Basics in Python

Understanding Python structure

D.S. Hwang

Department of Software Science
Dankook University

Outline

Big Picture I

Basic understanding of how a program gets executed on a computer:

- A computer itself is assembled from pieces of hardware including a processor, main memory, monitor, network and so on.
- Every computer runs some kind of operating system, such as Microsoft Windows, Linux, or Mac OS X.

Big Picture II

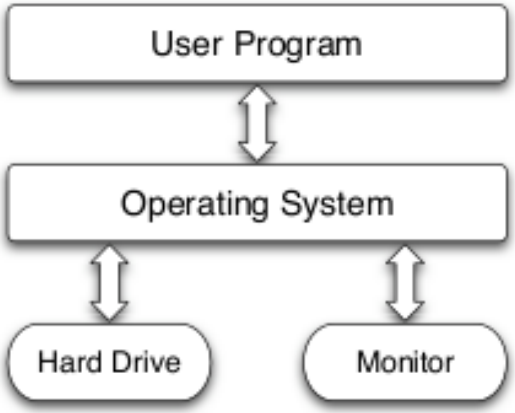


Figure: Talking to the operating system

Basic understanding of how a program gets executed on a computer:

- Only the people writing the OS have to worry about the differences among hardwar parts.

Big Picture III

- We only need to learn our way around the OS, and our programs can run on thousands of different kinds of hardware.

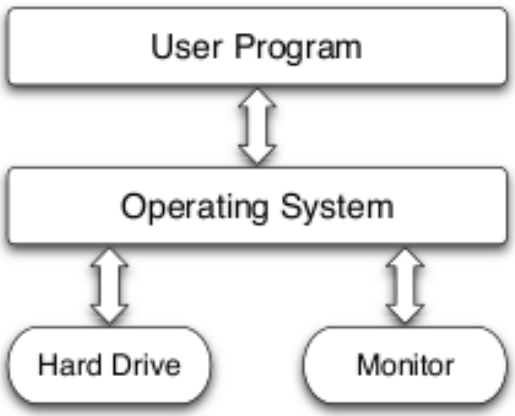


Figure: Talking to the operating system

Big Picture IV

How to communicate with OS

- Another layer between the programmer and the computer hardware.
  - use a programming language such as Python, C, Java etc.
  - use an interpreter or virtual machine
  - run a text-oriented program, called a shell
  - use an integrated development environment(IDE)

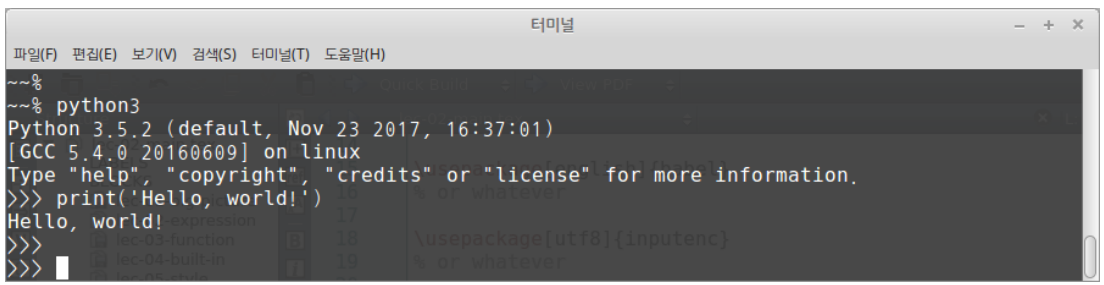


Figure: Python shell

Big Picture V

How to communicate with OS:

- Another layer between the programmer and the computer hardware.
  - use a programming language such as Python, C, Java etc.
  - use an interpreter or virtual machine
  - run a text-oriented program, called a shell
  - use an integrated development environment(IDE)

