1. Raspberry Pi の起動と終了について

1. USB ポートに WiFi ドングルが接続されていることを確認（外さないこと）
2. USB ポートに マウスとキーボードを接続する
3. モニターからの HDMI 端子を Raspberry Pi の HDMI 端子に接続する
4. 電子回路からのピンを接続する

以上を確認してから、

5. 電源を接続する（バッテリーもしくは USB チャージャーから、micro USB により接続する）

必ず守ること：電源を接続するのは、機器の接続の一番最後

電源を外すのは 一番最初

電源を接続したまま、電子回路の接続変更や、HDMI 端子の抜き差しは行わないこと

以上によりいろいろな表示のあと、モニターに以下のような画面が表示されるはず(この前に画面にメッセージが出た場合はエンター・キーを押すこと)

注意: 旧版ではアカウント名 `pi` パスワード `raspberry` (表示されない)を入力し `startx` (エンター) とする

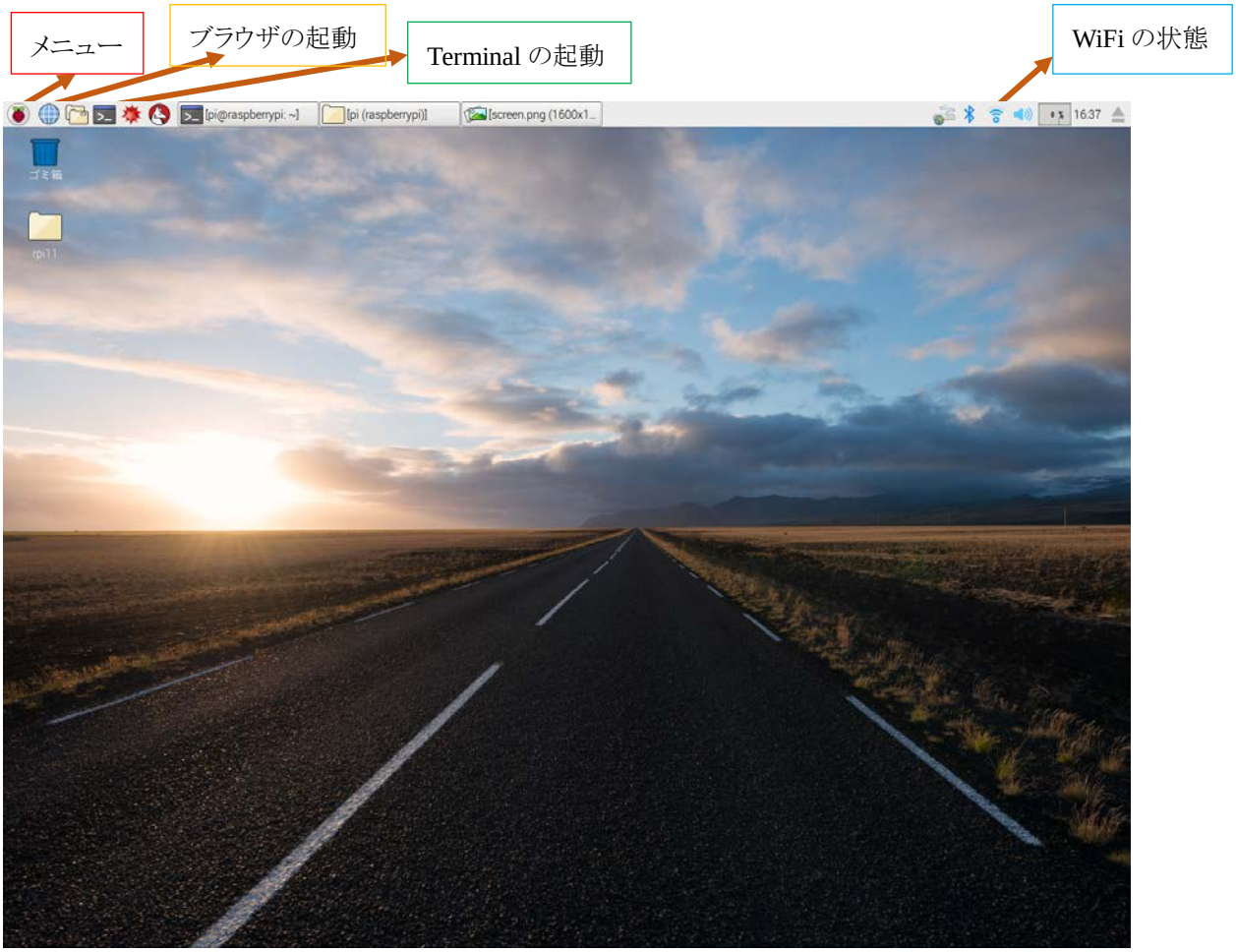

図 1. Raspberry Pi の起動後の画面

6. 終了時の手続き

(1) メニューをクリック(図 2 参照)

