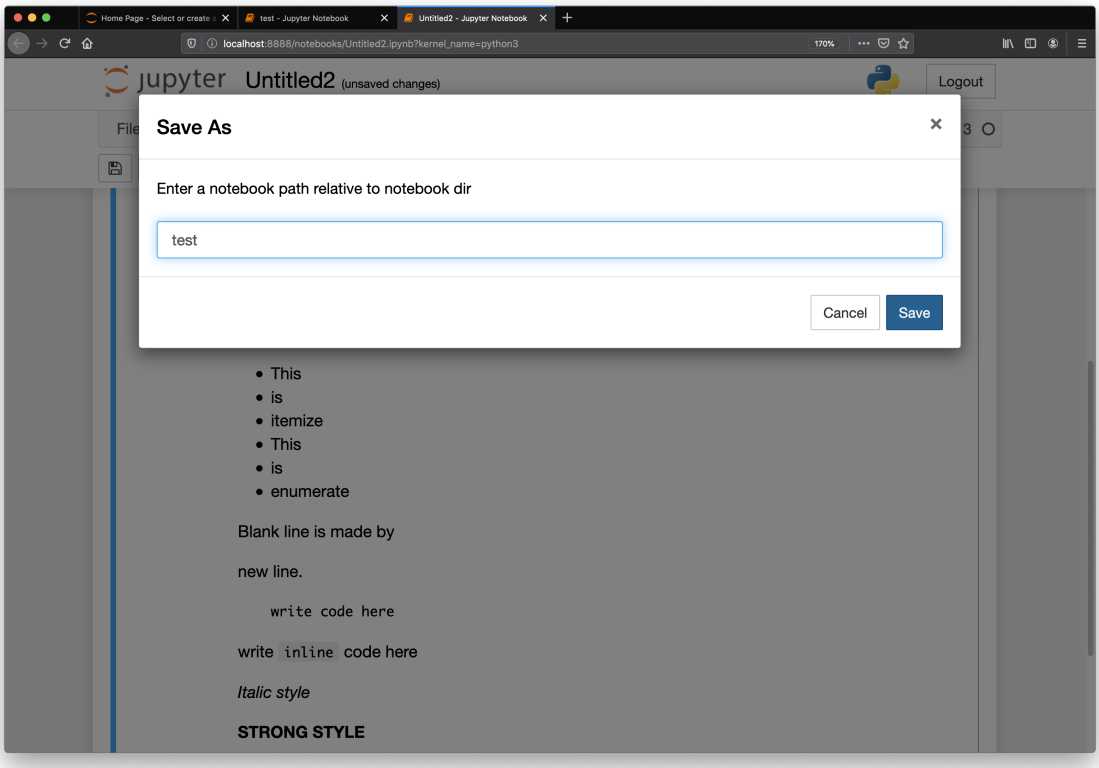


図 4.10: ファイル名を入力して保存

また JupyterNotebook を起動した端末で Ctrl-C を入力すると，Shutdown this notebook server (y/[n])? と出てくるので，y を入力して Enter を押すと JupyterNotebook サーバがシャットダウンされます．

その後ブラウザのタブを閉じることでも終了できます．

Listing 4.4: 端末から終了する方法

```
^C
Shutdown this notebook server (y/[n])? y
[C 20:41:40.854 NotebookApp] Shutdown confirmed
[I 20:41:40.855 NotebookApp] Shutting down 0 kernels
[I 20:41:40.855 NotebookApp] Shutting down 0 terminals
$
```

4.2.4 JupyterLab

JupyterLab は Jupyter プロジェクトにおける次世代の web ベースユーザインターフェースです．JupyterNotebook の機能を強化し，使い勝手を良くしたものです．

