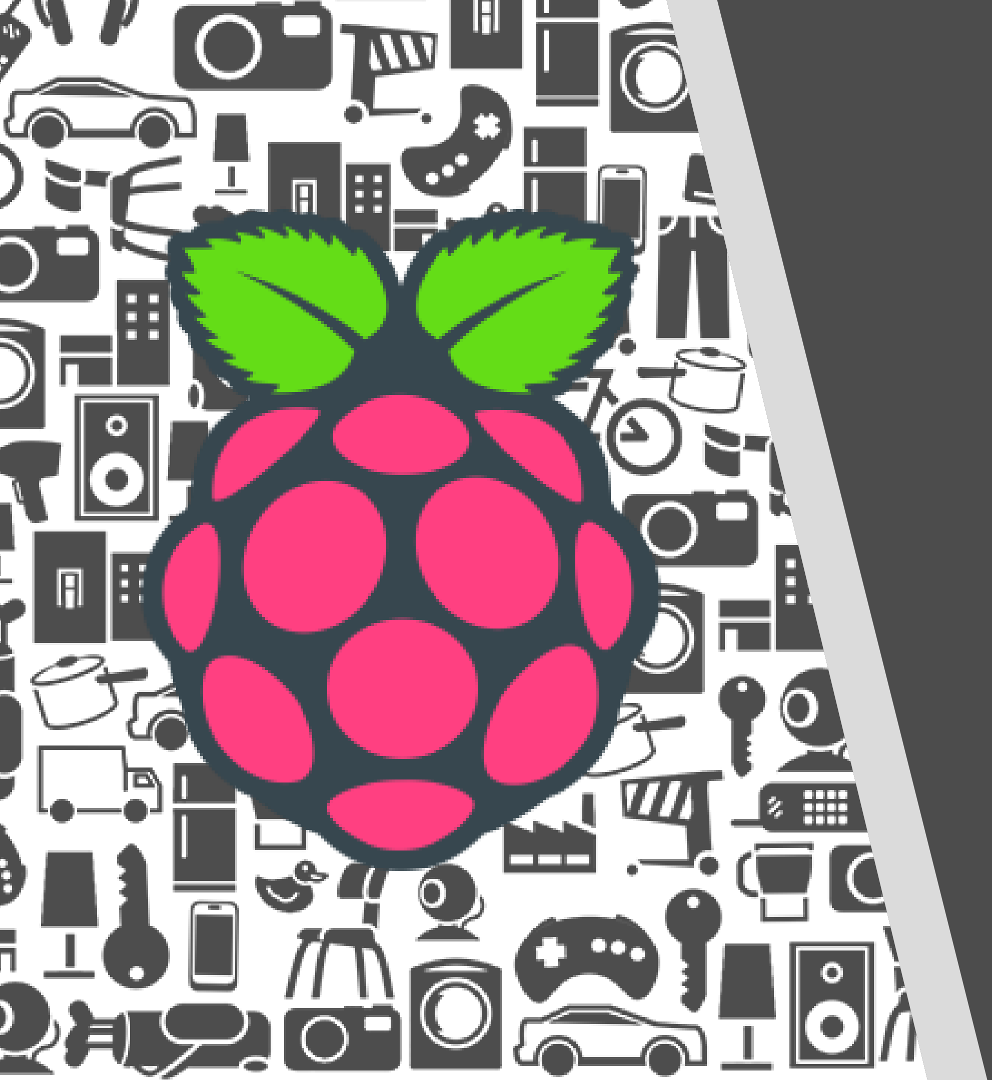


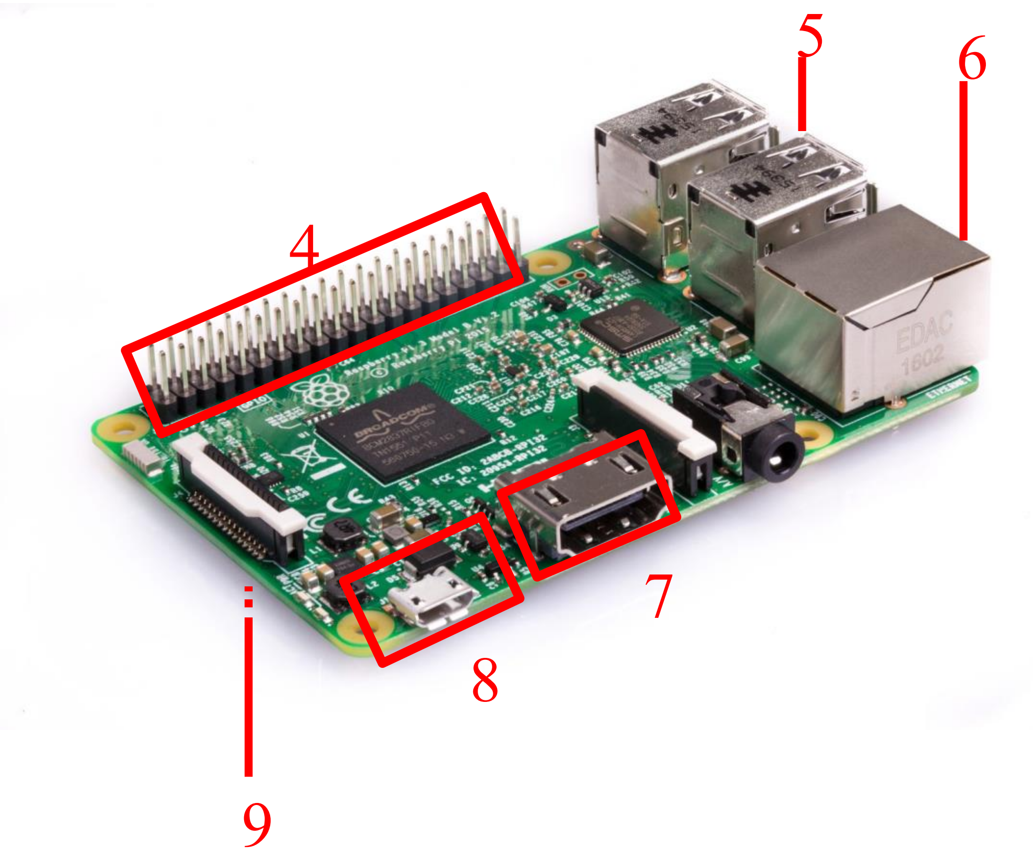
Raspberry PI

IoT & Wireless LAB 전기전자기초실험 II

Python programming

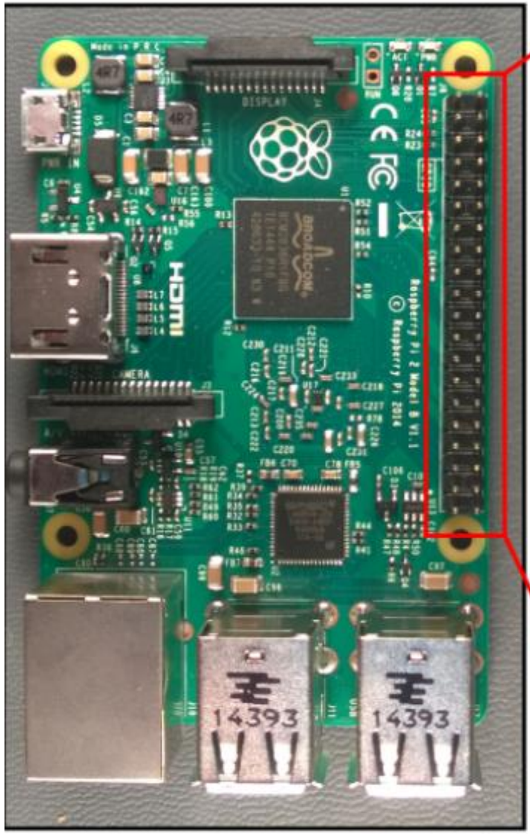
KU KONKUK UNIVERSITY