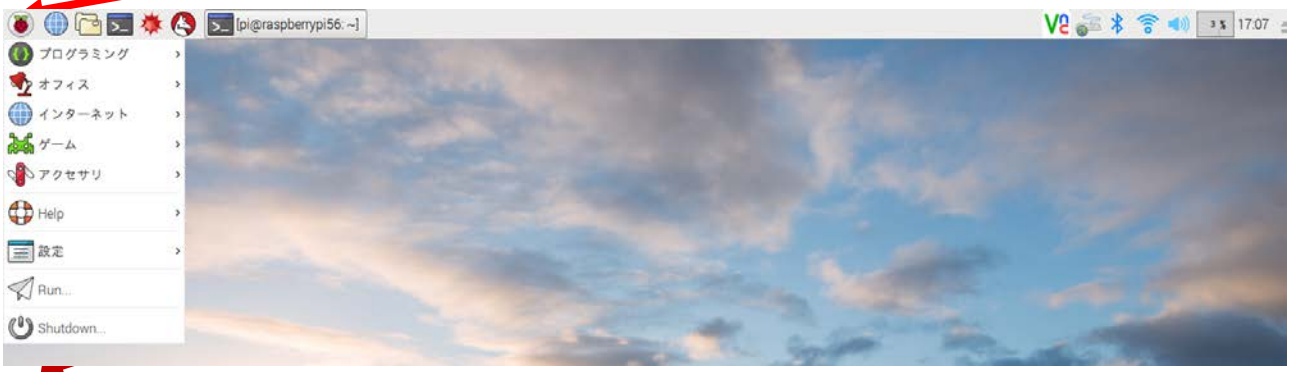

図 2. メニューをクリックした時の画面

(2) Shutdown ボタンマークをクリック （Shutdown とは「システムの停止」を意味する）

(3) 画面がブラックアウトし、Raspberry Pi 上の LED の緑色の点滅が消えたら

電源を抜く（Raspberry Pi の方を抜いても良いし、USB ケーブルのもう一端の方を抜いても良い。抜きやすい方を抜くこと。必ず端子をもつこと、ケーブルをもって抜いたりしないこと）

2. プログラムの作成と実行

Raspberry Pi は Linux という OS で動いている（Windows とは根っこでは同じですが、文化は少し違うもの）。アカウント名は pi、パスワードは raspberry である（これが要求されることはあまりないかもしれないが）。ただし、自分用のアカウントとパスワードも作ることができる。

ここでは Python というプログラミング言語を使って Raspberry Pi を動かしていく。

ここでは具体的に sample.py というファイルを作り、そのプログラムを実行することを示すことで、プログラムの作成方法と実行について、以下の手順で学ぶ。

(1) 最初にすること: Terminal の起動

(2) Python の起動と終了

(3) エディタの起動: leafpad

(4) エディタと Python の連携

(5) sample.py の作成と実行

(6) 作成したプログラムを自分用のフォルダに格納

(7) 応用: LED.py の作成と実行

(1) 最初にすること: Terminal の起動

このマークをクリックすると、Terminal が表示される

図 3. Terminal の起動

これにより、図 4 のような画面になる。中央が Terminal 画面である。画面の大きさや背景色、文字の大きさなどは調整可能である。

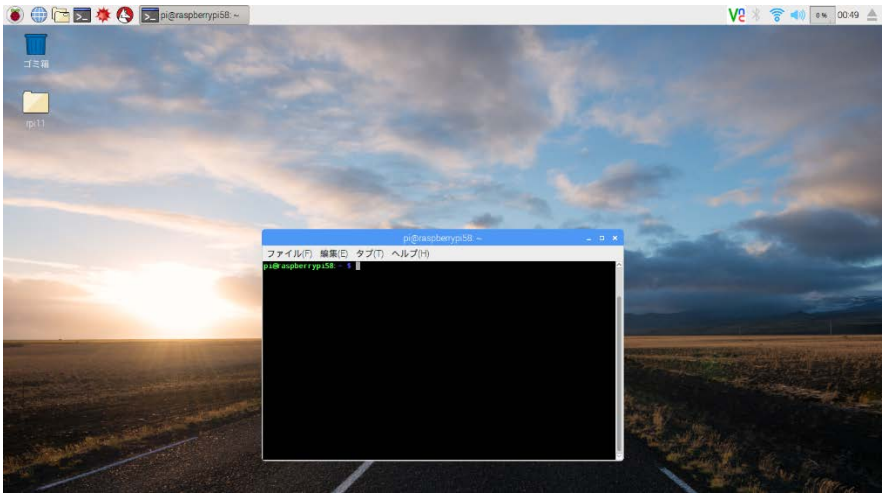

図 4. Terminal ウィンドゥの表示

なお、『メニュー』の Terminal をクリックするか、「ファイル」メニューから NewTerminal を選択すれば、図 5 に示すように、新たな Terminal 画面が表示される。こうすることで、この両方で別々の仕事を同時に行うことが可能となる（注意：左側の Terminal 画面はサイズを変更してある、p.12 の 5 では別な方法を紹介）

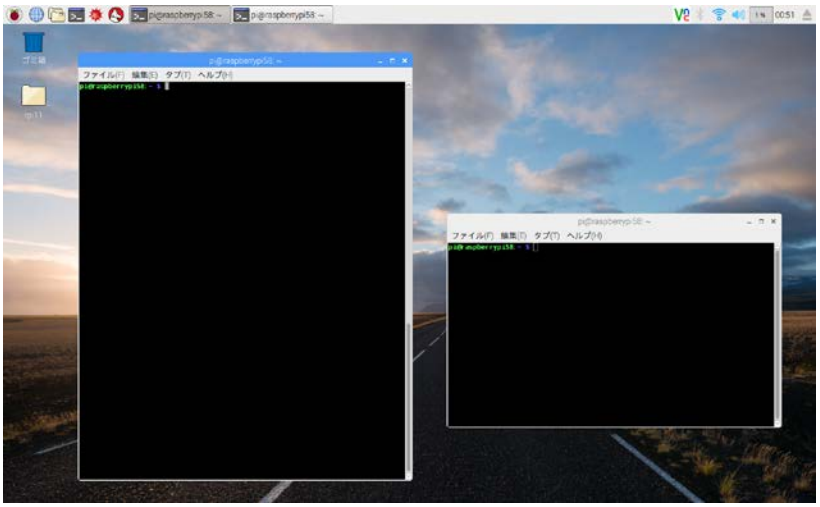

図 5.　2つの Terminal 画面を開いた状態

(2) Python の起動と終了

Python を起動するには、Terminal 画面で python と入力する(最後にエンター(Enter)・キーを押すこと)。すると、いろいろなメッセージとともに Python が起動する。図 6 は、そこで print(“Hello, world!”)と入力した状態である。なお、各行の先頭にある >>> はプロンプトといい、Python が入力待ちの状態であることを表している。