JupyterNotebook との違いとしては，複数のノートを同一画面で開ける，タブを自由に配置できるなどがあります．

ここでは JupyterLab の使い方を説明します．

4.2.4.1 起動方法

端末で次のコマンドを入力します．

Listing 4.5: notebook-start

```
1 $ cd ~/prog
2 $ jupyter-lab
```

4.2.4.2 新しいノートの作り方

JupyterLab を起動すると，図 4.11 に示すように，Launcher というタブが開きます．その中に次の項目があります．

- Notebook
- Console
- Other

Notebook をクリックすると，新しいノートが作られます．

また，上部のメニューバーの`File -> New -> Notebook`からも作成できます．

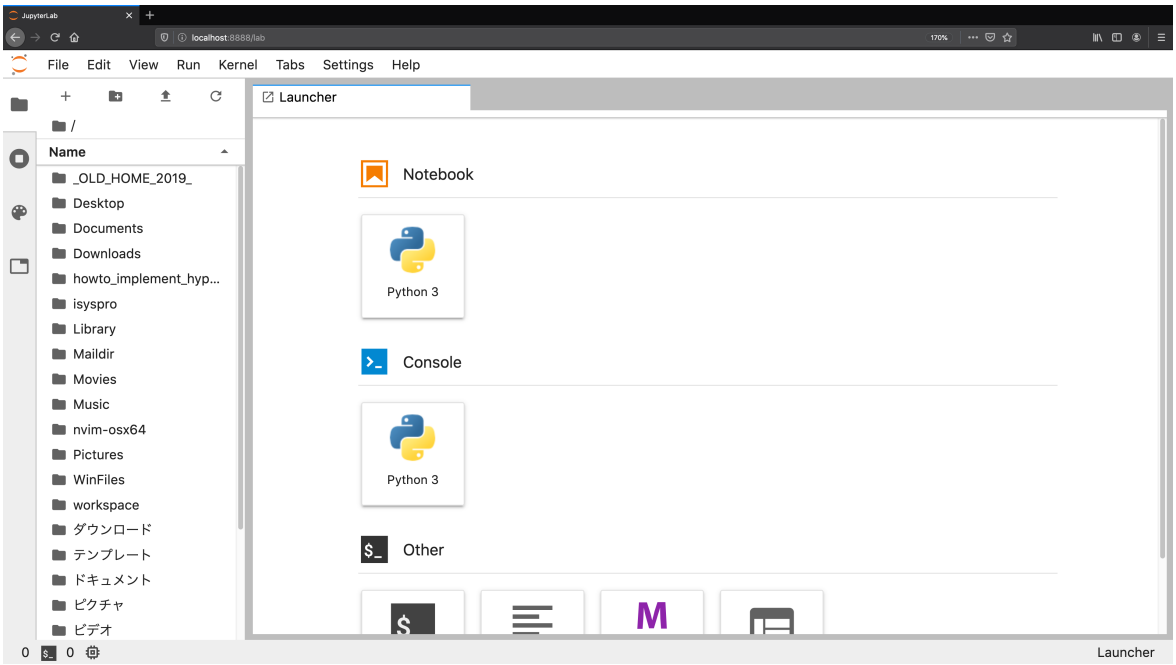


図 4.11: 新しいノートの作成

4.2.4.3　プログラムの作成と実行

使い方は JupyterNotebook と同じです．図 4.12 に示すように，セル内にプログラムを書き，`Ctrl+Enter`で実行します．

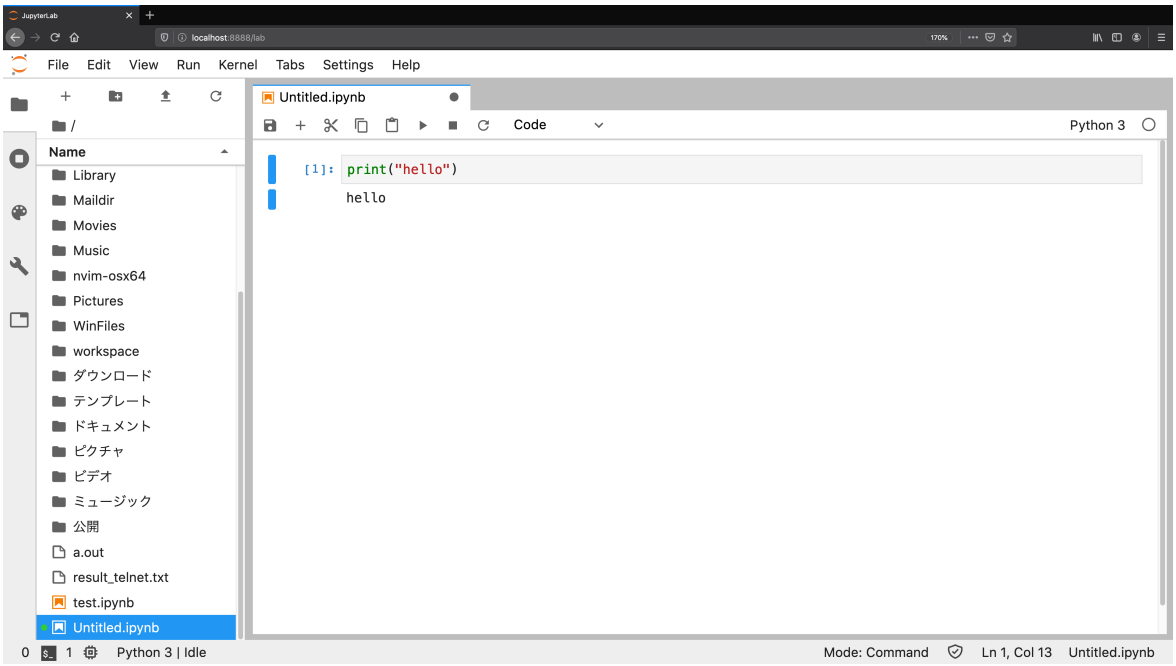


図 4.12: プログラムの実行

タブ内のメニューバーで`Markdown`を選べば Markdown モードになります．

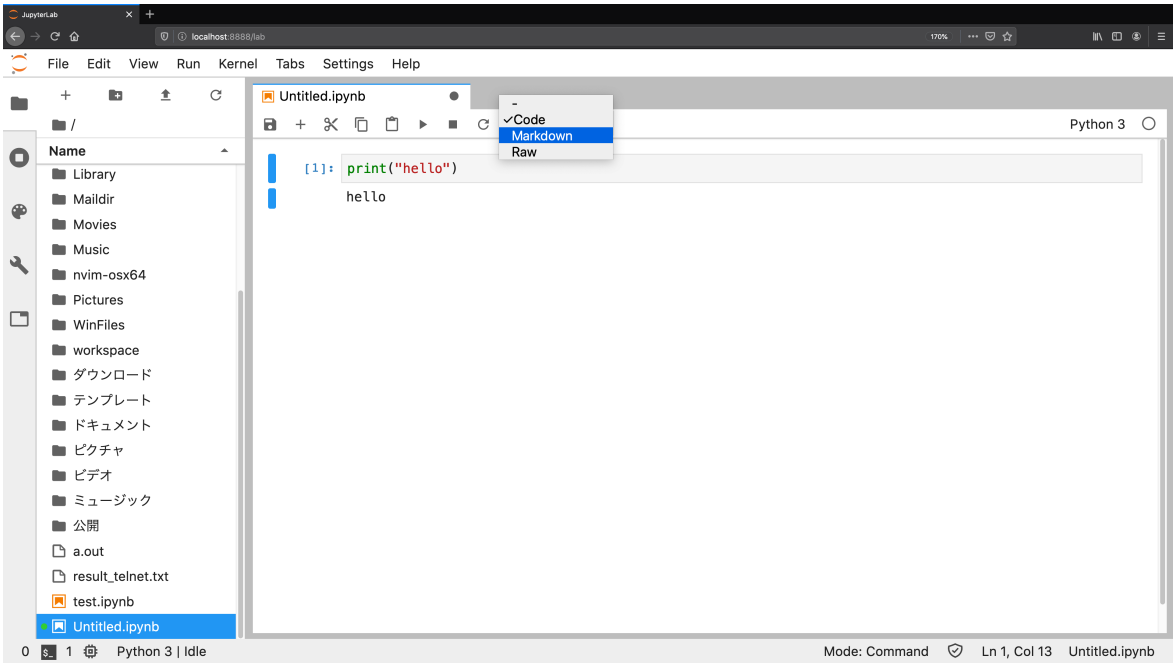


図 4.13: Markdown モード

4.2.4.4 複数のタブを開く

上部のメニューバーの`File -> New -> Notebook`から新しいタブを開くことができます.

