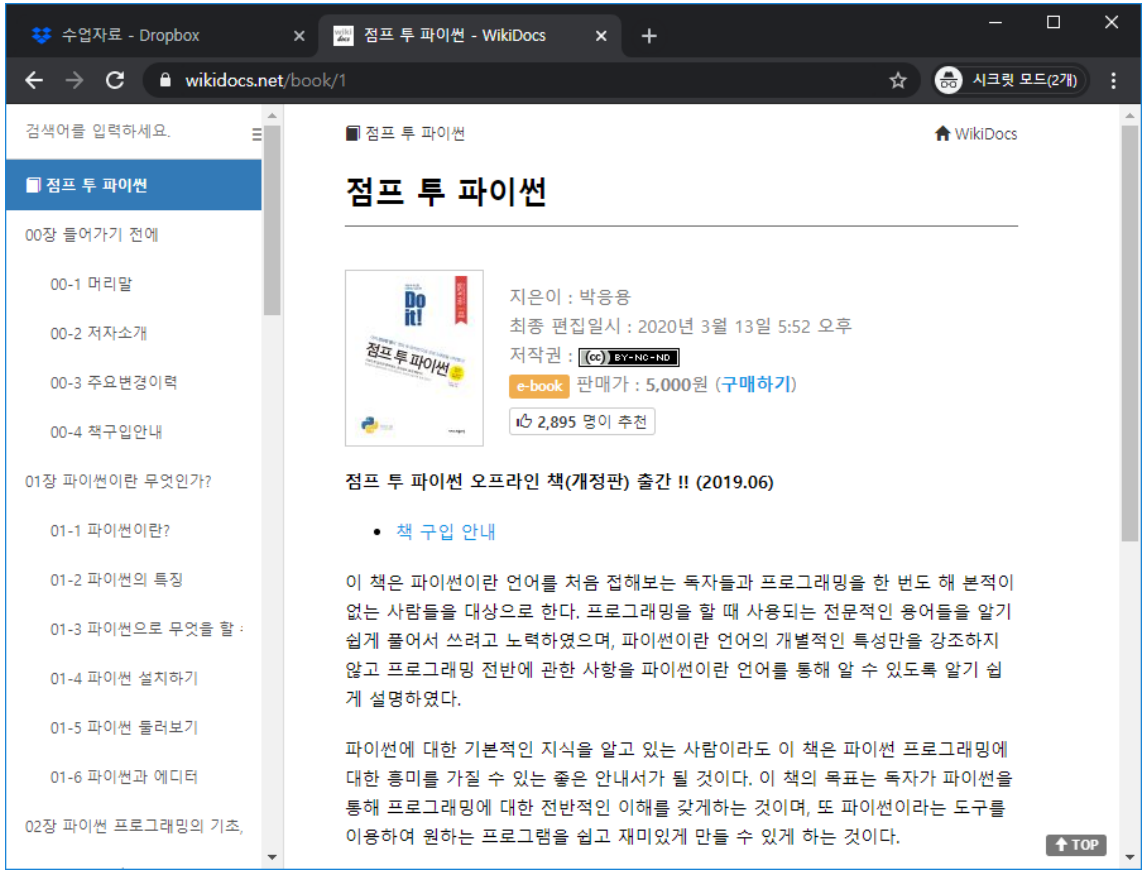
Financial Bigdata and Python

2. Basic grammar

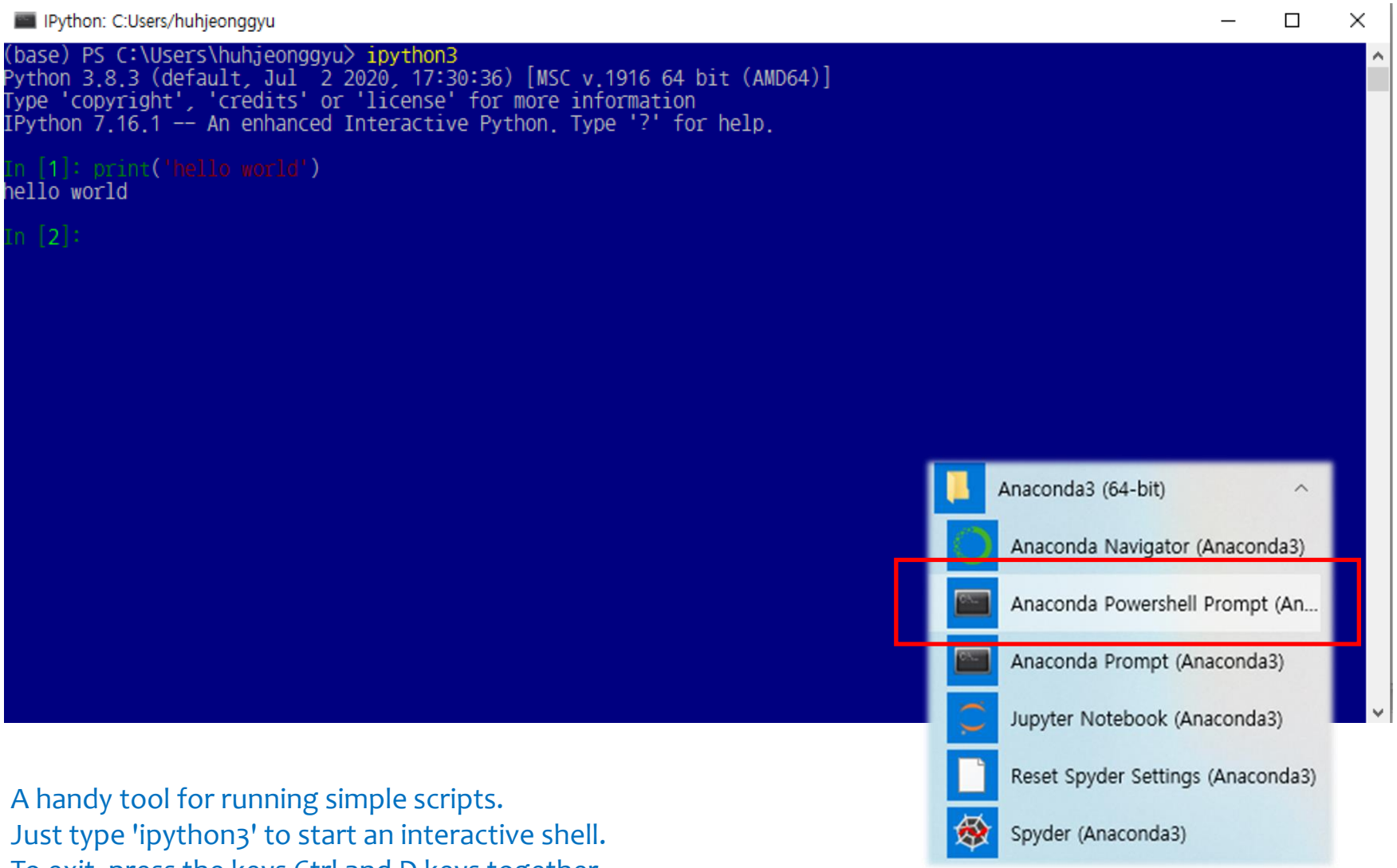
Jump to Python

https://wikidocs.net/book/1



Pahk Eung-Yong (pahkey@gmail.com)

An introduction to Python published on Wikidocs (https://wikidocs.net)



A handy tool for running simple scripts.
Just type 'ipython3' to start an interactive shell.
To exit, press the keys Ctrl and D keys together.

Anaconda Powershell

Anaconda PowerShell	Description
a=1 b=-2 c=0	creates integer variables 'a', 'b', 'c'.
d=a+b d	substitutes the sum of 'a' and 'b' into the variable 'd'. The output 'd' is an integer variable whose value is -1.
%who	A magic command that displays the current workspace. (Magic commands are separated from codes by prefixing them with %.)
%whos	It shows not only the variable name but also the variable value at once.
%cls	Screen cleanup. This does not initialize the workspace.
%who	checks that only the screen was cleaned up and the workspace was not initialized.
%reset	initializes workspace
%who	confirms that the workspace has been initialized.