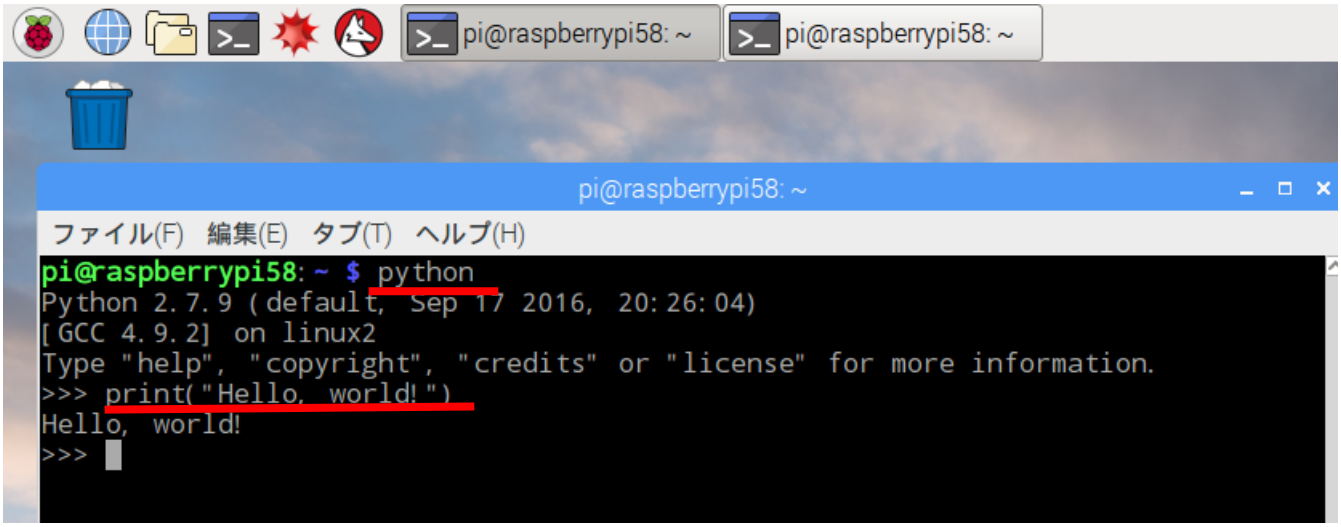

図 6. Python の起動と実行。赤下線部が入力。

Python を終了するには、 quit( ) と入力する(最後にエンター・キーを押す)。

Python で入力ミスがある場合には、矢印キーでカーソルを戻す、バックスペース・キーで文字消去、キー入力による書き換え、などができる。また ctrl-A (コントロール・キーを押しておいて a を押す)と行の先頭に、ctrl-E で行の最後にカーソルを移動させる、ctrl-K でカーソルから行末まで抹消する、ctrl-Y で抹消部分を復活させることもできる。さらに、上矢印キー(↑)を押すと前の入力が取り出せ、そこでカーソルキーを移動させ、簡単な行編集が可能である。編集した後にエンター・キーを押すと、編集後の Python コマンドが実行される。

(3) エディタの起動: leafpad

Python の中でも簡単な編集はできるが、そこで実行したことをまとめるのはやりにくい。そこで、leafpad というエディタと連携させることを考える。leafpad を起動させるには(Python を動かしている場合は Python を一旦終了させるか、もしくは別な Terminal を起動してから) Terminal 画面において `leafpad` と入力する(いつものように エンター・キーを最後に押す)。すると新たな画面が開かれる(図 7)。

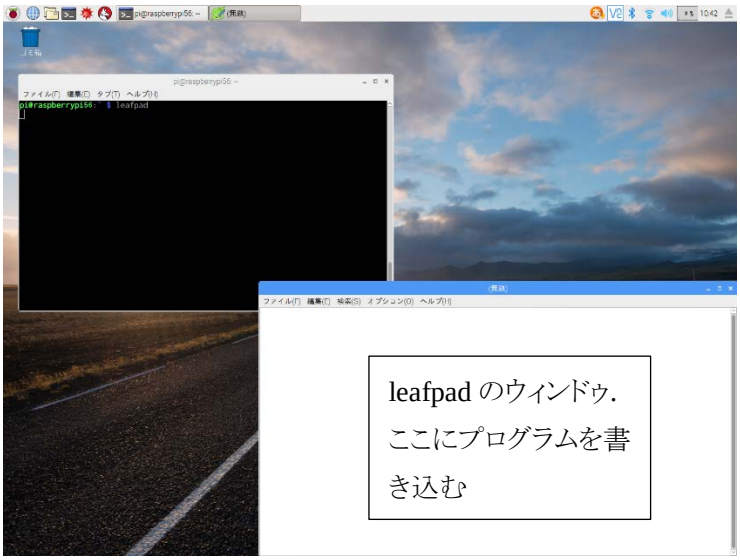

図 7. leafpad を起動したところ。右下が leafpad のウィンドウ

(4) エディタと Python の連携

leafpad を起動し、次を入力する。必ずすべて「半角(ASCII)」を用いること。空行もそのとおり入力し、各行の先頭に空白がある場合は空白の個数を揃えること。そうでないとエラーが起きる(それが Python の流儀である)

```
# sample.py
def fact(n):
  if (n <= 1):
     return(1)
  return(n * fact(n-1))

for i in range(1,10):
   print("%d! = %d"%(i,fact(i)))
#
```

ここまで leafpad に入力したら、Python と連携させてみよう。それに使うのは「コピー&ペースト」である。

マウスで def の先頭から for の前の空行まで範囲を指定し(マウスのドラッグを使用)、コピーする(範囲をしていておいてから「編集」メニューから「コピー」を選ぶ。図 8)。