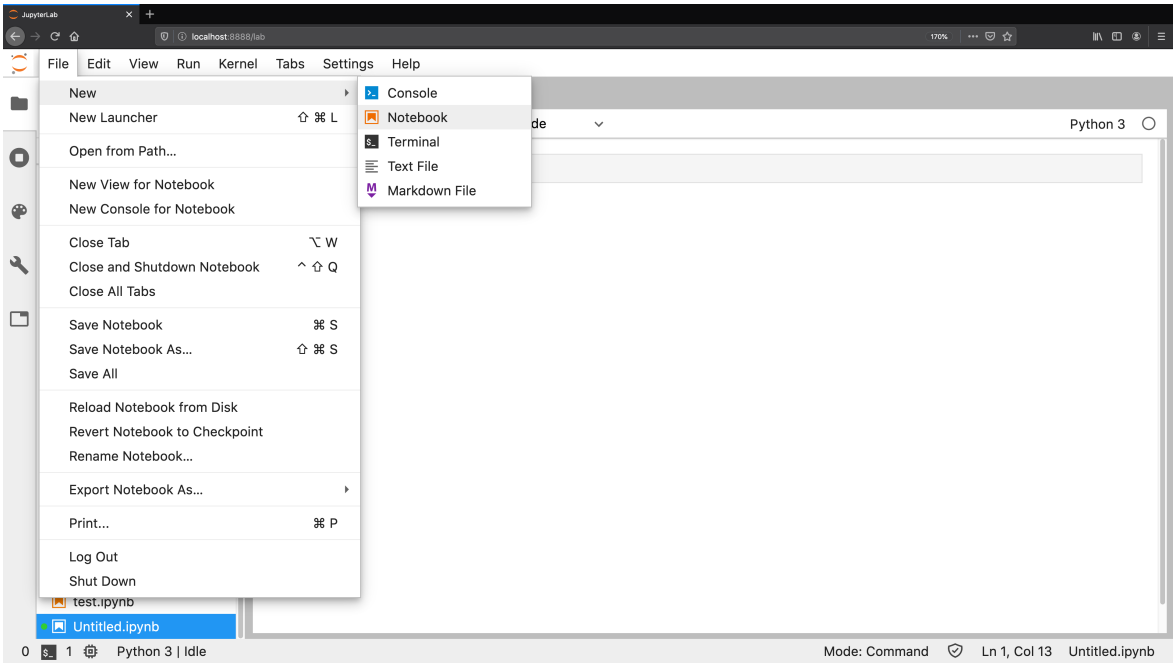


図 4.14: タブを複数開く

またタブは自由に配置できます.

4.2.4.5 ファイル名の変更・保存・終了

現在のノートブックのファイル名の変更・保存・終了の方法を示します. 上部のメニューバーの `File`からメニューを開き,

- Rename Notebook でファイル名の変更
- Save Notebook でファイルの保存
- Shut Down で終了

ができます.

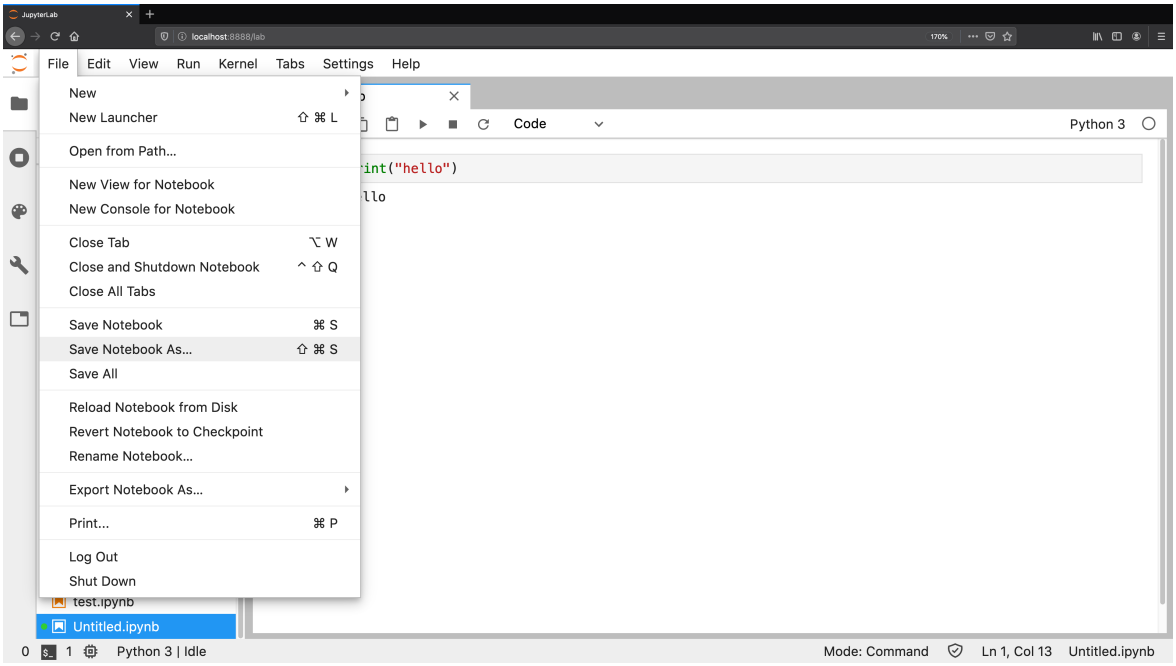


図 4.15: File メニューから操作を選択

4.3 Java コンパイラ

Java は，**オブジェクト指向言語** *(Object Oriented Language)* です．

Java のプログラムは高いポータビリティを持っており，同じプログラムを Mac, Windows, Linux いずれでも実行できます[1)]．

Java のソースファイルのファイル名は，ファイル中で宣言されているクラスの名前に，".java" の拡張子をつけたファイルにしなければなりません．例えば，MyTest というクラスの定義を含むソースファイルは，MyTest.java というファイル名でなくてはなりません．

まず，サンプルとして "Hello, Java World!" と表示するためのプログラムを紹介します．このプログラムは，以下の通りとなります．

Listing 4.6: MyTest.java

```
1 public class MyTest {
2     public static void main(String args[]) {
3         System.out.println("Hello,␣Java␣world!");
4     }
5 }
```

1) Android も Java が機能するプラットフォームとして有名です．

このプログラムをコンパイルするためには，javac コマンドを用いて以下のようにコマンドを入力します．

```
$ javac MyTest.java↵
```

これにより，MyTest.class という名前の**クラスファイル** *(Class File)* が生成されます．クラスファイルとは，ソースファイルをコンパイルした結果のバイトコードが書かれています．バイトコードは，Java の実行環境が解釈して実行します．この実行環境のことを Java Runtime Environment（JRE）と言います．