Anaconda PowerShell	Description
a=5 b=2	create integer variables 'a' and 'b'.
a+b a-b a*b a/b	arithmetic operation. If it is not an operation that assigns to a new variable, the result can be checked immediately.
a%b a//b	the remainder when 'a' is divided by 'b'. the quotient when 'a' is divided by 'b'.
c = 3.14 d = -2.5	create real variables 'c' and 'd'.
c+d	Arithmetic operations can also be performed on real variables.
e = 'life is short' e	creates a character variable 'e'. checks the value of the variable 'e'.
f = True g = False	creates logical variables 'f' and 'g'
f g	check the values of the variables 'f' and 'g'

Anaconda PowerShell	Description
a=5 b='2' c=3.14	creates an integer variable 'a' creates a string variable 'b' creates a real variable 'c'
type(a) type(b) type(c)	checks variable types.
d=int(b) d	converts the string variable 'b' to the matching integer variable 'd'.
e=float(b) e	converts the string variable 'b' to the matching real variable 'e'.
f=float(a) f	converts the integer variable 'a' to the matching real variable 'f'.
g=str(a) g	converts the integer variable 'a' to the matching string variable 'g'.
h=str(c) h	converts the real variable 'c' to the matching string variable 'h'.

Exercise 1)

Hong Gil-dong's scores for each subject are as follows.

Korean: 80, English 75, Math 55

Let's find the average score of Gil-dong Hong.

Exercise 2)

Determine whether the natural number 13 is odd or even.

Anaconda PowerShell	Description
a = [1,2,3,4,5] a	creates a list with multiple variables
b = 6 c = 7 d = [b,c] d	also creates a list by collecting existing variables
e = [a,b,c,d]	even creates a list containing lists
%reset y	initializes workspace
a = [1,2,3,4,5] a[1] a[0]	creates a list returns the 1st element of the list. In Python, indexes start at 0. returns the 0th element of the list.
a[2] = 'love' a	changes the 2nd element of the list to 'love'.
a[-1] a[-2]	returns the 1st element from the end of the list. returns the 2nd element from the end of the list. (Think of the 0th element from the end of the list as a virtual element called end of list(eol).)

Anaconda PowerShell	Description
a[0:4] a[:4]	returns from the 0th element to the 3rd element (not including the 4th element). [a,b) : a is included, but b is not.
a[1:4]	returns from the 1st element to the 3rd element.
a[-4:] → a[-4:-0]	returns 4 elements from the end of the list. Violation of the rule [a,b)? No.

0	1	2	3	4	
1	2	love	4	5	eol
-5	-4	-3	-2	-1	-0

a[-4:-1]	returns from the 2nd to the 4th elements from the end of the list.
del a[2] a	deletes the 2nd element from the list.
b = [6,7,8] c = a+b c	merges two lists into a new list.
len(a)	returns the length of the list.

Anaconda PowerShell	Description
a.append('hope') a	appends the element 'hope' to the end of the list.
a.extend([10,20]) a	extends the list by adding another list to its end.
a.insert(3,30) a	inserts 30 at the 3rd index position. Elements that were originally after the 3rd value are pushed one by one.
del a[-1] a	deletes the last element.
a.remove(10) a	finds the element '10' and delete it. (if there are multiple 10s, it deletes the first element of them)
a.index(2)	finds the value ‘2’ and return its index.
b = [1,3,2,5,4] c = sorted(b) c	sorts the list 'b' and assign it to 'c'. Note that 'b' does not change.
list(reversed(b))	returns the reversed list of 'b'. Note that 'b' is unchanged.