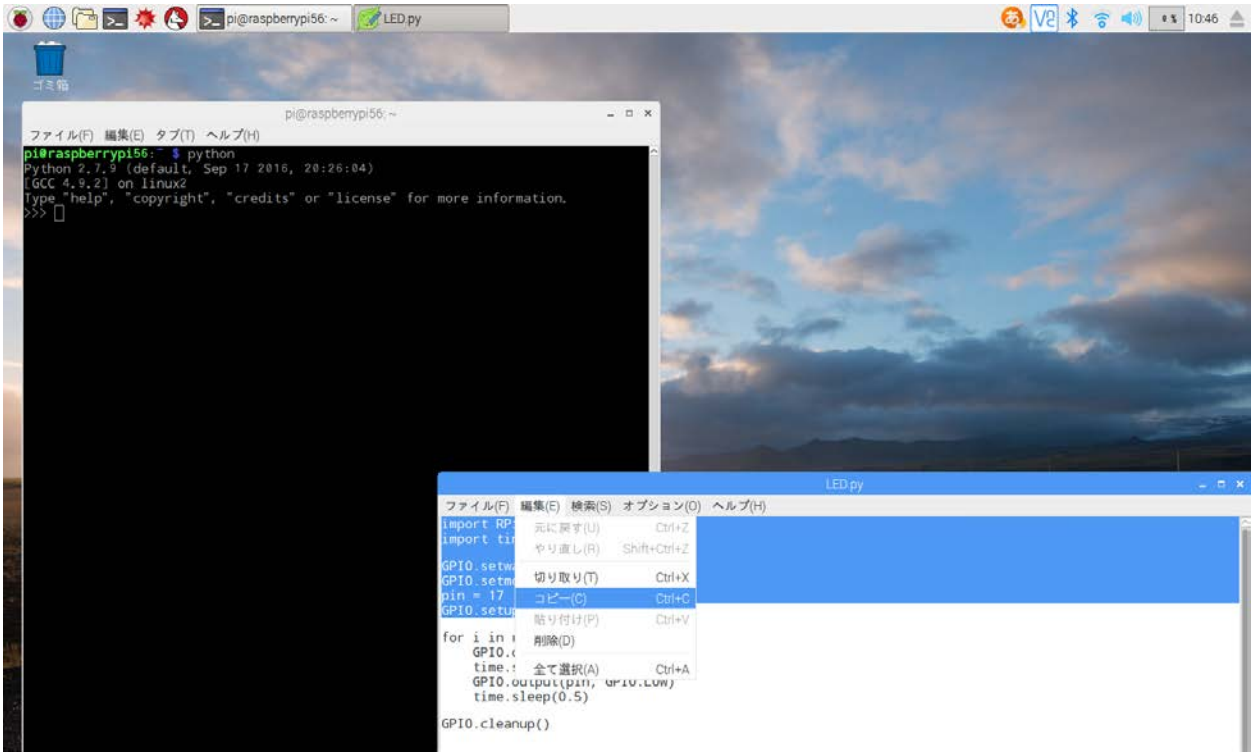

図 8.　leafpad ウィンドゥで範囲を選択(青色部分)し、編集メニューを開いて「コピー」を選択する

次に Terminal ウィンドゥにマウスを移動させ、(Python が起動していなければ)Python を起動し、「ペースト」する(編集メニューから「貼り付け」を選ぶ。図 9)。すると、Terminal 画面に fact 関数の定義が入力される。ただ、カーソルが ... のところで止まっているかもしれない。これは Python が追加の入力を待っている状態である。これに対しては、エンター・キーをおし、