Java の実行環境を使ってプログラムを実行させるためには，java コマンドを使用します．引数には，main 関数の入っているクラスのクラス名を指定します．引数に指定するのは，あくまでもクラス名のみで，クラスファイルの名前ではないことに注意します．

MyTest クラスを実行させる場合には次のようにします．

```
$ java MyTest ↵
Hello, Java world!
$
```

4.4 C

C によるプログラミングの方法を説明します．デバッグ方法を解説します．ここでは，C プログラムのより高度なコンパイル方法を紹介します．

4.4.1 C コンパイラ

C **コンパイラ**（*C Compiler*）は C で書かれたソースコードをもとにして実行ファイルを出力します．以下，C コンパイラを用いてソースファイルをコンパイルし，実行する手順を説明します．

C のソースファイルには，“.c” という拡張子をつけます．例えば，sample.c というファイル名とします．ソースファイルは，Emacs や vi などのテキストエディタで作成してください．

まず，ターミナルに “Hello, C world!” という文字列を出力する簡単なソースコードを示します．ソースファイルのファイル名は，test.c としてください．左側に書かれている数字は行数を示していますが，説明のために便宜上つけてあるだけで，実際のソースコードには入力する必要はありません．

Listing 4.7: test.c

```
1 #include<stdio.h>
2
3 int main()
4 {
5     printf("Hello,␣C␣world!\n");
6     return 0;
7 }
```

利用しているターミナルによっては，ソースコードにあらわれるバックスラッシュ（\）が円記号になる場合もありますが，これは同じ ASCII コード（0x5C）が割り当てられていることによって起こります．強制的にバックスラッシュを入力するには，⌥+ě を同時に押します．

このソースファイルをコンパイルするためには，gcc というコマンドを使ってコンパイルを行います．gcc とは，GNU Compiler Collection（gcc）を実行するためのコマンドです．gcc は Free Software Foundation という組織がオープンソースで提供しているコンパイラで，様々なコンピュータ上で動作します．gcc を使ってコンパイルをするために，以下のように test.c が入っているディレクトリでコマンドを実行してください．

```
$ gcc test.c↵
```

コンパイルを実行すると，a.out という名前の実行ファイルが生成されます．生成された実行ファイルを実行するには，以下のようにします．

```
$ ./a.out↵
Hello, C world!
$
```

実行すると，“Hello, C world!” という文字列がターミナルに表示されます．

gcc コマンドでオプションを指定せずにコンパイルを行うと，前述のように a.out という名前の実行ファイルが出力されます．これを任意の名前に変えるためには，gcc の “-o” オプションを使用します．例えば，hello という実行ファイルを出力したい場合，以下のようにします．

```
$ gcc -o hello test.c↵
```

4.4.2 ヘッダファイル，ライブラリ

先ほどのプログラムでは，プログラム内で文字列を出力するために printf 関数が使われています．この関数を用いるために，printf 関数が宣言されている stdio.h という名前のファイルをインクルードしています．このようなファイルを**ヘッダファイル** *(Header File)* と言います．

また，関数の宣言に対応する関数の実体は，**ライブラリファイル** *(Library File)* と呼ばれるファイルの中に存在します．ライブラリファイルは，コンパイル時に結合する必要があり，このことを**リンク** *(Link)* すると言います．標準的なライブラリファイルは自動的にコンパイル時にリンクされますが，そうではないライブラリは明示的に指定する必要があります．これについては，4.4.8 節で詳しく説明をします．

どの関数がどのインクルードファイルに宣言されているかを調べるためには，man コマンドを使ってマニュアルで調べます．調べるときには，シェルから “man (関数名)” と実行します．また，このマニュアルの中では，関数の使い方や関数の引数，オプション，注意などが載っていますので，関数を使うときにはこのマニュアルを見るようにしましょう．

例えば，sqrt 関数の man を開くと，マニュアルに以下のように載っています．書式の部分では，関数の戻り値，引数などが書いてあります．説明の部分では，この関数の使い方などが書いてあります．バグの部分では，関数を使う際の注意が書いてあるので，注意が必要です．

sqrt 関数の man

```
SQRT(3)            Linux Programmer's Manual            SQRT(3)

NAME
    sqrt, sqrtf, sqrtl - square root function

SYNOPSIS
    #include <math.h>

    double sqrt(double x);
    float sqrtf(float x);
    long double sqrtl(long double x);

    Link with -lm

  Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

    sqrtf(), sqrtl():
        _ISO99_SOURCE || _POSIX_C_SOURCE >= 200112L
            || /* Since glibc 2.19: */ _DEFAULT_SOURCE
            || /* Glibc versions <= 2.19: */ _BSD_SOURCE || _SVID_SOURCE

DESCRIPTION
    These functions return the non-negative square root of x.

RETURN VALUEE
    On succeess, these functions return the square root of x.

    If x is a NaN, a NaN is returned.

    If x is +0 (-0), +0 (-0) is returned.

    If x is positive infinity, positive infinity is returned.

    If x is less than -0, a domain error occurs, ans a NaN is returned.
```

```
ERRORS
    See math_error(7) for information on how to determine whether an
    error has occured when calling these functions.

    The following errors can occur:

    Domain error: x less than -0
        errno is set to EDOM. An invalid floating-point exception
        (FE_INVALID) is raised.

ATTRIBUTES
    For an explanation of the terms used in this section, see at tributes(7).

CONFORMING TO
    C99, POSIX.1-2001, POSIX.1-2008.

    The variant returning double also conforms to SVr4, 4.3BSD, C89.

SEE ALSO
     cbrt(3), csqrt(3), hypot(3)

COLOPHON
    This page is part of release 5.10 of the Linux man-pages project.
    A description of the project, information about reporting bugs,
    and the latest version of this page, can be found at
    https://www.kernel.org/doc/man-pages/.

                                2017-09-15                      SQRT(3)
```

4.4.3 参考になる資料

- man gcc
- ハーバート・シルト著, 柏原 正三 監修「独習 C」(翔泳社)
- B.W. カーニハン/D.M. リッチー著 石田晴久訳「プログラミング言語 C (第 2 版) ANSI 規格準拠」(共立出版)

4.4.4 デバッグ

4.4.4.1 printf を使った簡単なデバッグ

ここでは，C プログラムのデバッグについて説明します．**デバッグ**（*Debug*）とは，プログラムの不具合（バグ）を修正することです．簡単なデバッグの方法としては，ソースコード中にデバッグ情報の出力を埋め込む方法があります．