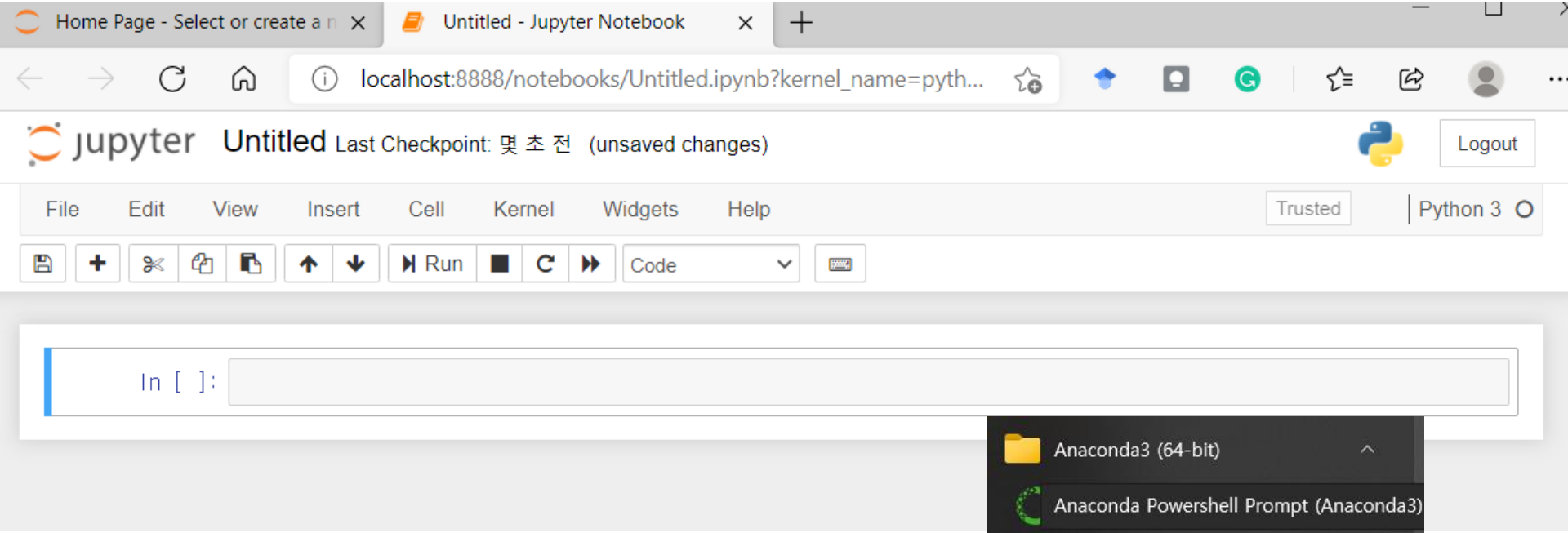
Exercise 3)

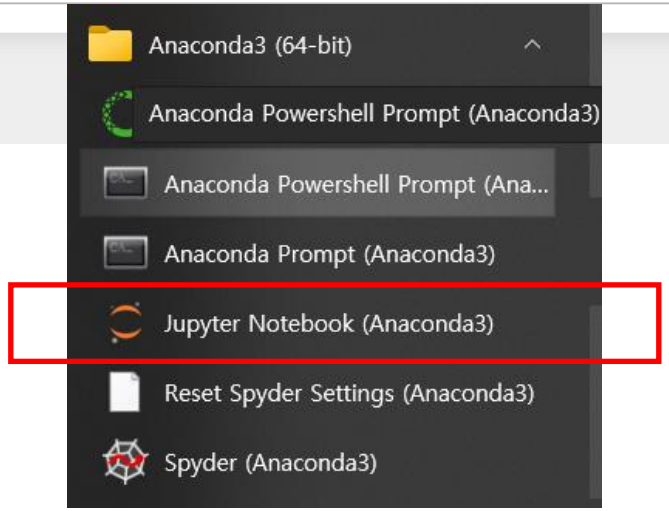
Change the list [1, 3, 5, 4, 2] to [1, 10, 20, 4, 2] by changing the elements with indices 1, 2.
(It is recommended to use only one line code instead of two! But, of course, using two lines is ok.)

Exercise 4)

Let's make the list [1, 3, 5, 4, 2] into [5, 4, 3, 2, 1].

Anaconda PowerShell	Description
a = 1 id(a)	returns the memory address of a variable
b = [1,2,3,4,5] c = b id(b) id(c)	The lists 'b' and 'c' have the same memory address. When assigning 'b' to 'c', instead of assigning the values of 'b', it substitutes the memory address of the list 'b'.
c[1] = 'love' c b	Changing 'c' causes to change 'b' also. (Generally, it is an undesired result.)
b = [1,2,3,4,5] c = b[:] id(b) id(c)	slices all values in list 'b' to copy them to 'c' deeply. In this case, the memory addresses of 'b' and 'c' are different.
c[1] = 'love' c b	So, changing 'c' does not change 'b'.