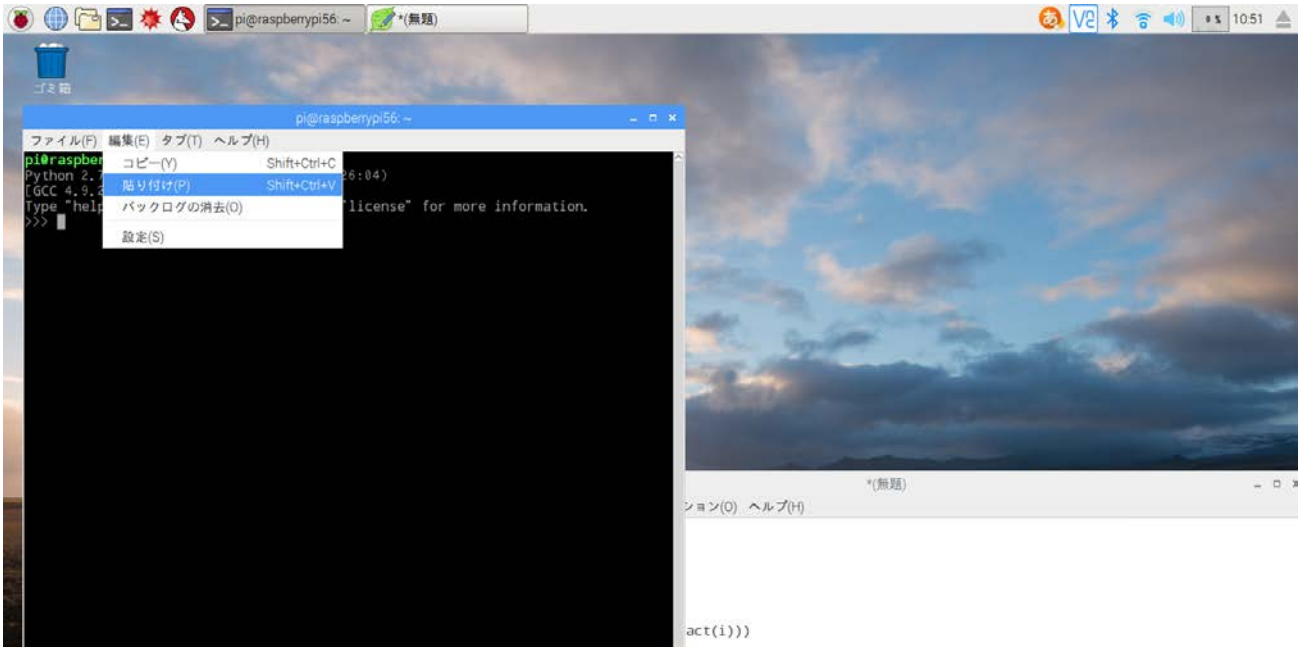

図 9. Python に『貼り付け』。Terminal の編集メニューから「貼り付け」を選択する

続けて `fact(10)`を入力しよう。

```
>>> fact(10)
```

(ここで、先頭にある `>>>` はプロンプトであり、fact(10)が入力。ただしいつものようにエンター・キーを押すこと)

これにより 3628800 という数が表示されていれば…おめでとう、あなたにとって初めての Python 関数を定義できたことになる。これは 10!（10 の階乗、1×2×3×…8×9×10）が計算されたのである。

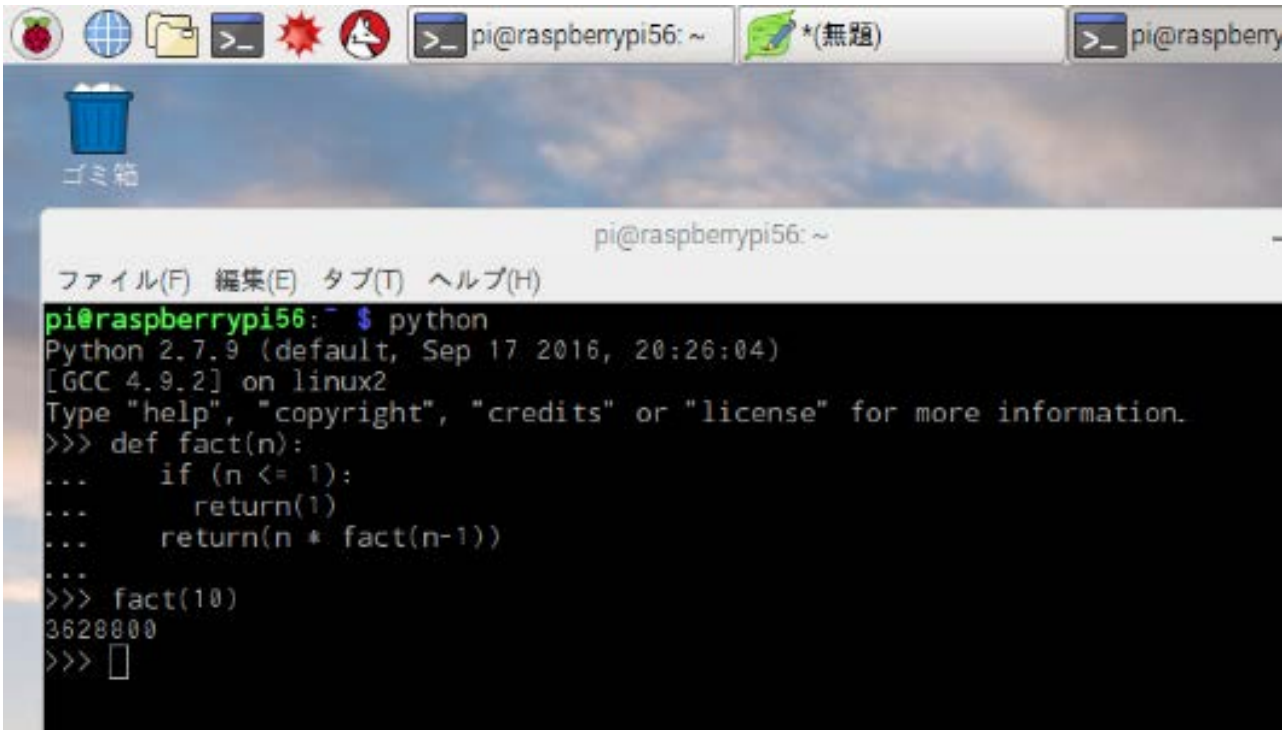

図 `10. leafpad` からコピーしたコードを `Python` に貼り付け、`fact(10)`を実行したところ

なお、エラーがあったときは、その原因を解析し、leafpad 上で修正し、再度「コピー＆ペースト」する。エラーがなければ次を実行する。

ここまでうまく行ったら、次に leafpad の for から # までをコピーし、python にペーストしよう(ただしいつものようにエンター・キーを押すこと)。ここで # は「コメント」を表す。どのような表示が得られたであろうか。
ちなみに、 `for` は C 言語の `for` 文に相当し、range(1,10)は 1 から 10 までのリスト、次の `print` は C 言語の `printf` の機能と同等のことを行っている(“の次の%が format である文字列と、その中の%d の箇所に埋め込まれる数との区切りを表している、C 言語で書くと `printf(“%d! = %d”, i, fact(i));` となる)。

```
Python 2.7.3 (default, Jun 22 2016, 03:14:32)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> def fact(n):
...     if (n <= 1):
...         return(1)
...     return(n * fact(n-1))
...
>>> fact(10)
3628800
>>> for i in range(1,10):
...     print("%d! = %d"%(i,fact(i)))
...
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
```

図 11.　Python にプログラムを貼り付け、実行した後の状況

注意：Windows（や Mac）で書いたプログラムを Raspberry Pi に持ち込むと、日本語の文字コードが違うためと、改行コードが違うために、うまくいかないことがある。これには `nkf` というコマンドで解消することが可能である。