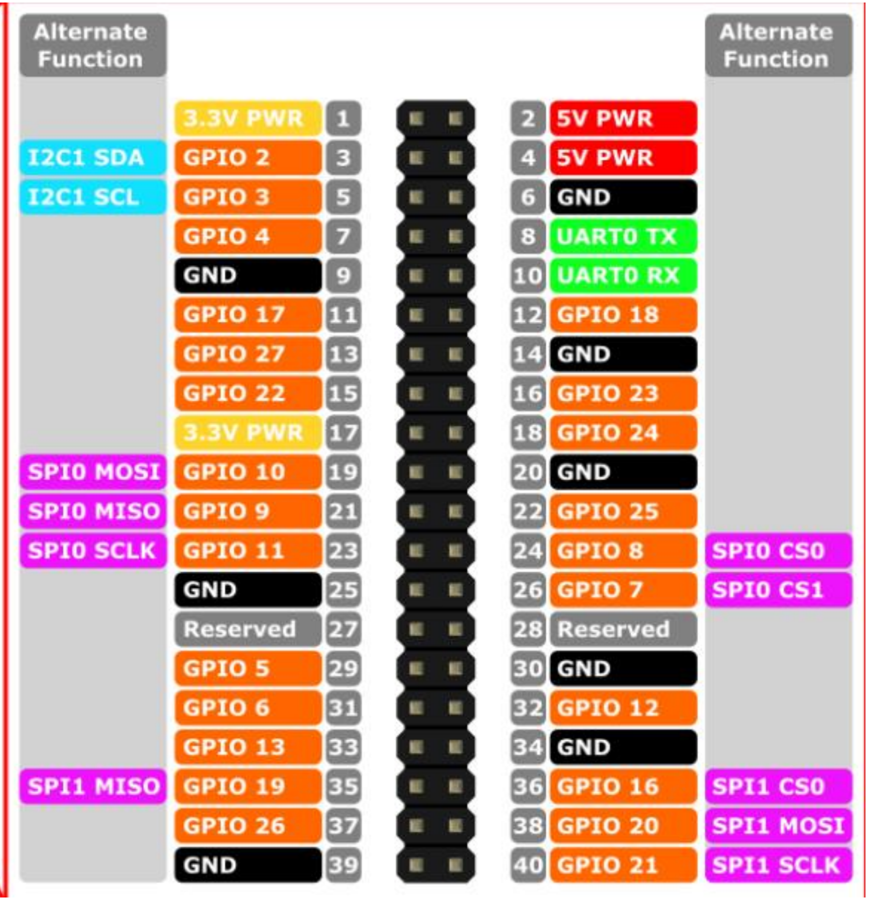
Model Specification



1 Quad Core 1.2GHz 64bit CPU
2 1GB RAM
3 BCM43438 wireless LAN and BLE
4 40-pin extended GPIO
5 USB ports
6 LAN cable port
7 HDMI port
8 2.5A USB power source
9 Micro SD port