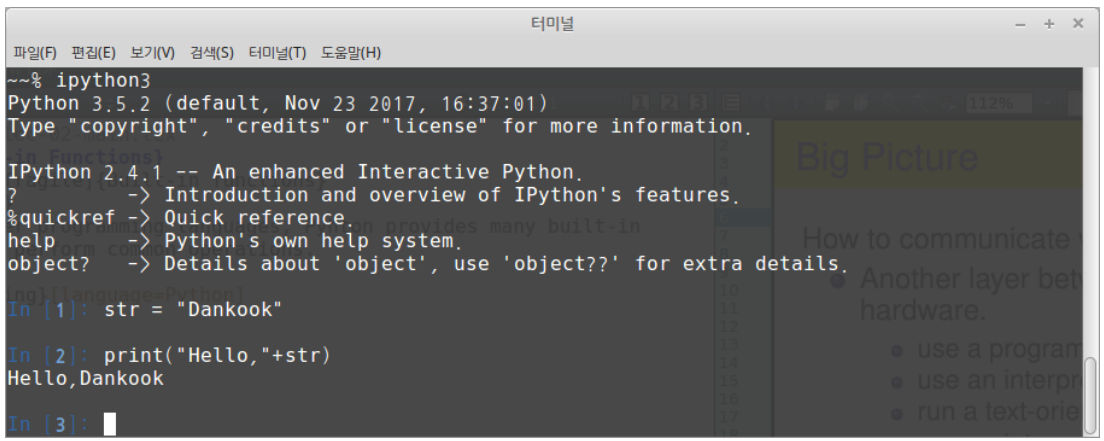


Figure: IPython shell

Big Picture VI

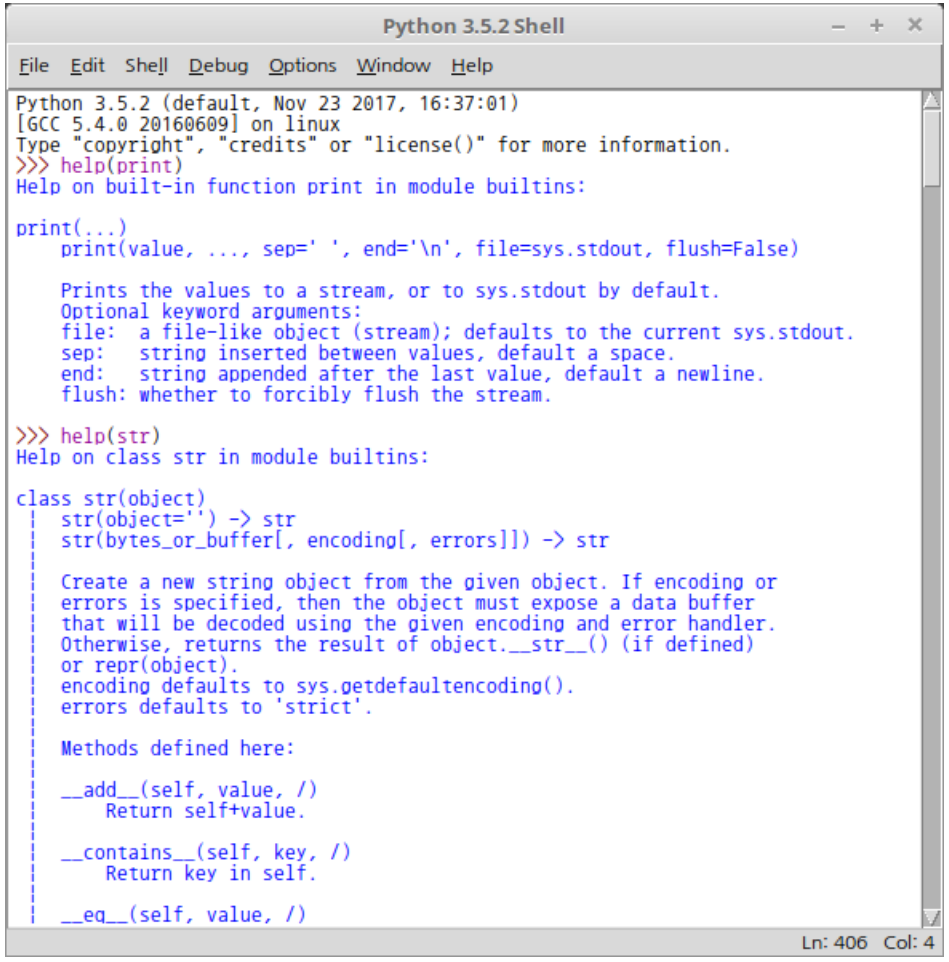
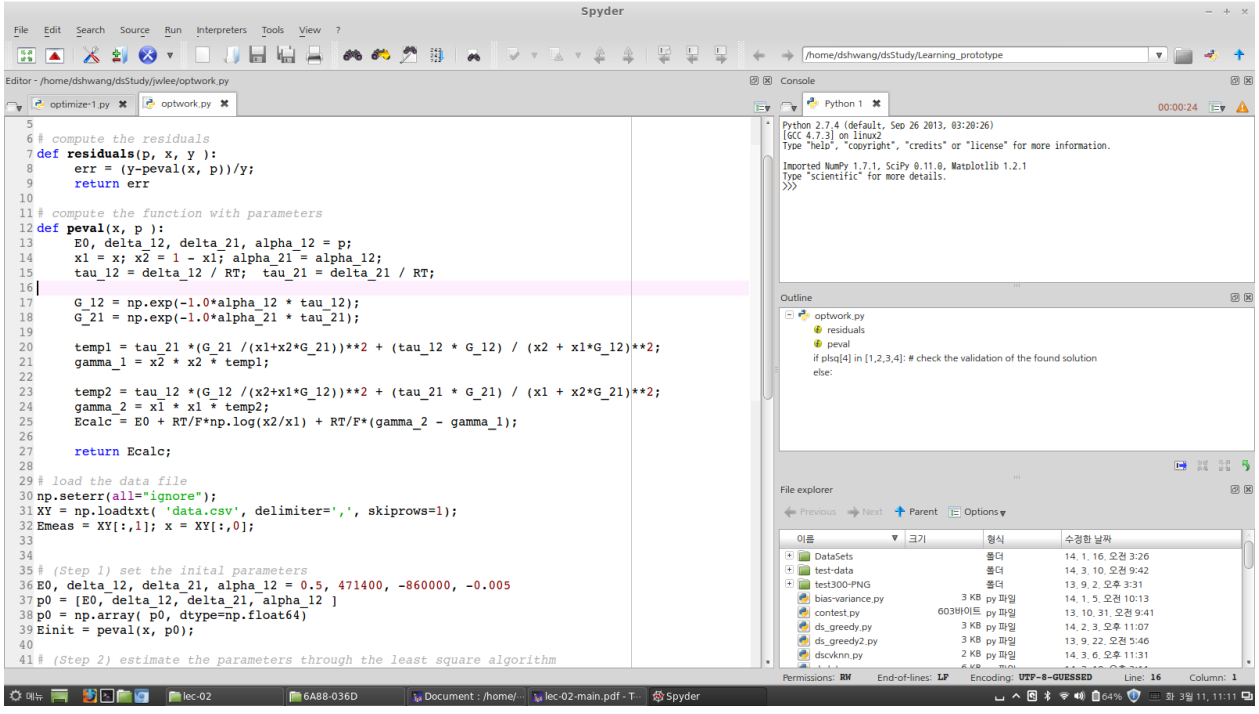