簡単にできるデバッグの方法として，printf を使ってプログラム中で使用している変数などの情報を出力する方法があります．これは単にプログラム中で printf 関数を使って変数の中身を表示するだけですが，簡単にできます．しかし，この方法だとデバッグ情報を出力する部分がプログラム中に埋め込まれてしまい，デバッグが終わった後にデバッグ情報の出力部分をコメントアウトする手間がかかります．

そこで，#ifdef 文を用いてデバッグのときとそうでないときで，プログラムのコードを切替えられるようにします．これにより，デバッグの時だけ，デバッグ情報を出力することが可能になります．

次のプログラムは，#ifdef 文を使ってプログラムを切替えられるようにしたものです．

Listing 4.8: test_debug.c

```
1  #include<stdio.h>
2
3  int main()
4  {
5      printf("Hello,␣C␣world!\n");
6  #ifdef DEBUG
7      printf("This␣is␣debug.\n");
8  #endif
9      return 0;
10 }
```

test_debug.c は test.c の 6 行目〜8 行目に#ifdef 文を使ったプログラムを追加したものです．“#ifdef (マクロ名) 〜 #endif” にはさまれた部分は，(マクロ名) が定義されているとコンパイルの対象になります．もし，マクロが定義されていない場合，この部分はコンパイル時に無視されます．つまり，コンパイル時にマクロを定義することで，デバッグ用のプログラムを生成できます．

マクロを定義する方法には，#define をソースファイル中に定義する方法とコンパイル時にコンパイラのオプションで指定する方法があります．このうち，コンパイル時にマクロを指定するには，コンパイルオプションに “-D(マクロ名)” と指定します．

```
$ gcc -o test_debug test_debug.c -DDEBUG↵
$ ./test_debug↵
Hello, C world!
This is debug.
$ gcc -o test_debug test_debug.c↵
```

```
$ ./test_debug↵
Hello, C world!
```

4.4.4.2 デバッガの利用

デバッガ *(Debugger)* というデバッグをサポートするためのプログラムを使うことで，効率的にデバッグを行うことができます．ここでは，上記 2 つの方法をそれぞれ説明します．

printf を使ったデバッグの方法について説明しましたが，この方法では，デバッグしたい箇所に全部のデバッグ情報を書かなければなりません．また，短いプログラムの場合，ソースコードを見ているだけでも簡単にバグを発見できますが，ソースプログラムが長く，複雑になると，バグを見つけることが難しくなってきます．

そこで，デバッガというデバッグをサポートするツールを使います．一般的なデバッガでは，プログラムを途中で止めて変数の中身を見たり，1 行ごとにプログラムを実行したりできます．これにより，非常に効率良くデバッグを行うことができます．例えば，課題のプログラムなどでバグがどこにあるかわからなくなったら，デバッガを使ってデバッグすることで，バグを見つけることができるかもしれません．

ここでは，次の簡単な剰余計算をするプログラムをデバッグするとします．デバッガには，gdb（GNU デバッガ）を使用します．

Listing 4.9: test_gdb.c

```
1  #include <stdio.h>
2
3  int mymod(int i, int j) {
4      while(i >= j)
5          i -= j;
6      return i;
7  }
8
9  int main() {
10     int x = 20,y = 3;
11     printf("%d␣(mod␣%d)␣=␣%d\n",x,y,mymod(x,y));
12     return 0;
13 }
```

デバッガを利用するためには，コンパイル時にデバッグに必要な情報をプログラムに埋め込んでおく必要があります．そのためには，gcc に “-g” オプションをつけてソースコードをコンパイルします．

```
$ gcc -o test_gdb test_gdb.c -g↵
```

このようにして実行ファイルを生成すると gdb を利用してデバッグを行うことができます．

次に，gdb コマンドを使って，gdb を起動させます．引数には，実行するファイルのファイル名を指定します．

```
$ gdb test_gdb↵
```

起動すると，以下のように表示されます．ここから，コマンドでデバッガを操作します．

```
$ gdb test_gdb↵
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.

           （省略）

(gdb)
```

デバッガの使い方はいろいろとあるのですが，ここでは，プログラムを特定の場所で止めるブレークポイントの機能を使って，途中でプログラムを止めてデバッグする方法を例にして説明をしていきます．

ブレークポイントを設定するためには，break コマンドを使います．break コマンドの引数には，関数名や行数を指定します．このとき，複数のファイルで構成されているプログラムにブレークポイントを設定するためには，ファイル名を指定する必要があります．ファイル名を指定するには，“break (ソースファイル名):(関数名 or 行数)” というように，引数にファイル名を含めて記述します．

```
(gdb) break mymod↵
Breakpoint 1 at 0x100000e96: file test_gdb.c, line 4.
```

ブレークポイントを設定したら，run コマンドを使ってプログラムを実行します．このとき，デバッグするプログラムに実行するための引数を付けたい場合は，run コマンドの引数にその引数を入力します．

```
(gdb) run↵
Starting program: /home/gikan/tebiki/tebiki01/compiler/test_gdb

Breakpoint 1, mymod (i=20, j=3) at test_gdb.c:4
4     while(i >= j)
```

実行したプログラムは，ブレークポイントを設定した場所に到達すると，その場所でプログラムが一時停止し，gdb でコマンドを入力できるようになります．

この状態で，変数の中身を見るためには，print コマンドを使用します．例えば，変数 i の中身を表示するためには，以下のようにします．

```
(gdb) print i↵
$1 = 20
```

また，この状態で変数の中身を書き換えることもできます．その場合には，“(変数名) = (値)” と入力します．

一時停止している状態で 1 行処理を進ませたい場合には，next コマンドを使います．これにより，1 行ごとに処理を実行させていくことができます．また，next など，何度も同じコマンドを入

力しなければならないとき，毎回同じコマンドを入力するのは非常に面倒です．そのため，gdb には前の操作をもう一度繰り返す機能がついています．これを行うためには，何も入力されていない状態で ↵ を押します．

一時停止している状態では，さらに他の場所にブレークポイントを設定できます．また，設定したブレークポイントを削除するためには，clear コマンドを使います．clear コマンドの引数には，break コマンドと同じようにして，削除するブレークポイントの関数名や行数を指定します．