Run Jupyter
from the start menu

Jupyter Notebook	Description

pocket = ['paper', 'cellphone']
card = True
if 'money' in pocket :
  print('Go home by taxi')
elif card :
  print('Go home by taxi')
else :
  print('Go home on foot.')
	if <Boolean expression 1> :
	  <statement 1>
	  ...
	elif <Boolean expression 2> :
	  <statement 2>
	  ...
	else :
	  <statement 3>
	  ...
	
	Conditional statement:
	It executes statements by branching according to a condition.
	
	A colon (:) comes at the end of each expression.
	
	All statements executed based on whatever a Boolean expression is true must be indented with the same spacing as the expression.
	
	* a in b : True if a is in b, False otherwise

Jupyter Notebook	Description

money = 2000
if money >= 3000 :
  print('Go home by taxi')
else :
  print('Go home on foot')
	• comparison operator
	x < y : x is less than y
	x > y : x is greater than y
	x == y : x and y are equal
	x != y : x and y are not equal
	x >= y : x is greater than or equal to y
	x <= y : x is less than or equal to y

money = 2000
card = True
if money >= 3000 or card :
  print('Go home by taxi')
else:
  print('Go home on foot')
	• logical operator
	'x or y' : true if either one of x or y is true
	'x and y' : x and y must both be true to be true
	not x : true if x is false

pocket = ['paper', 'money', 'cellphone']
if 'money' in pocket :
  pass
else :
  print('Take out a card')
	pass : used to indicate to do nothing
	  even when the conditional expression is satisfied.

Exercise 5)

What would be the result of running the following code?

```
a = 'Life is too short, you need python'

if 'wife' in a:
   print('wife')
elif 'python' in a and 'you' not in a:
   print('python')
elif 'shirt' not in a:
   print('shirt')
elif 'need' in a:
   print('need')
else:
   print('none')
```

Jupyter Notebook	Description

treeHit = 0
while treeHit < 10 :
  treeHit = treeHit +1
  print('Tree was shot %d times.' % treeHit)
  if treeHit == 10 :
  print('The tree falls.')
	while <Boolean expression> :
	  <statement 1>
	  <statement 2>
	  ...
	
	while loop: executes the statements
	  while the expression is true.
	
	• Format characters and the print function
	
	print('1:%d 2:%f 3:%s 4:%e'%(3,3.14,'love',3.14))
	↓ run
	1:3 2:3.14 3:love 4:3.14e+00
	
	%d: integer, %f: real number,
	%s: string, %e: scientific notation
	
	print('This number is %5.2f'%3.14567)
	↓ run
	OO3.15

Jupyter Notebook	Description

coffee = 10
while True:
money = int(input('Put money: '))
if money == 300:
print("Here's coffee")
coffee = coffee -1
elif money > 300:
print('Here;s change %d and coffeee.' % (money -300))
coffee = coffee -1
else:
print('No money back and no coffee.')
print('The amount of coffee left is %d.' % coffee)
if coffee == 0:
print('Coffee ran out. Stop selling.')
break
	<Example> coffee machine
	
	while True :
	<statement>
	if <Boolean expression> :
	break
	
	runs in an infinite loop, executes the statement,
	and exits the loop
	when the expression is true.
	 

Exercise 6)

Let's find the sum of multiples of 3 among the natural numbers from 1 to 1000 using the while loop.