Figure: IDLE

Big Picture VII




```
5
6 # compute the residuals
7 def residuals(p, x, y ):
8       err = (y-peval(x, p))/y;
9       return err
10
11 # compute the function with parameters
12 def peval(x, p ):
13      E0, delta_12, delta_21, alpha_12 = p;
14      x1 = x; x2 = 1 - x1; alpha_21 = alpha_12;
15      tau_12 = delta_12 / RT;  tau_21 = delta_21 / RT;
16
17      G_12 = np.exp(-1.0*alpha_12 * tau_12);
18      G_21 = np.exp(-1.0*alpha_21 * tau_21);
19
20      temp1 = tau_21 *(G_21 /(x1+x2*G_21))**2 + (tau_12 * G_12) / (x2 + x1*G_12)**2;
21      gamma_1 = x2 * x2 * temp1;
22
23      temp2 = tau_12 *(G_12 /(x2+x1*G_12))**2 + (tau_21 * G_21) / (x1 + x2*G_21)**2;
24      gamma_2 = x1 * x1 * temp2;
25      Ecalc = E0 + RT/F*np.log(x2/x1) + RT/F*(gamma_2 - gamma_1);
26
27      return Ecalc;
28
29 # load the data file
30 np.seterr(all="ignore");
31 XY = np.loadtxt( 'data.csv', delimiter=',', skiprows=1);
32 Emeas = XY[:,1]; x = XY[:,0];
33
34
35 # (Step 1) set the inital parameters
36 E0, delta_12, delta_21, alpha_12 = 0.5, 471400, -860000, -0.005
37 p0 = [E0, delta_12, delta_21, alpha_12 ]
38 p0 = np.array( p0, dtype=np.float64)
39 Einit = peval(x, p0);
40
41 # (Step 2) estimate the parameters through the least square algorithm
```

Figure: Spyder

Expressions

Python commands are statements.