最後に，プログラムの実行を再開する場合には，continue コマンドを使用します．これにより，プログラムの実行を再開します．また，デバッガを終了するには，quit コマンドを使用します．

これ以外にも，条件付きのブレークポイントの設定やバックトレースの表示など，大変便利な機能があります．とくに，セグメンテーションフォルトをはじめとするランタイムエラーの追跡に，gdb はたいへん役に立つでしょう．詳細は，man コマンドや gdb 起動中の help コマンド，インターネット上の情報などを参考にしてください．

4.4.5 分割コンパイル

大きなプログラムは，いくつものソースファイルに分割して作成します．その場合，コンパイラを使ってソースファイルごとにオブジェクトファイルを作成し，最後にそれをリンクします．このことを**分割コンパイル**と言います．

まず，オブジェクトファイルを作成するために，gcc で “-c” オプションをつけてソースファイルをコンパイルします．次に，このままでは実行できないため，リンカを使ってプログラムをリンクします．これには，gcc の引数にオブジェクトファイルを指定して実行します．

例えば，次のような 2 つのソースファイルがあるとします．

Listing 4.10: test_main.c

```
1 extern void myfunc();
2 int main() {
3     myfunc("Hello, make world.\n");
4     return 0;
5 }
```

Listing 4.11: test_main.c

```
1 #include <stdio.h>
2 #include <string.h>
3
4 void myfunc(char *s) {
5     printf("(%d) %s",(int)strlen(s),s);
6 }
```

これらのファイルをコンパイルしてオブジェクトファイルを作成するには，以下のようにします．

```
$ gcc -c test_main.c↵
$ gcc -c test_lib.c↵
```

次に，生成されたオブジェクトファイルをリンクします．リンクするためには，以下のようにします（-o オプションは，プログラムの実行ファイル名を決めるためのもので，リンク自体には関係ありません）．これにより，test というプログラムが作成され，普通のプログラムと同じようにして実行できます．

```
$ gcc -o test test_main.o test_lib.o↵
$ ./test↵
(19) Hello, make world.
```

4.4.6 make を使ったコンパイル

大きなプログラムを分割コンパイルする場合，毎回すべてのソースコードをコンパイルしていたら非常に時間がかかってしまいます．そこで，*make* というプログラムを使って，前回コンパイルしたときから編集されたファイルのみを再コンパイルするようにします．

make コマンドでは，*Makefile* というソースコードをコンパイルする手順などを記した設定ファイルを作成します．例えば，先ほどの分割コンパイルの例のプログラムでは，以下のように Makefile を記述します．

Makefile

```
CC = gcc
SRC = test_main.c test_lib.c
OBJ = test_main.o test_lib.o
TARGET = test

$(TARGET): $(OBJ)
        $(CC) -o $(TARGET) $(OBJ)
```

make コマンドを使うためには，Makefile があるディレクトリで，make コマンドを実行します．これにより，編集されたファイルのみを自動的に選んで，必要なアクションを実行します．

```
$ make↵
gcc -o test test_main.o test_lib.o
$ ./test↵
(19) Hello, make world.
```

4.4.7 最適化

最適化（*Optimization*）とは，コンパイラが効率のよい実行ファイルを生成する機能です．この機能を使うことで，プログラムの実行速度が速くなったり，実行ファイルのサイズが小さくなったりします．

最適化を行うためには，コンパイル時のオプションに “-O” に続けて数字のパラメータを与えます．“-O1”,“-O2”,“-O3” の順に最適化の度合いが強くなっていきます．“-O2” のオプションをつけ

てコンパイルするためには，以下のようにします．

```
$ gcc -O2 -o test test.c↵⏎
```

4.4.8 ライブラリ

ライブラリ *(Library)* とは，ある機能を提供するための関数群をまとめたものです．この関数群をファイルにまとめたものを**ライブラリファイル**と言います．例えば，算術演算のライブラリであれば，libm.a というライブラリファイルを使用します．

ライブラリに入っている関数を使うためには，まず，その関数に必要なヘッダファイルをインクルードします．しかし，これだけだと関数の宣言のみで関数の中身がありません．そのため，コンパイル時にライブラリファイルを一緒にリンクします．

次のようなプログラムをコンパイルするとします．このプログラムは，math.h をインクルードしており，プログラム中で算術演算のライブラリに入っている sqrt 関数を使用しています．

Listing 4.12: test_math.c

```
1 #include <stdio.h>
2 #include <math.h>
3 int main() {
4     double x = 2.0;
5     double y = sqrt(x);
6     printf("%f\n",y);
7     return 0;
8 }
```

このプログラムをコンパイルするときには，“libm.a” のライブラリファイルを必要とします．コンパイル時にライブラリファイルを指定するためには，“-l” の後にライブラリファイルの “lib” と “.a” を取り除いた部分を記述します．例えば，“libm.a” というライブラリファイルであれば，“-lm” とします．以下に実際にコンパイルするときのコマンドを示します．

```
$ gcc test_math.c -lm↵⏎
```

標準のライブラリは，/lib か/usr/lib にあるという設定になっているので，gcc はこの 2 カ所以外は探しません．これら以外にライブラリがある場合は，ライブラリの絶対パス名を書くか，-L オプションを用いて，ライブラリを検索するディレクトリを指定する必要があります．

4.5 C++コンパイラ

ここでは，C++コンパイラの使い方について説明をします．

4.5.1 C++プログラムのコンパイルと実行

C++のソースファイルの拡張子は，“.c++”，“.cc”，“.cpp”，“c” とします．ただし，C のソースファイルと紛らわしいので “.c” をつけるのは，やめたほうが良いでしょう．

C の場合は，コンパイラに gcc を使っていましたが，C++の場合には g++を使います．C と C++のコンパイル方法の違いは，gcc を使うか g++を使うかということだけで，オプションなどはほとんどかわりません．

例えば，次の test.cpp をコンパイルするとします．

Listing 4.13: test.cpp

```
#include <iostream>

int main() {
    std::cout << "Hello, C++ world" << std::endl;
    return 0;
}
```

このソースファイルをコンパイルするには，以下のようにします．

```
$ g++ test.cpp↵
$ ./a.out↵
Hello, C++ world
```

これにより，a.out という名前の実行ファイルが生成されます．

C コンパイラと同様にして，“-o” オプションをつけることで，生成される実行ファイル名を指定できます．また，C の部分で説明をした gdb でのデバッグも，C と同様にして行うことができます．