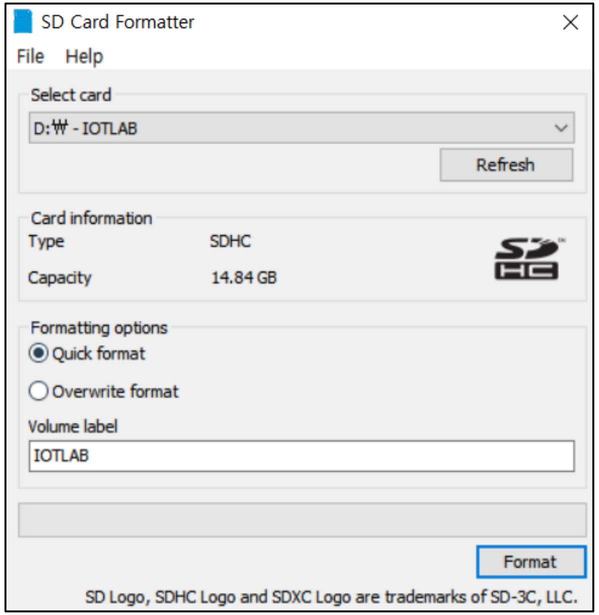
40 pin GPIO





Alternate Function					Alternate Function
	3.3V PWR	1	2	5V PWR	
I2C1 SDA	GPIO 2	3	4	5V PWR	
I2C1 SCL	GPIO 3	5	6	GND	
	GPIO 4	7	8	UART0 TX	
	GND	9	10	UART0 RX	
	GPIO 17	11	12	GPIO 18	
	GPIO 27	13	14	GND	
	GPIO 22	15	16	GPIO 23	
	3.3V PWR	17	18	GPIO 24	
SPI0 MOSI	GPIO 10	19	20	GND	
SPI0 MISO	GPIO 9	21	22	GPIO 25	
SPI0 SCLK	GPIO 11	23	24	GPIO 8	SPI0 CS0
	GND	25	26	GPIO 7	SPI0 CS1
	Reserved	27	28	Reserved	
	GPIO 5	29	30	GND	
	GPIO 6	31	32	GPIO 12	
	GPIO 13	33	34	GND	
SPI1 MISO	GPIO 19	35	36	GPIO 16	SPI1 CS0
	GPIO 26	37	38	GPIO 20	SPI1 MOSI
	GND	39	40	GPIO 21	SPI1 SCLK

OS installation



SD Card Formatter

File Help

Select card

D:₩ - IOTLAB

Refresh

Card information

Type	SDHC
Capacity	14.84 GB

Formatting options

[✓ Quick format]

Overwrite format

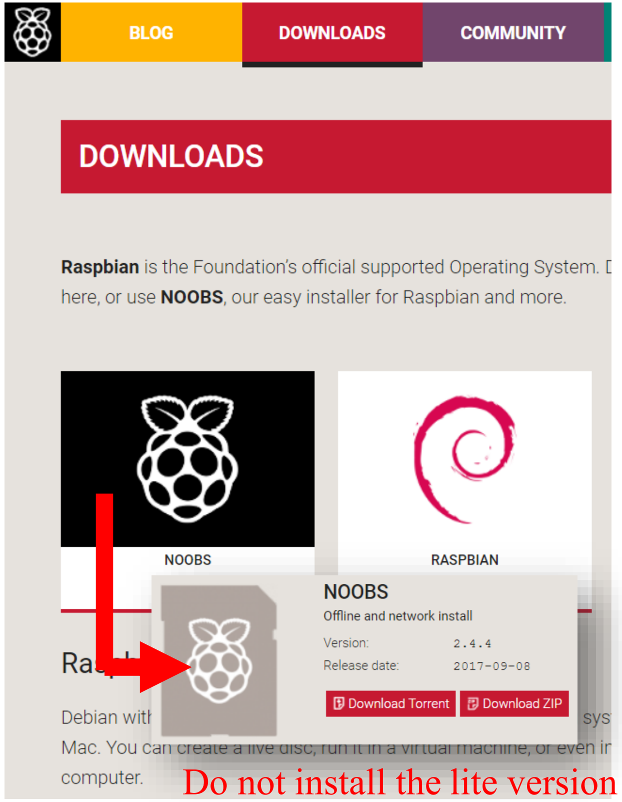
Volume label

IOTLAB

Format

SD Logo, SDHC Logo and SDXC Logo are trademarks of SD-3C, LLC.

https://www.raspberrypi.org/downloads/



DOWNLOADS

Raspbian is the Foundation's official supported Operating System. [...] here, or use **NOOBS**, our easy installer for Raspbian and more.

Ra

Debian wit

Mac. You can create a live disc, run it in a virtual machine, or even i

computer.

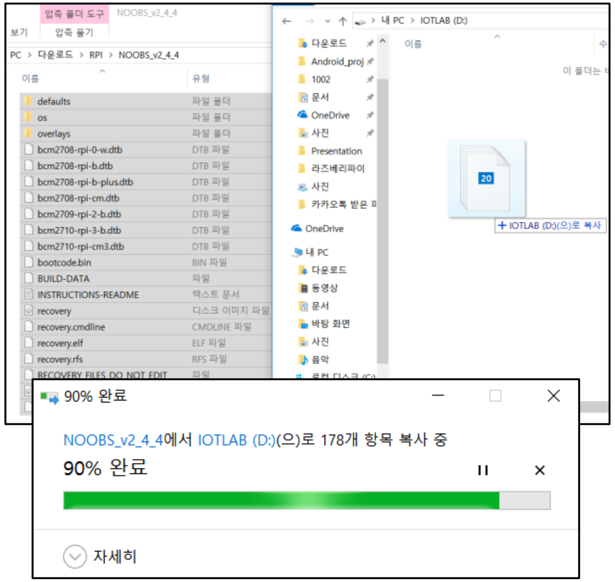
NOOBS

Offline and network install

Version: 2.4.4

Release date: 2017-09-08

Do not install the lite version



PC › 다운로드 › RPI › NOOBS_v2_4_4

내 PC › IOTLAB (D:)