- expression statement, or expression
  - operator
  - operand
- variable

```
1 >>> 4 + 13
2 17
3 >>> import math
4 >>> pi = math.sqrt(math.pi)+2.2
5 >>> print pi
6 3.97245385091
7 >>>
```

Expressions

Type `int`

- Every value in Python has a particular type.
- An expression involving values of a certain type produces a value of that same type.
- Python doesn't round integer expressions

Type `float`

- represents numbers with fractional parts
- We can omit the zero after the decimal point when writing a floating-point number

```
1 >>> 17 / 10
2 1.7
3 >>> 17 // 10
4 1
5 >>> 17 % 10
6 7
```

Expressions

Finite precision

- Real computers have a finite amount of memory, which limits how much information they can store about any single number.

```
In [3]: 1.0 / 3.0
Out[3]: 0.3333333333333333
```

Operator precedence

1. exponentiation, $**$
2. negation, $-$
3. multiplication, division, remainder, and quotient$*$, $/$, $\%$, $//$
4. addition and subtraction, $+$, $-$

```
In [4]: (212-32.0)*5.0/9.0 # Fahrenheit to Celsius
Out[4]: 100.0
```

Expressions

Variable

- can be used later.
- Their names can use letters, digits, and the underscore symbol, but can not start with digits.

Assignment statement

- make a variable by giving it a value
- *degrees_celsius* = 26.0's evalatuation
  1. Evaluate the expression on the right of the = sign.
  2. Store that value with the variable on the left of the = sign.
- memory model of a variable and its associated value

Expressions

```
1 >>> number = 10
2 >>> number *= number
3 >>> number
4 100
```

Combined operator

1. Evaluate the expression to the right of the = sign.
2. Apply the operator attached to the = sign to the variable and the result of the expression.
3. Assign the result to the variable to the left of the = sign.

Expressions

```
1 >>> number = 10
2 >>> number *= number
3 >>> number
4 100
```

Combined operator

1. Evaluate the expression to the right of the = sign.
2. Apply the operator attached to the = sign to the variable and the result of the expression.
3. Assign the result to the variable to the left of the = sign.

Expressions

```
1 >>> etc
2 Traceback (most recent call last):
3   File "<pyshell#7>", line 1, in <module>
4     etc
5 NameError: name 'etc' is not defined
6 >>> 2*math.pi*
7 SyntaxError: invalid syntax
8 What things go wrong!
```

- Error messages are one of its few weaknesses from the point of view of novice programmers.
- The last line is the one that tells us what went wrong: the name something wasn't recognized.
- The second case is caused by illegal syntax.

Function

Convert t degrees Fahrenheit to Censius:

$$f(t) = \frac{5}{9}(t - 32)$$

```
>>> def to_celsius(t):
...     return (t - 32.0) * 5.0 / 9.0
...
```

Python Function vs. Math Function

Differences of (Python) function

- A function definition is another kind of Python statement; it defines a new name whose value can be rather complicated but is still just a value.
- The keyword `def` is used to tell Python that we're defining a new function.
- We use a readable name like `to_celsius` for the function rather than something like *f* whose meaning will be hard to remember an hour later.
- There is a colon instead of an equals sign.
- The actual formula for the function is defined on the next line. The line is indented four spaces and marked with the keyword `return`.

Triple–dot Prompt

- Python displays a triple-dot prompt automatically when defining a new function.
- Python automatically indents the body of the function by the required amount.

```
>>> def to_celsius(t):
...     return (t - 32.0) * 5.0 / 9.0
...
>>> to_celsius(80)
26.666666666666668
>>> to_celsius(78.8)
26.0
>>> to_celsius(10.4)
-12.0
```

Function

General form:

```
def function_name(parameters):
    block
```

- The `def` keyword tells Python that we're defining a new function.
- The name of the function comes next, followed by zero or more parameters in parentheses and a colon.
- A parameter is a variable that is given a value when the function is called.
- What the function does is specified by the block of statements inside it.