4.5.2 参考になる資料

- man g++
- 高橋 航平著 επιστημη（エピステーメー）監修「独習 C++ 新版」（翔泳社）
- B. ストラウストラップ著 長尾高弘訳「プログラミング言語 C++（第 3 版）」（アスキー・アジソンウェスレイシリーズ）
- M. A. エリス，B. ストラウストラップ著 足立剛徳，小山裕司訳「注解 C++リファレンスマニュアル」（トッパン）

4.6 MATLAB

MATLAB とは MathWorks 社が開発している数値解析ソフトウェアです．C や C++，Java といったプログラミング言語では実装が困難な行列計算やベクトル演算，グラフ化等を簡単に行うことができ，数値計算やネットワークから，制御工学や電気工学といった分野にも非常に有用な機能を多数備えています．また，MATLAB Engine というライブラリを導入して，C や C++から MATLAB の機能を利用する，といったことも可能になります．MATLAB は Windows，macOS，Linux だけでなく iOS や Android でも利用できます．すべての OS での画面について操作を説明することは困難なため，ここでは Ubuntu における画面を用いて解説します．

4.6.1　起動

インストールされたプログラムの一覧から MATLAB を選択することで，ウィンドウが「コマンドウィンドウ」,「ワークスペース」等に分割された GUI で MATLAB が起動します．

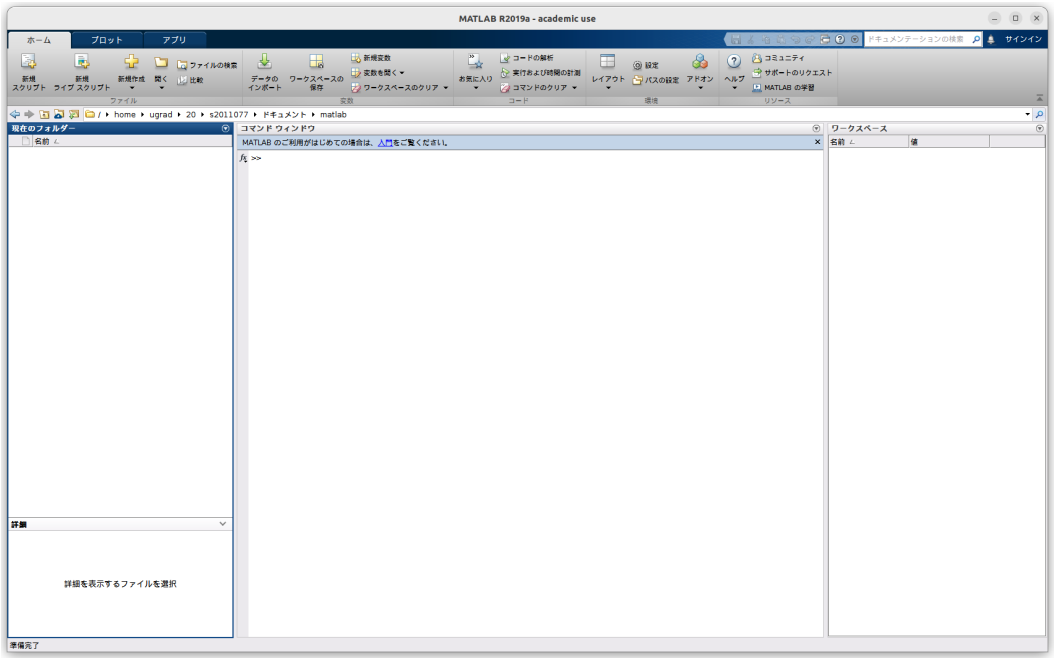


図 4.16: MATLAB

MATLAB はターミナルから利用できますが，GUI を用いて利用したほうが配列や変数の状態を確認できたり，「*Simulink*」といった MathWork 社が提供するソフトウェアと連携しやすくなりますので，ここでは GUI で操作しているものとして解説していきます．

4.6.2　計算の実行

MATLAB で変数や配列の初期化及び代入したり，それを用いた数値計算をしたりという場合は「**コマンドウィンドウ**」に記述します．変数の初期化を行う場合は，以下のように記述します．

```
a=10
```

このように記述し，⏎ を押すことで入力した内容が実行されます．この場合の実行結果は以下の通りです．これはコマンドウィンドウに表示されます．

```
a =

     10
```

MATLAB では基本的に「int」や「float」といった変数型を指定する必要はありません．上記の a に整数ではなく小数点以下を含む実数を代入してもエラーは起きません．配列の場合は列区切り

に「,」を，行区切りに「;」を用います．例えば，サイズ 2Œ2 の行列の初期化は以下のように記述します．また，記述した数式の末尾に「;」を加えると，実行後に実行結果がコマンドウィンドウに表示されません．サイズが大きい行列の計算を行う場合等，コマンドウィンドウが流れてしまうことを考えて「;」を加えたほうが良い場合もあります．

```
A=[10,10;20,20];↵
```

「;」を加えて実行結果を表示しなかった場合でも変数や配列の内容を確認する方法はあります．例えば先ほど初期化を実行した「A」の内容を確認したい場合，次のように記述すれば確認できます．

```
A↵
```

この場合の実行結果は以下の通りです．

```
A =

     10    10
     20    20
```

また，すべての変数と配列の内容は「**ワークスペース**」でも確認できます．ワークスペース内にはこれまでに利用した変数や配列名が表示され，確認したい対象の名前をダブルクリックするとコマンドウィンドウが分割されて対象の内容が表示されます．