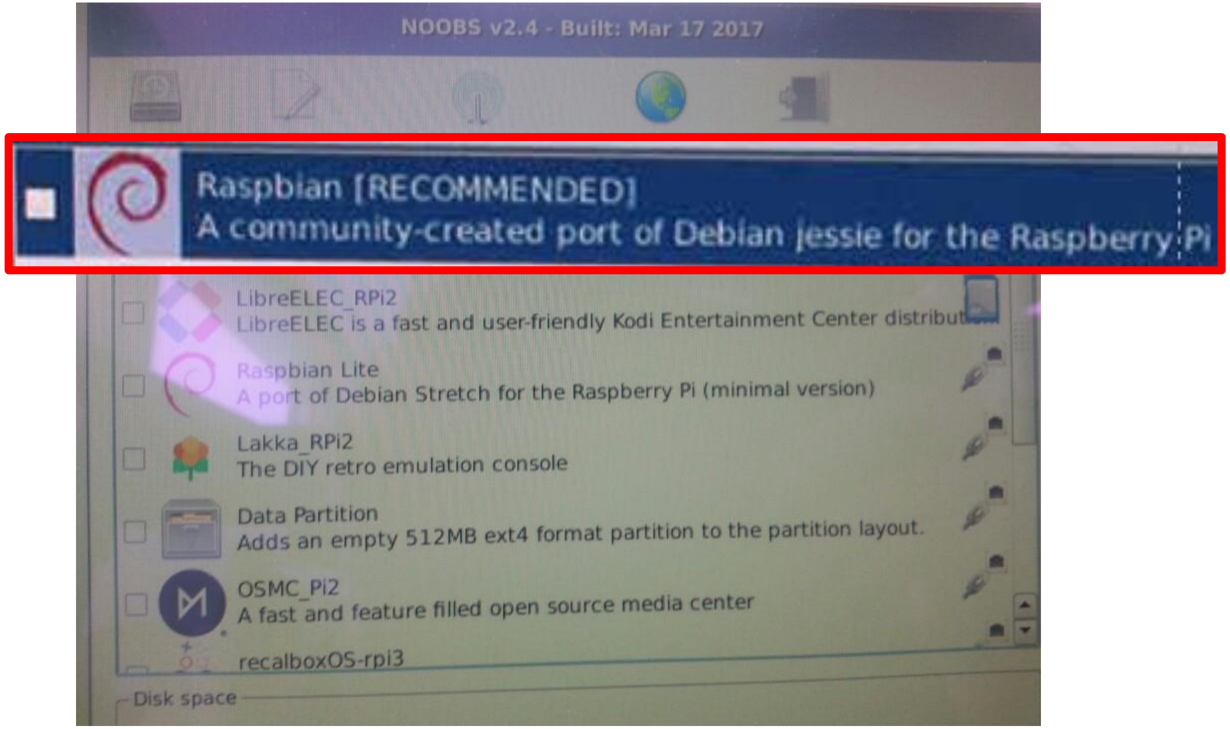
이름	유형
defaults	파일 폴더
os	파일 폴더
overlays	파일 폴더
bcm2708-rpi-0-w.dtb	DTB 파일
bcm2708-rpi-b.dtb	DTB 파일
bcm2708-rpi-b-plus.dtb	DTB 파일
bcm2708-rpi-cm.dtb	DTB 파일
bcm2709-rpi-2-b.dtb	DTB 파일
bcm2710-rpi-3-b.dtb	DTB 파일
bcm2710-rpi-cm3.dtb	DTB 파일
bootcode.bin	BIN 파일
BUILD-DATA	파일
INSTRUCTIONS-README	텍스트 문서
recovery	디스크 이미지 파일
recovery.cmdline	CMDLINE 파일
recovery.elf	ELF 파일
recovery.rfs	RFS 파일
RECOVERY_FILES_DO_NOT_EDIT	파일

90% 완료

NOOBS_v2_4_4에서 IOTLAB (D:)(으)로 178개 항목 복사 중

90% 완료

OS installation



NOOBS v2.4 - Built: Mar 17 2017

Raspbian [RECOMMENDED]
A community-created port of Debian jessie for the Raspberry Pi

LibreELEC_RPi2
LibreELEC is a fast and user-friendly Kodi Entertainment Center distribut

Raspbian Lite
A port of Debian Stretch for the Raspberry Pi (minimal version)

Lakka_RPi2
The DIY retro emulation console

Data Partition
Adds an empty 512MB ext4 format partition to the partition layout.

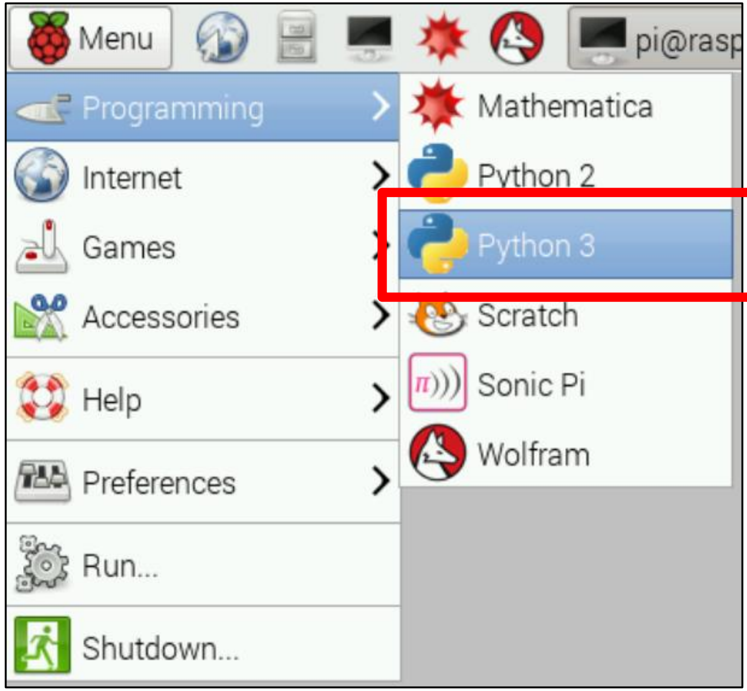
OSMC_Pi2
A fast and feature filled open source media center