(5) sample.py の作成と実行

以上により、Python プログラムがエラーなくちゃんと動くことが確認できた。しかし、今までは、Python コードの断片だけを動かすようなものであった。ちょっとずつコードを書いては、それが動くことを確かめていく。というこの方法は Python のプログラム開発としては王道である。ここではそうやって動くことが保証されたプログラムを「まとめて」動かす方法について説明する。それはとても簡単である。

1. `leafpad` で「保存」を選び、ここではホーム・ディレクトリ(Windows でいうフォルダ) pi に sample.py という名前で保存する。

2. Terminal ウィンドゥで、(Python が動いていれば、一旦終了させる)、/home/pi が作業用ディレクトリになっていることを確認する(`pwd` コマンドを実行し、 /home/pi となっていればよい。そうでなければ `cd` を入力すること)。

そして、 python sample.py を入力する(いつものようにエンター・キーを押すのを忘れないように)。すると、先程みた表示が Terminal ウィンドゥに表示されるはずである。

以上、Python プログラムの二通りの実行方法について紹介した。

(6) 作成したプログラムを自分用のフォルダに格納

このようにして次々とプログラムを作っていくと、ホームディレクトリにプログラムがたまり続ける。そこで、ついには、「整理しよう」という気になることであろう。そのためには

1. ディレクトリをつくる
2. ファイル(Python プログラム)を、1 で作成したディレクトリに移動する
3. Python プログラムを実行するには「作業用ディレクトリ」を、格納したディレクトリに変更してから実行する

ことが必要である(注意:1と2はコマンドを知らなくても、メニューからも実行が可能。しかし 3 はコマンドを知らないとできない)

1 のためのコマンドは `mkdir` である。今作業用ディレクトリがホームとして、そこに `Programs` というディレクトリを作ろう。それには `mkdir Programs` と入力すれば良い。ここで、`mkdir` と `Programs` との間の空白が重要である(1個以上あればよい)。`mkdir` がコマンド、`Programs` がその引数であることを空白が表しているからである。できたかどうかを確認するには `ls` というコマンドを使う。それを実行して `Programs` が表示されればよい。

2 のためのコマンドは `mv` である。`sample.py` プログラムを `Programs` ディレクトリに移すには、

```
mv sample.py Programs/
```

とする(最後の/はなくてもよい)。なおここでも `mv` と `sample.py` の間、また `sample.py` と `Programs` との間に一個以上の空白(スペース)が必要である。これら全部を打ち込むのは大変と思うかもしれない。そこで、次のことを試して欲しい: `mv s`TAB `P`TAB ここで TAB は Tab キーを押すことを表す。なお最後にエンターキーを押す。これを単語補完という。`s`TAB とすることで今の作業用ディレクトリにある `s`から始まるファイルやディレクトリが表示され、一個の場合はそれで決定されるのである。このようにすることで入力の手間が省略できる。

(7) 応用: LED.py の実行

ウェブページから LED.py をダウンロードしよう。leafpad を起動し、LED.py ファイルを開いて、その内容を確認しよう。GPIO ピンの 17 を使用することを仮定しているが、12 番ピンにプログラムを変更し、「上書き保存」しよう。また、これに合わせて LED を光らせる回路(FirstWeek.pdf の図 1.2 を参照)を組む(注意:回路と Raspberry Pi とを接続する場合は、Raspberry Pi の電源を一旦切ること!)。

このプログラムを実行するには、 sudo python LED.py (sudo, python, LED.py の間にスペース)とする。

ここで python LED.py が LED.py ファイルを python で実行する、という意味である。その前の sudo は、LED.py が GPIO を利用しており、これは「管理者権限」が必要なので、「sudo コマンド」によりそのコマンドを管理者権限で実行させるものである。(注意: Python 実行にいつも sudo が必要というわけではない。前に作った sample.py プログラムでは、 python sample.py で実行ができたことを思い出そう。GPIO を使うには sudo が必要なのである)

3. ファイルの管理(ディレクトリの構造)

どのディレクトリにどのようなファイルやディレクトリがあるかは、Windows OS と同様に、メニューバーにある「フォルダー」アイコンをクリックするなどすれば見ることができる。しかし(これも Windows OS と同様)、これでは見えないファイルやディレクトリがあったりする。そこで Terminal ウィンドゥに入力するコマンドをまず紹介する。

(1) `ls` (L の小文字と S の小文字): LiSt の省略形

このコマンドにより、現在の作業用ディレクトリにあるファイルやディレクトリの一覧が表示される。他のディレクトリにあるファイルなどを表示するには `ls /usr/bin/` のように、「`ls`、空白、ディレクトリ(のパス)」を書けば良い。また、「隠しファイル・ディレクトリを表示する `a` や ファイルの大きさなどを表示するための l オプション」がある。 例えば、 `ls -la /home` のようにすれば、`/home` ディレクトリにあるディレクトリやファイルをすべて表示するとともに、その属性(ファイルの大きさやアクセス権限など)が表示される。

ここで「`.`」と「`..`」という見慣れないものが表示されるであろう。
これは2つとも「ディレクトリ」であり、「`.`」はそのディレクトリそのものを表し、「`..`」は「親ディレクトリ」を表す。ここでディレクトリの「親子」関係については次で解説する。

(2) ディレクトリとは

さて、ディレクトリとはなんだろうか?これは Microsoft Windows の「フォルダ」と同じものである。ただ、Linux では、もっと体系的で統一されたものになっている。

まずシステム全体は `/` (スラッシュ一個)からなるディレクトリの下にある。これをルート(root)ディレクトリという。あとで見るように、すべてのディレクトリやファイルの「親元・祖先」である。その下には、`usr` や `home` などのディレクトリがある。それらは、 `/usr` とか `/home` のように表される。またこれらはその下にいろいろなディレクトリを含む。例えば `/usr` というディレクトリの下には、`local` や `bin` というディレクトリがある。これらは、`/usr/local` とか `/usr/bin` という名前で参照できる(この書き方をパス(path)という。つまり、`/`からどのように辿っていけばそのファイルやディレクトリにたどりつけるか、という道を表す)。