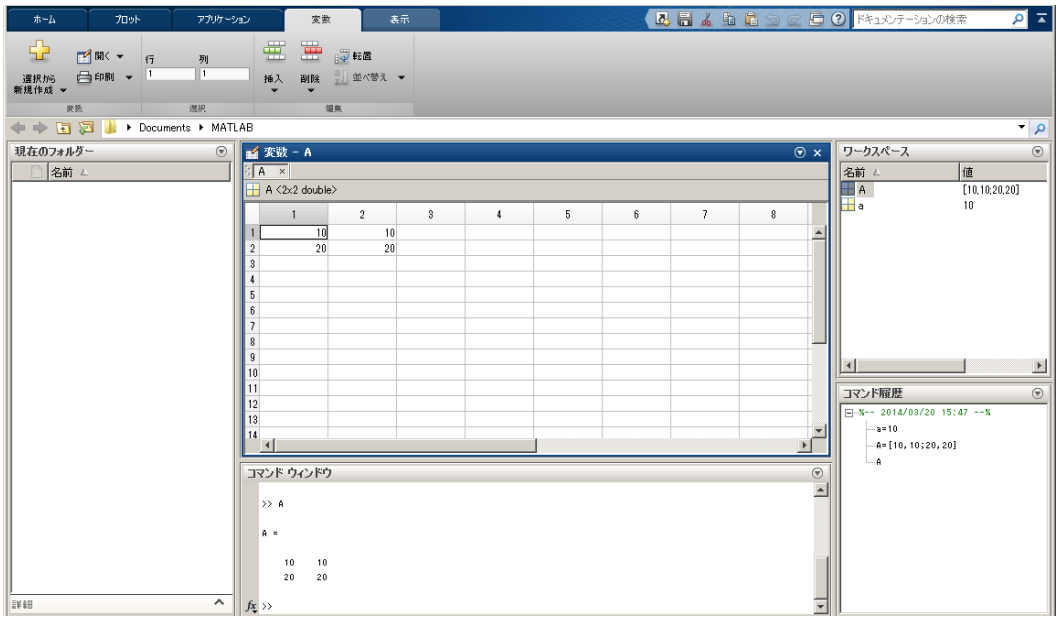


図 4.17: 変数内容の参照

節の始めでも説明したとおり，MATLAB は行列計算を簡単に行うことができます．たとえば行列の積は以下のようにして計算できます．

```
A*[1;2]↵
```

この場合の実行結果は以下の通りです．

```
ans =

     30
     60
```

C や C++で行列の積を計算する実装を考えた場合，for 文を用いて各要素についてひとつずつ計算する，といった方法がまず頭に浮かぶかもしれませんが，MATLAB では「*」を用いるだけで積を求めることができます．この実行結果で注意して欲しい点がひとつあります．この計算では計算結果を別の変数に代入していないため，実行結果のこれまで「a」や「A」にあたる部分には「ans」と表示されています．この「ans」は変数として残らず，ワークスペースにも残らないため，この後に ans の内容は確認できません．計算結果を後で確認したい場合には以下のように変数の代入式にする必要があります．

```
Answer=A*[1;2];↵
```

このほかにも逆行列や行列式の計算などが MATLAB では簡単に行うことができます．複雑な行列計算が行われるグラフ理論を学ぶ場合等に MATLAB を用いれば，学習の手助けになるかもしれません．

4.6.3 関数の実装

MATLAB ではよく利用する計算の一連の流れを関数として保存し，コマンドウィンドウから呼び出すことができます．関数を記述する場合には，まずコマンドウィンドウへ以下のように記述し，MATLAB が扱うことのできる.m ファイルを作成します．

```
edit filename.m↵
```

これを実行すると新たにエディタ画面が起動します．関数を記述する際には先頭に関数名及び引数と返り値を示す記述と，末尾に関数の終わりを示す記述が必要となります．例えば引数の値を 2 倍にして返す関数ならば以下のように記述します．

filename.m

```
function[value]=twice(Parameter)
value=2*Parameter;
endfunction
```

1 行目の value が引数を表し，twice が関数名，Parameter が返り値を示しています．引数や返り値は複数指定でき，その場合は変数名を「,」で区切ります．3 行目は関数の記述の終わりを示しています．

関数を記述し，ファイルを保存するとコマンドウィンドウで関数を呼び出すことができるようになります．例えば以下のように記述します．

```
x=twice(a)↵
```

実行結果は以下の通りです．

```
x =

     20
```

また，これまで一行ずつ記述してきましたが，複数の記述を一度に実行する方法もあります．まず，関数の記述の際に作成したような.m ファイルを作成し，エディタに目的である複数の記述を行います．ここでは「filename2.m」というファイルに，以下のこれまでに行った行列の初期化と計算を記述します．

filename2.m

```
a=10;
A=[10,10;20,20];
A*[1;2]
Answer=A*[1;2];
```

1 行目の記述はコメントです．MATLAB ではコメント行は先頭に「%」を記述します．記述後にファイルを保存し，コマンドウィンドウでファイル名「filename2.m」を記述し，実行すると記述された内容が一行ずつ実行されます．

4.6.4 その他のコマンドと関連リンク

ここではもっとも初歩的な行列演算を取り上げて解説しましたが，MATLAB では行列演算だけでなくさまざまな計算に利用できる関数が備えられています．MATLAB のその他のコマンドについては MathWorks 社公式サイトのドキュメンテーションセンター内の MATLAB 項目（https://jp.mathworks.com/help/matlab/functionlist.html）から学ぶことができます．また，冒頭で紹介した MATLAB engine についても同項目内にページが存在します（https://jp.mathworks.com/help/matlab/calling-matlab-engine-from-cpp-programs.html）．個人の計算機環境で MATLAB を用いて数値計算を行いたい場合，MATLAB と互換性を持った数値解析ソフトウェアの「*GNU Octave*」(https://www.gnu.org/software/octave/) が役に立ちます．GUI のデザインは MATLAB とは異なりますが，こちらはフリーウェアであり，C の標準ライブラリに含まれる関数の多くが実装されているため，C に慣れた人はこちらのほうが使いやすいかもしれません．ただし，MATLAB で実装されている関数のすべてが Octave にも実装されているわけではないという点に注意してください．

4.6.5 終了

最後に MATLAB を終了する方法について紹介します．終了する方法としては 2 つあります．

- 画面右上にある x をクリック
- コマンドプロンプトで quit または exit と入力

また，終了時に確認ダイアログボックスが表示されるように基本設定を以下の手順で変更するこ

とができます.

1. [**ホーム**] タブの [**環境**] セクションで [**基本設定**] をクリックします.
2. [**MATLAB**]→[**一般**]→[**確認ダイアログ**] を選択
3. [**MATLAB を終了する前に確認**] チェックボックスを選択して [**OK**] をクリック

2024 年度　情報科学類　計算機運用委員会　構成員

教員
叶 秀彩（委員長）
新城 靖
小林 諒平
塩川 浩昭
保國 惠一
技術職員
山崎 豊
小林 佳美
安田 雄司

本手引き（付録）は 2024 年度手引を元に改版され，下記の方々の協力により作成されました．

氏名
近藤 拓未
田口 瑛啓

発行日　2025.4
編集　筑波大学情報学群情報科学類計算機運用委員会