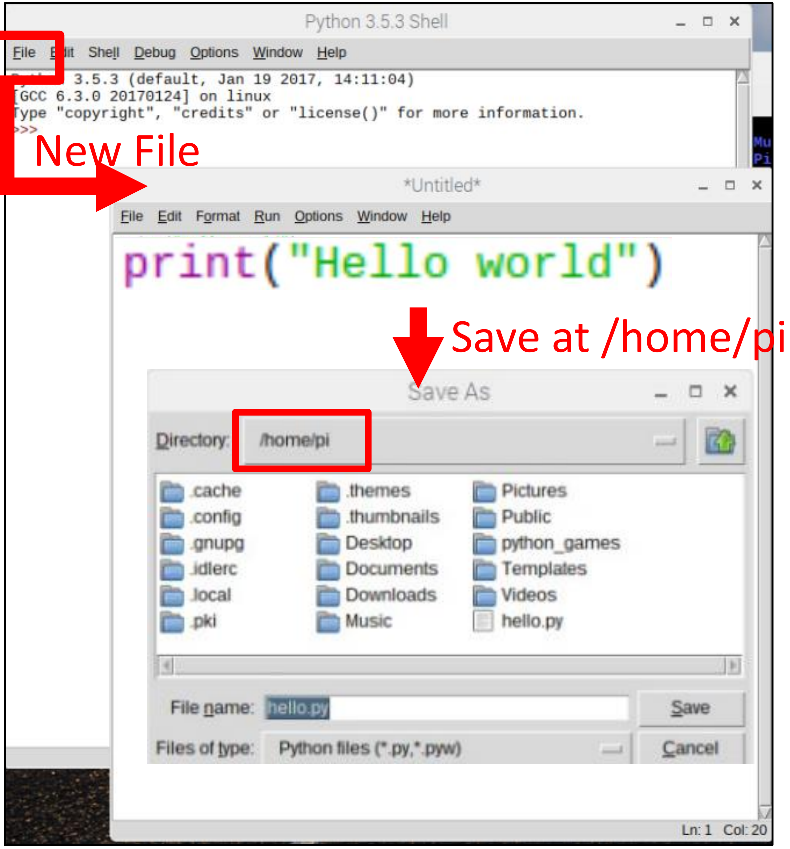
recalboxOS-rpi3

Disk space

https://www.raspberrypi.org/help/videos/#noobs-setup

Start python in raspberry





```
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

```
print("Hello world")
```

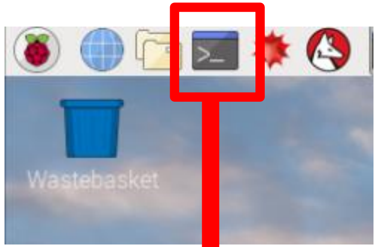
Directory: /home/pi

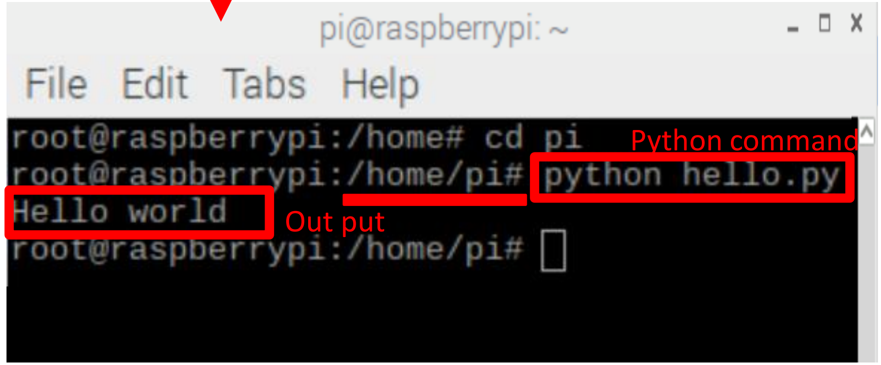
File name: hello.py

Files of type: Python files (*.py,*.pyw)

Start python in raspberry

▶ Open a terminal and start hello.py file





```
root@raspberrypi:/home# cd pi
root@raspberrypi:/home/pi# python hello.py
Hello world
root@raspberrypi:/home/pi# 
```

명령어	의미
ls	파일과 디렉토리의 목록 출력
cd	디렉토리 이동
cp	파일이나 디렉토리를 복사
mv	파일이나 디렉토리 이름을 변경하거나 다른 디렉토리로 이동
rm	파일 삭제
mkdir/rmdir	디렉토리 생성/디렉토리 삭제
cat	텍스트 파일의 내용 출력
more	텍스트 파일의 내용을 화면에 한페이지씩 출력
touch	빈 파일의 생성 혹은 파일의 생성 시간을 현재로 변경
ln	파일사이의 링크 생성
rpm	패키지 설치
gzip/gunzip	파일 압축/파일 압축 해제
chmod	소유권 변경
tar	파일 묶기, 풀기

변수명 및 예약어

변수명, 함수명, 클래스명 등의 첫 문자는 영문자 또는 underscore(_)이고,
두 번째 문자부터는 영문자, 숫자, underscore(_)일 수 있다.

파이썬에서 이미 사용하고 있는 몇몇 예약어는 변수로 사용할 수 없으며, 예약어에는 다음과 같은 것들이 있다.

and | assert | break | class | continue | def | del | elif | else | except
exec | finally | for | from | global | if | import | in | is | lambda | not
or | pass | print | raise | return | try | while | yield

내장 함수 이름이나 모듈 이름을 변수명으로 사용하게 되면, 그 이후부터 그 함수를 사용할 수 없다.

치환문 (대입문)

치환문(=)은 우변의 객체 혹은 식을 좌변의 변수명에 할당하는 것이다. 변수 이름은 선언할 필요가 없다. 변수가 필요한 시점에서 선언할 수 있다. 변수형은 우측의 객체에 의해서 결정된다.

```
>>> a = 1
>>> b = a
>>> c, d = 3, 4
>>> x = y = z = 0
>>> e = 3.5;  f = 5.6
>>> print a, b, c, d, e, f, x, y, z
1  1  3  4  3.5  5.6  0  0  0
>>> e, f = f, e
>>> print e, f
5.6  3.5
```