ちょっとややこしいが、`/` はディレクトリとディレクトリをつなく記号でもある。ここで、`/usr/local` を例にとると、これは `/usr` というディレクトリの中のディレクトリである。このとき、`/usr` を親ディレクトリ、もしくは「上」のディレクトリという。`/usr` からみると`/usr/local` は「子」もしくは「下」のディレクトリになる。このようにディレクトリに親子関係があるシステムをディレクトリ階層という。そしてこのディレクトリ階層はしばしば木構造で表される普通の木と違って、根(root)が上にある):

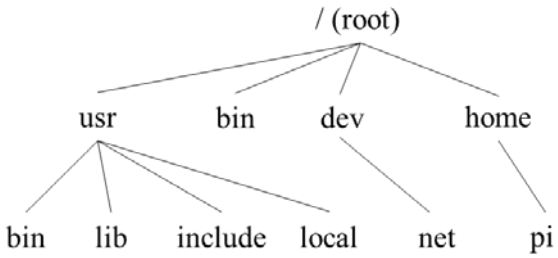

図 13. Linux のディレクトリ階層を「木構造」で表示

(3) `pwd` (Print Working Directory の省略形)

現在の作業用ディレクトリのパスを表示する。

(4) ディレクトリを渡り歩く

Linux での仕事は Microsoft Windows とはかなり異なる。まず Terminal を起動し、作業用ディレクトリを適切なものに設定し、その中のファイルを編集し、実行する、というのが標準的な作業である。つまり、「アイコンをクリック」というプログラムの起動方法はあまり使わない（プログラムを作る方としては、Linux のやり方の方が簡単なのである）。

そのためには、まず、編集や実行の対象となるファイルがあるディレクトリを、作業用ディレクトリに設定しなければならない。そのために使われるのが `cd` コマンドである。`cd` は change directory(ディレクトリを変更する)の頭文字である。`cd` の使い方には大雑把にいって、3通りの方法がある。

1. 今の作業用ディレクトリの中にあるディレクトリを、新たな作業用ディレクトリに設定する

例えば、今の作業用ディレクトリに `src` というディレクトリがあったとする。なお、`src` があるかどうかは `ls` コマンドで確認でき、それがディレクトリであるかどうかは `ls` で表示される色で区別できる---青色がディレクトリである。その src ディレクトリを新たな作業用ディレクトリにするには、

```
cd  src
```

というコマンドを実行する（最後にエンター・キーを忘れないこと）。このようなディレクトリの指定方法を相対パスという。この「相対」とは次に述べる「絶対」の反対語である。そしてこれは、`src` として指定されたディレクトリが（同じ名前のディレクトリやファイルが他所にもあるかもしれないけれども）、今の作業用ディレクトリの下にあることを意味するものである。

2. 次は、絶対パスによる指定である。LinuxOS には、`/tmp` という誰でも書き込めるディレクトリがある（ただし、シャットダウンすると中身が消えるので注意）。それを作業用ディレクトリに指定するには次のようなコマンドを実行する:

```
cd  /tmp
```

相対パスとの違いは、ディレクトリの指定に `/` から始まる名称を書いていることである。このように、ルート(/)からはじめて、どのディレクトリの中に指定したディレクトリがあるかという道筋を全部書く（いわば住所を地球、日本国からはじめて、愛知県、名古屋市、昭和区、というように全部書く方式）のが絶対パスである。ちなみにパス(path)とは道、という意味である。

相対パスは、今いるところ（作業用ディレクトリ）からどうやって道をたどれば、対象とするディレクトリにたどり着けるか、を書いたものである。それに対し、絶対パスは、必ずルートからのたどり方を書いたものである。

ここで、相対パスだと「子」のディレクトリや「孫」のディレクトリを指定するのはできるのはわかったことだろう。しかし「親」はどうやって指定したらよいだろうか？

相対パスで「親」のディレクトリを指定するには「. . 」と書く。ドットを2つ並べたものである。（注意：半角文字を常に使うこと）。今、`/home/pi` が作業用ディレクトリであるとする。そこから相対パス方式で、`/usr/bin` を作業用ディレクトリにするには、今述べたように

```
cd  ../../usr/bin
```

と書く。 図 13 をみながら考えてみよう。最初の`..`で作業用ディレクトリが `/home` となる。次の `..` でルートに移る。だからそこから `usr/bin` と指定すればよい、というわけである。

言い換えれば相対パスは作業用ディレクトリを基点とした、絶対パスはルート(/)を基点としたファ

イルやディレクトリの指定の方法というわけである。

(3) 3番目の `cd` の使い方は、次のようなものである

```
cd
```

これにより、ホームが作業用ディレクトリになる。Raspberry Pi では `/home/pi` がホームである。

(5) ディレクトリの作成と削除

今まで見たように、ディレクトリはとても重要なものである。仕事や種類によってファイルを分類して、何が入っているか、どういう作業に関係しているかが分かるような名前をつけたディレクトリを作ることは、作業効率にとても影響する(これは、コンピュータ上の作業だけではなく、日常生活でも同じである)。今までは、すでにあるディレクトリについて見てきた。ここでは、新しくディレクトリを作る、削除する、すでにあるディレクトリのコピーする、などの方法を紹介する。

コマンド名	意味
`mkdir` 名前	「名前」という新たなディレクトリを作業用ディレクトリの下に作る(make dir)
`mv` 名1 名2	「名1」というファイル／ディレクトリの名前を「名2」に変更する(move)
`cp` 名1 名2	「名1」というファイルの中身をコピー(copy)したファイルを作り「名2」とする
`cat` 名前	「名前」というファイルの中身をディスプレイに表示する
`rm` 名前	「名前」というファイルを削除する(remove)