Exercise 7)

Let's write a program that displays an asterisk (*) as follows using the while loop.

```
*
**
***
****
*****
```

Jupyter Notebook	Description

marks = [90, 25, 67, 45, 80]
 	
number = 0
for mark in marks:
  number = number +1
  if mark >= 60:
  print('Student %d passed.' %number)
  else:
  print('Student %d has failed.' %number)
	for variable in list (or tuple, string):
	  <statement>
	
	for loop: repeats the statement a predetermined number of times
	
	Type 1 of For loop)
	
	A = [c,d,e]
	for a in A :
	  <statement>
	: repeats the statement for the number of elements in A
	
	1st loop: executes the statement with 'a = c'
	2nd loop: executes the statement with 'a = d'
	3rd loop: executes the statement with 'a = e'

Jupyter Notebook	Description

marks = [90, 25, 67, 45, 80]
 	
for number in range(len(marks)):
  if marks[number] < 60:
  continue
  print('Student %d passed.' % (number+1))
	Type 2 of For loop)
	
	for i in range(3) :
	  <statement>
	: repeat the statement 3 times
	
	1st loop: executes the statement with 'i = 0'
	2nd loop: executes the statement with 'i = 1'
	3rd loop: executes the statement with 'i = 2'

for i in range(2,10):
  for j in range(1, 10):
  print(i*j, end=' ')
  print('')
	• Nested loop
	'i' varies from 2 to 9,
	'j' varies from 1 to 9.
	Therefore, the statement for the for loop is executed a total of 72 (=8×9) times.
	
	• Range() function
	range(5) : creates a range from 0 to 5-1
	(i.e. 0,1,2,3,4)
	range(2,5) : creates a range from 2 to 5-1
	(i.e. 2,3,4)

Exercise 8)

Let's print the numbers from 1 to 100 using the for loop.

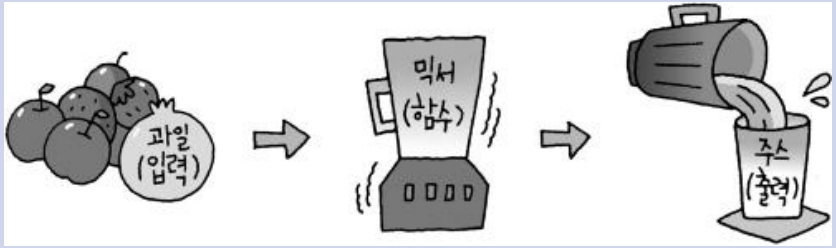
Exercise 9)

There are a total of 10 students in class A. The midterm scores of these students are as follows:

[70, 60, 55, 75, 95, 90, 80, 80, 85, 100]

Let's find the average score of class A using the for loop.

Jupyter	Description

def add(a, b):
  return a + b
 	
a = 3
b = 4
c = add(a,b)
c
	def function_name (parameter):
	  <statement 1>
	  <statement 2>
	  ...
	
	Function: a logic machine that produces a specific output according to a predefined procedure for given inputs.
	 

def value_print(a) : print(‘This number is %d’%a) a = 3 value_print(a)	A function may have no output. Notice that there is no 'return' in the left function.
def say(): print('hi') say()	A function may have no input. (It may have neither.)

Jupyter	Description
a = 1 def vartest(a): a = a +1	creates the variable "a" in the global memory
vartest(a) def vartest(a): a = a +1	calls a function "a'" is created in local memory and the value of "a" is copied to "a'" The value of "a'" changed to "a'+1" "a'" is deleted
print(a)	The variable "a" points to the variable "a" in global memory. Even if local and global memory variables have the same name, note that they are different variables with separate memory addresses.

Code without functions	Code with functions

a = [1,2,3,4,5]
for e in a :
  print(e)
print('There are %d elements'%len(a))
 	
b = ['love','hope']
for e in b :
  print(e)
print('There are %d elements'%len(b))
 	
c = [3.14,5.28,-1]
for e in c :
  print(e)
print('There are %d elements'%len(c))
	def list_print(a) :
	  for e in a :
	  print(e)
	  print('There are %d elements'%len(b))
	
	a = [1,2,3,4,5]
	list_print(a)
	
	b = ['love','hope']
	list_print(b)
	
	c = [3.14,5.28,-1]
	list_print(c)

If the overlapping part of code is written as a function,

1) code readability is high
2) it is less chance of making mistakes when writing code
3) code is easy to maintain

So, it is a good programming practice to write overlapping parts as functions.

Exercise 10)

Write a function "is_odd" that determines whether a natural number is odd or even.