```
>>> a_int = 1                # 정수 선언
>>> type(a_int)              # 자료형 확인
<type ‘int’>
>>> a_str = ‘myString’       # 문자열 선언
>>> a_str
‘myString’
>>> type(a_str)              # 자료형 확인
<type ‘str’>
```

Python Numbers

Python supports four different numerical types
- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

Python Strings

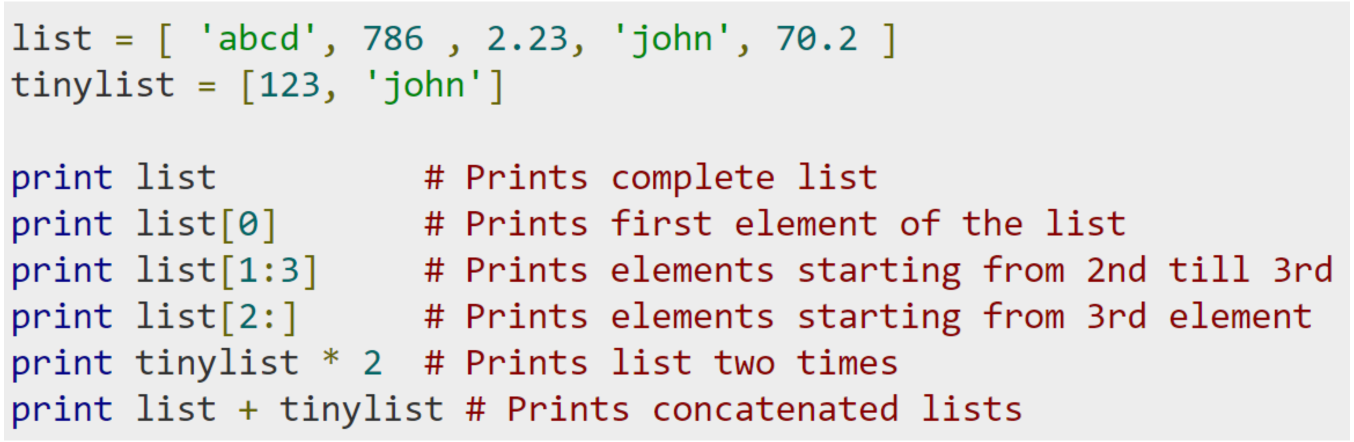
```python
str = 'Hello World!'

print str          # Prints complete string
print str[0]       # Prints first character of the string
print str[2:5]     # Prints characters starting from 3rd to 5th
print str[2:]      # Prints string starting from 3rd character
print str * 2      # Prints string two times
print str + "TEST" # Prints concatenated string
```

```
Hello World!

H

llo

llo World!

Hello World!Hello World!

Hello World!TEST
```

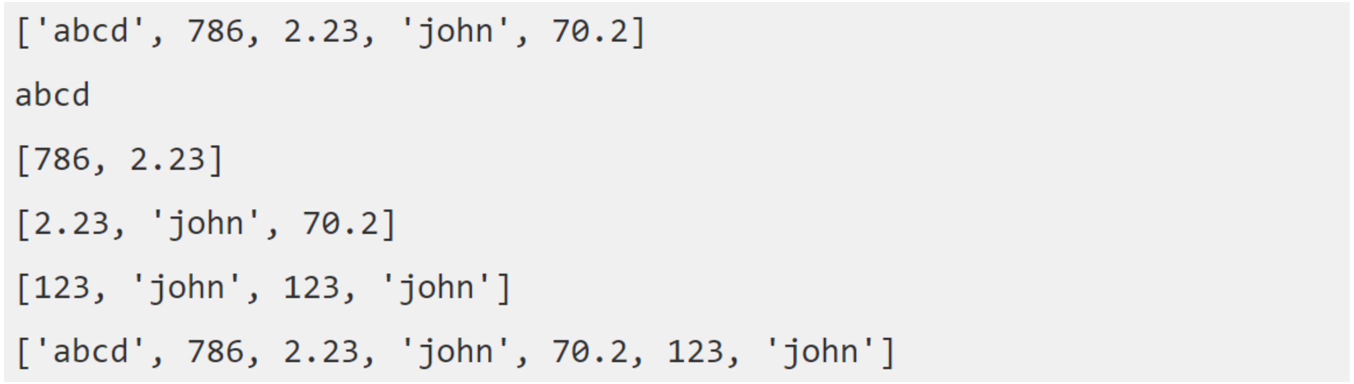
Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets [ ]. To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

Python Lists



```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print list          # Prints complete list
print list[0]       # Prints first element of the list
print list[1:3]     # Prints elements starting from 2nd till 3rd
print list[2:]      # Prints elements starting from 3rd element
print tinylist * 2  # Prints list two times
print list + tinylist # Prints concatenated lists
```



```
['abcd', 786, 2.23, 'john', 70.2]

abcd

[786, 2.23]

[2.23, 'john', 70.2]

[123, 'john', 123, 'john']

['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
The main differences between lists and tuples are: Lists are enclosed in brackets [ ] and their elements and size can be changed, while tuples are enclosed in parentheses ( ) and cannot be updated. Tuples can be thought of as read-only lists.

Python Tuples

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')

print tuple           # Prints complete list
print tuple[0]        # Prints first element of the list
print tuple[1:3]      # Prints elements starting from 2nd till 3rd
print tuple[2:]       # Prints elements starting from 3rd element
print tinytuple * 2   # Prints list two times
print tuple + tinytuple # Prints concatenated lists
```

```
('abcd', 786, 2.23, 'john', 70.2)

abcd

(786, 2.23)

(2.23, 'john', 70.2)

(123, 'john', 123, 'john')

('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
```

Python Tuples

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists.

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
list = [ 'abcd', 786 , 2.23, 'john', 70.2  ]
tuple[2] = 1000    # Invalid syntax with tuple
list[2] = 1000     # Valid syntax with list
```