ここまでは、ファイルとディレクトリの違いをはっきり書いてこなかった。ここでは、**ファイル**とはテキストやプログラムなどをさし、**ディレクトリ**とは下にいろいろなファイルやディレクトリを作れる「フォルダ」をさす。基本的にはファイルのコピーや削除という操作コマンドは、ディレクトリのコピーや削除のためのコマンドとしても使えるが、コマンドに対してパラメタ(余分な引数)が必要である。ここでは `-r` というパラメタを紹介する。この `r` は再帰的(recursive)を表すもので、簡単にいえば「繰り返し実行」を意味する。このパラメタはディレクトリのコピーと削除などに使える。

4. ネットワークの利用

白井研究室で提供している Raspberry Pi には Wifi アダプタ(ドングルともいう)が USB に接続されている。このため、一般の PC と同様、Raspberry Pi からブラウザを通してネット上のいろいろな情報を収集することができる。それだけではない。PC からこの Raspberry Pi に「ログイン」して作業したり、PC と Raspberry Pi のプログラムが協働していろいろなことが行えるようになっている(そのいくつかは、実験により確かめる)。

PC からこの Raspberry Pi に「ログイン」するには、以下のものが必要である。

(1) PC 上のソフト。Linux OS の PC であれば、Terminal ウィンドゥから `slogin` コマンドでできる(つまり、特別なソフトは不要)。Windows OS の PC では、 TeraTerm に代表される通信ソフトが必要である。

(2) Raspberry Pi に割り当てられた IP(ネットワーク)アドレス。家庭用の PC などでは DHCP という仕組みを使って接続されるたびにいろいろな値が割り振られる方式をとっている(そうすることで、いろいろな機器を自由にネットに接続することが可能)。この実験においてはこれでは不便なので、DHCP ではなく「固定アドレス」を採用している。自分が使っている Raspberry Pi に割り当てられている IP アドレスを調べるには次のようにする:

```
/home/pi$  ifconfig
```

5. プロセスの中断と並列処理

プログラムを実行しているとき(実行中のプログラムのことをプロセスという)に、思わぬバグによってプロセスが停止しないことがある。それを停止させるには、(Windows の場合と同様に)実行し中のプロセスのウィンドゥを「閉じる」以外に、次のような方法がある。習熟しておくと便利である。

(1) Terminal から実行しているプロセスの停止: ctrl-C (コントロール・キーを押しておいて c を押す、コントロール C と表す) ―――大抵のものはこれで停止する。停止しない場合は何回かこれを押す。

(2) Terminal から実行しているプロセスの中断: ctrl-Z (コントロール・キーを押しておいて z を押す、コントロール Z と表す) 。ctrl-C との違いは、停止ではなく「中断」(一時停止)にある。つまり、後で再開したり、並列的に走らせたり、「殺す」ことも可能である。そのためのコマンド

fg : 再開する(fg は ForeGround の略)

bg : 並列的に「裏で」走らせる。(bg は BackGround の略)

jobs : 中断されているプロセスのリストを見る。それぞれには番号が表示される

kill %n : n にはプロセスの番号が入る。最も最近中断させたプロセスを殺すには kill %% とする

(3) 複数の仕事を並列で走らせる。例えば leafpad と python を走らせる場合がこれに相当する。それにはまず

```
leafpad &
```

とする。これにより leafpad が起動され、しかも Terminal では次のコマンドを受け付ける状態になる。そこで

```
python
```

を実行すればよい。

6. Python プログラムの実行

Raspberry Pi では対話的に Python プログラムを作成するため、IDLE(Python2 と Python3 の 2 種類がある)、および IPython が用意されている。図 14 に示すように、「プログラミング」メニューから選択して起動することができる。ただし、GPIO を使うプログラムの場合は、Terminal から `sudo idle` や `sudo ipython` として起動すること。

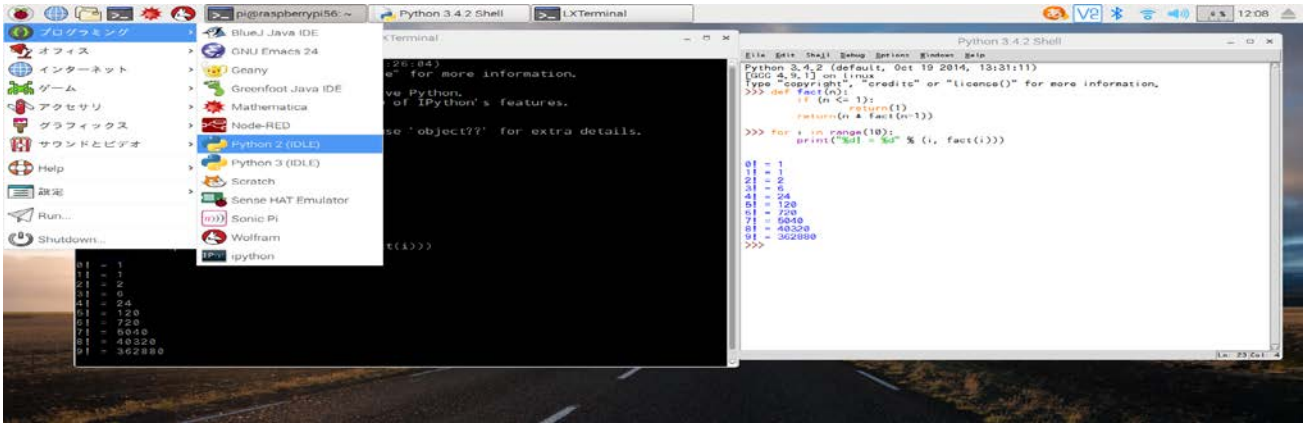

図 14. Python の対話的なプログラム作成のためのツール、IPython と IDLE