Function

```
1 def to_celsius(t):
2     return (t-32.0)*5.0/9.0
3
4 to_celsius(80)
```

- The `def` keyword tells Python that we're defining a new function `to_celsius()`.
- A parameter is `t` that is given a value when the function is called.
- What the function does is specified by the block of statements inside it.

Function I

`lambda` function

```
var = lambda arg1, arg2, ... : expr
```

- Support a simple way to create small anonymous functions, i.e. funcitons without a name
- `lambda` functions are used in combination with the functions such as `filter()`, `map()`, and `reduce()`

```
1 >>> f = lambda x, y: x*y+2
2 >>> f(10,33)
3 332
```

Function II

`lambda` function

```
map(fun, seq)
```

- Apply `func` to all the elements of `seq`

```
1 >>> celsius = [ 38.7, 36.5, 37.1, 36.2, 34.7]
2 >>> fahrenheit = map(lambda x: (9.0/5)*x+32, celsius)
3 >>> fahrenheit =list(map(lambda x: (9.0/5)*x+32, celsius))
4 >>> print(fahrenheit)
5 [101.66000000000001, 97.7, 98.78, 97.16000000000001, 94.46000000000001]
```

Function III

`lambda` function

```
filter(fun, seq)
```

- Filter out all the elements of of `seq` by using `fun` and return the element which is true

```
1 >>> fib=[0,1,1,2,3,5,8,13,21,34,55,89]
2 >>> rst=filter(lambda x: x % 2, fib)
3 >>> rst=list(rst)
4 >>> rst
5 [1, 1, 3, 5, 13, 21, 55, 89]
```

Function IV

`lambda` function

```
reduce(fun, seq)
```

- Apply `fun` to `seq` each by each

```
1 >>> import functools
2 >>> max_f = lambda a, b: a if a > b else b
3 >>> functools.reduce(max_f, range(100))
4 99
```

Local Variables

Polynomial function:

$$f(x) = ax^2 + bx + c, a \neq 0$$

```
>>> def polynomial(a, b, c, x):
...     first = a * x * x
...     second = b * x
...     third = c
...     return (first + second + third)
...
```

- Local variables exist only during function execution.
- A parameter is `t` that is given a value when the function is called.
- When the function finishes executing, the local variables no longer exist.
- Trying to access a local variable from outside the function causes an error.

Local Variables

```
1 >>> polynomial(2, 3, 4, 1.3)
2 11.280000000000001
3 >>> first
4 Traceback (most recent call last):
5 File "<stdin>" , line 1, in <module>
6 NameError: name 'first' is not defined
```

- Trying to access a local variable from outside the function causes an error.
- The scope of a variable is the area of the program that can access it.

Built-in functions

- Like other programming languages, Pyhton provides many built-in functions that perform common operations.

```
>>> abs(-9)
9
>>> round(3.3)
3.0
>>> pow(2, 4)
16
>>> int(34.6)
34
>>> float(21)
21.0
```

Good Programming

- It is important that you choose names for your variables that will help you remember what they are for.
- Keep up with the programming style guide.

Summary

- An operating system is a program that manages your computer's hardware on behalf of other programs.
- Programs are made up of statements.
- Every value in Python has a specific type.
- Expressions are evaluated in a particular order.
- Variables must be given values before they are used.
- When a function is called, the values of its arguments are assigned to its parameters, the statements inside the function are executed, and a value is returned.
- Python comes with predefined functions called `built-in`s.

Problem 1

What are the values of the following expressions, assuming that $n = 17$ and $m = 18$?

1. n // 10 + n % 10
2. n % 2 + m % 2
3. (m+n)//2
4. (m+n)/2.0
5. int(0.5*(m+n))
6. int(round(0.5*(m+n)))

Problem 2

What are the values of the following expressions, assuming that $s =''$ *Hello*$''$ and $t =''$ *World*$''$?

1. len(s) + len(t)
2. s[1]+s[2]
3. s[len(s)//2]
4. s + t
5. t + s
6. s * 2

Problem 3 I

What is a projectile?