Python Dictionary

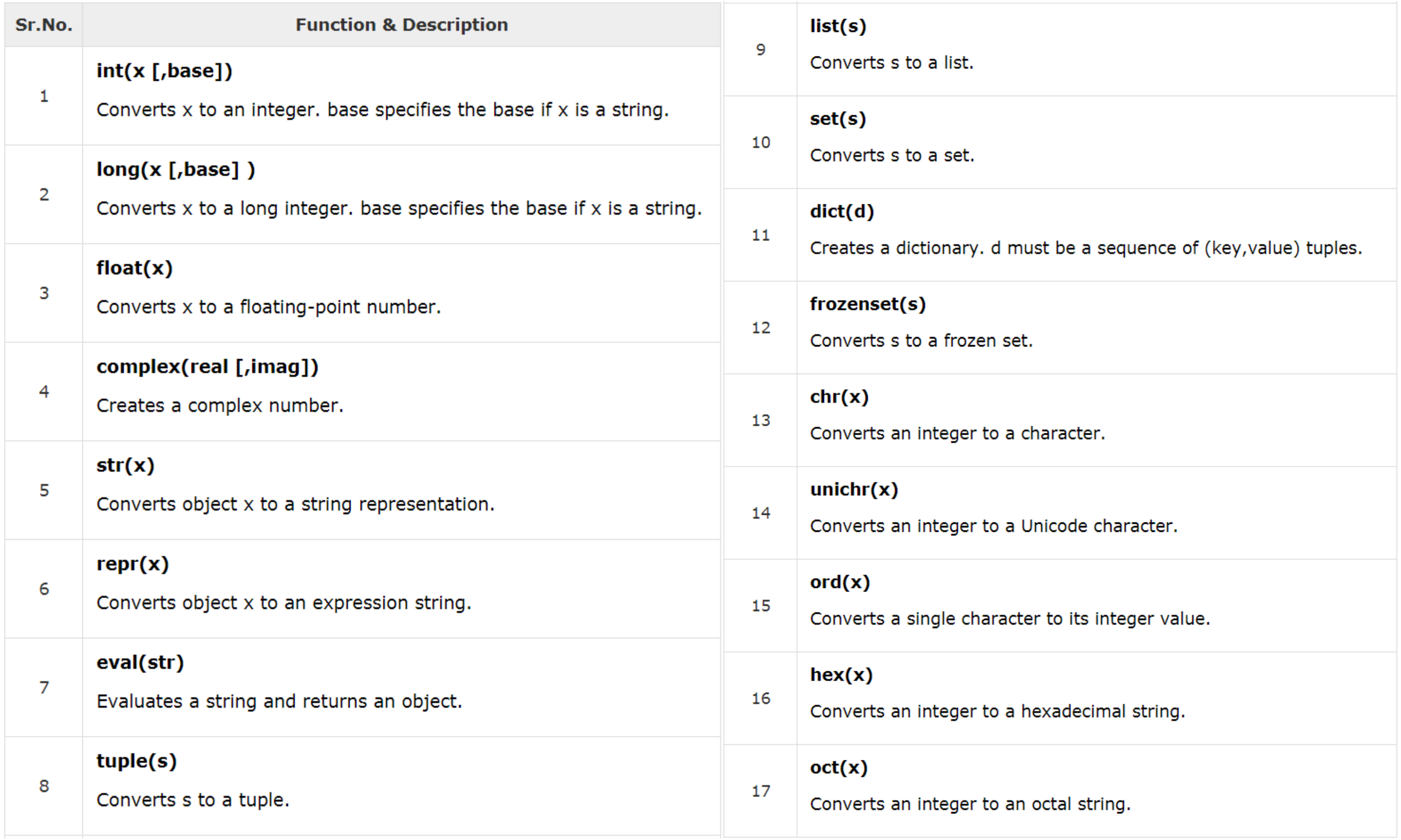
Python’s dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
Dictionaries are enclosed by curly braces { } and values can be assigned and accessed using square braces [ ].

Python Dictionary

```python
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}


print dict['one']       # Prints value for 'one' key
print dict[2]           # Prints value for 2 key
print tinydict          # Prints complete dictionary
print tinydict.keys()   # Prints all the keys
print tinydict.values() # Prints all the values
```

```
This is one

This is two

{'dept': 'sales', 'code': 6734, 'name': 'john'}

['dept', 'code', 'name']

['sales', 6734, 'john']
```

Data Type Conversion



Sr.No.	Function & Description
1	**int(x [,base])** Converts x to an integer. base specifies the base if x is a string.
2	**long(x [,base])** Converts x to a long integer. base specifies the base if x is a string.
3	**float(x)** Converts x to a floating-point number.
4	**complex(real [,imag])** Creates a complex number.
5	**str(x)** Converts object x to a string representation.
6	**repr(x)** Converts object x to an expression string.
7	**eval(str)** Evaluates a string and returns an object.
8	**tuple(s)** Converts s to a tuple.
9	**list(s)** Converts s to a list.
10	**set(s)** Converts s to a set.
11	**dict(d)** Creates a dictionary. d must be a sequence of (key,value) tuples.
12	**frozenset(s)** Converts s to a frozen set.
13	**chr(x)** Converts an integer to a character.
14	**unichr(x)** Converts an integer to a Unicode character.
15	**ord(x)** Converts a single character to its integer value.
16	**hex(x)** Converts an integer to a hexadecimal string.
17	**oct(x)** Converts an integer to an octal string.

For / If / While

- Indent spaces are regarded as inner loop

Source code input

Terminal console output

```
for i in range(3):
 TAB     print("Hello world")
```

```
Hello world
Hello world
Hello world
```

```
for i in range(2):
         print(“Hello world")
         print(“Electronics")
```

```
Hello world
Electronics
Hello world
Electronics
```

```
dog_name = "BINGO"

for char in dog_name:
         print(char)
```

```
B
I
N
G
O
```

For / If / While

▶ Iterable data type

Source code input

```
for i in 2:
        print(i)
```

Terminal console output

```
TypeError : int object is not iterable
```

▶ If statements

Source code input

```
Name = "Andy‟

if len(name) > 3:
        print("Nice name,")
        print(name)
else:
        print("That's a short name,")
        print(name)
```

▶ While statements

Source code input

```
count = 0

while (count < 5) :
        print("The count is",count)
        count = count + 1

print("End")
```