- A projectile is an object that is given some energy to cause it to go up, reach a maximum height, and then fall down to the ground.
- The projectile can be a soccer ball, rock, bullet, baseball, or cannon ball.
- These objects all take on the same characteristic arc as they fly through the air.
- Soccer balls get kicked, rocks and baseballs get tossed, and bullets and cannon balls all fly through the air with a characteristic upside-down u-shape.

Problem 3 II

Design a program to compute the height of a ball in vertical motion.

$$h(t) = v_0 \times -\frac{1}{2} \times g \times t^2$$

- g Earth's gravity is different than the gravity on other planets. That force is 32 feet/sec every second.
- v_0 The starting upward velocity will differ for each problem.
- d The initial height of the projectile is important. If the projectile starts up high, it can get even higher after it is launched.

Problem 3 III

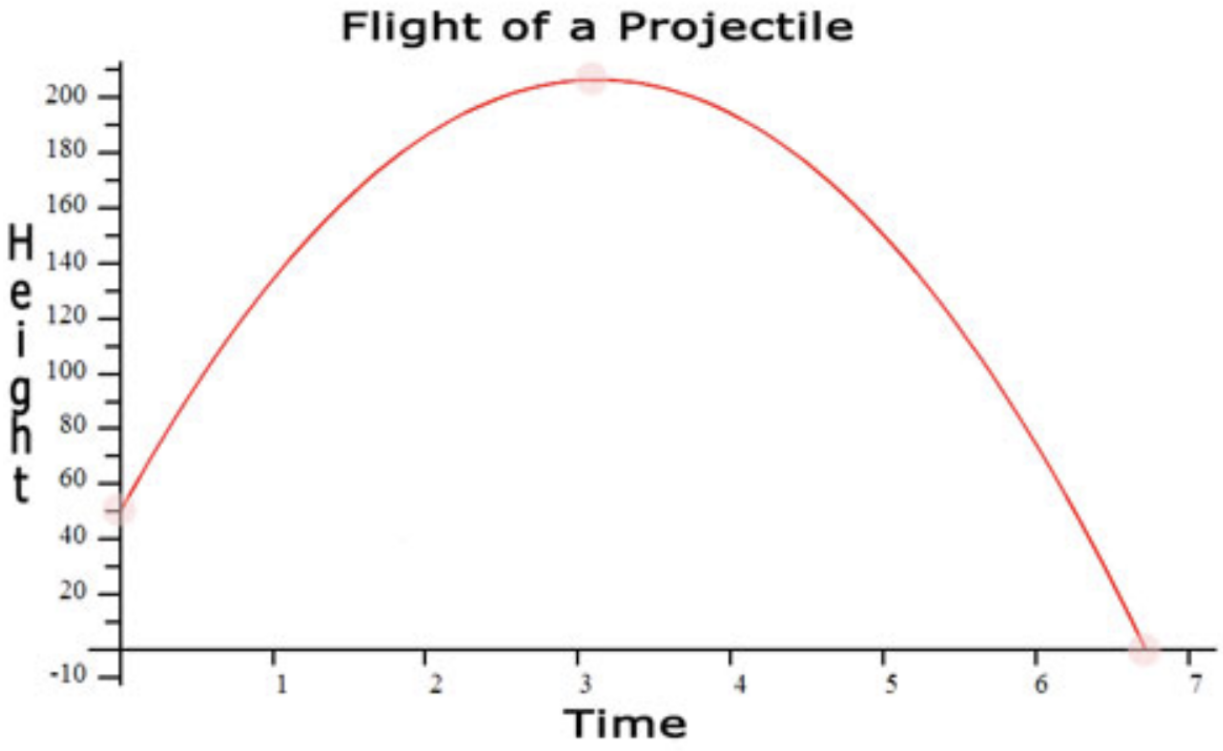


Figure: Plot of a projectile

Problem 3 IV

```
1  # Program for computing the height of a ball in vertical motion
2
3  v0 = 100     # Initial velocity
4  g = 32       # Acceleration of gravity(32 feet/sec)
5  t = 3.2      # Time
6  d = 50       # initial vertical position
7
8  # compute the vertical position
9  h = - 0.5*g*t**2 + v0*t + d
10
11 print(h, ' feet')
```