Python Arithmetic Operators

Assume variable a holds 10 and variable b holds 20, then

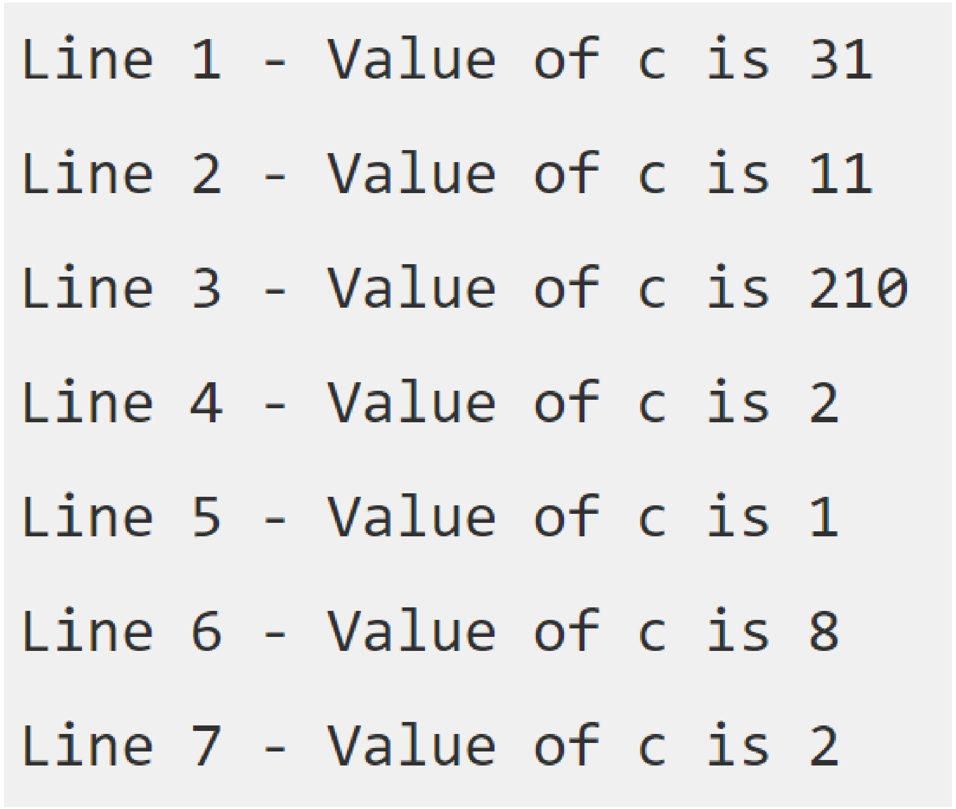
Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) –	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Python Arithmetic Operators

```
a = 21
b = 10
c = 0

c = a + b
print "Line 1 - Value of c is ", c

c = a - b
print "Line 2 - Value of c is ", c

c = a * b
print "Line 3 - Value of c is ", c

c = a / b
print "Line 4 - Value of c is ", c
```

```
c = a % b
print "Line 5 - Value of c is ", c

a = 2
b = 3
c = a**b
print "Line 6 - Value of c is ", c

a = 10
b = 5
c = a//b
print "Line 7 - Value of c is ", c
```

Python Arithmetic Operators



```
Line 1 - Value of c is 31
Line 2 - Value of c is 11
Line 3 - Value of c is 210
Line 4 - Value of c is 2
Line 5 - Value of c is 1
Line 6 - Value of c is 8
Line 7 - Value of c is 2
```

Python Comparison Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.
Assume variable a holds 10 and variable b holds 20, then

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Python Comparison Operators

```
a = 21
b = 10
c = 0

if ( a == b ):
   print "Line 1 - a is equal to b"
else:
   print "Line 1 - a is not equal to b"

if ( a != b ):
   print "Line 2 - a is not equal to b"
else:
   print "Line 2 - a is equal to b"

if ( a <> b ):
   print "Line 3 - a is not equal to b"
else:
   print "Line 3 - a is equal to b"
```

```
if ( a < b ):
   print "Line 4 - a is less than b"
else:
   print "Line 4 - a is not less than b"

if ( a > b ):
   print "Line 5 - a is greater than b"
else:
   print "Line 5 - a is not greater than b"

a = 5;
b = 20;
if ( a <= b ):
   print "Line 6 - a is either less than or equal to  b"
else:
   print "Line 6 - a is neither less than nor equal to  b"

if ( b >= a ):
   print "Line 7 - b is either greater than  or equal to b"
else:
   print "Line 7 - b is neither greater than  nor equal to b"
```

Python Comparison Operators

```
Line 1 - a is not equal to b
Line 2 - a is not equal to b
Line 3 - a is not equal to b
Line 4 - a is not less than b
Line 5 - a is greater than b
Line 6 - a is either less than or equal to b
Line 7 - b is either greater than or equal to b
```

Python Assignment Operators

Assume variable a holds 10 and variable b holds 20, then

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Python Assignment Operators

```
a = 21
b = 10
c = 0

c = a + b
print "Line 1 - Value of c is ", c

c += a
print "Line 2 - Value of c is ", c

c *= a
print "Line 3 - Value of c is ", c
```

```
c /= a
print "Line 4 - Value of c is ", c

c  = 2
c %= a
print "Line 5 - Value of c is ", c

c **= a
print "Line 6 - Value of c is ", c

c //= a
print "Line 7 - Value of c is ", c
```

Python Assignment Operators

```
Line 1 - Value of c is 31
Line 2 - Value of c is 52
Line 3 - Value of c is 1092
Line 4 - Value of c is 52
Line 5 - Value of c is 2
Line 6 - Value of c is 2097152
Line 7 - Value of c is 99864
```

Python Bitwise Operators

Assume variable a holds 60 and variable b holds 13, then

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

Python Bitwise Operators

```
a = 60            # 60 = 0011 1100
b = 13            # 13 = 0000 1101
c = 0

c = a & b;        # 12 = 0000 1100
print "Line 1 - Value of c is ", c

c = a | b;        # 61 = 0011 1101
print "Line 2 - Value of c is ", c

c = a ^ b;        # 49 = 0011 0001
print "Line 3 - Value of c is ", c
```

```
c = ~a;           # -61 = 1100 0011
print "Line 4 - Value of c is ", c

c = a << 2;       # 240 = 1111 0000
print "Line 5 - Value of c is ", c

c = a >> 2;       # 15 = 0000 1111
print "Line 6 - Value of c is ", c
```

Python Bitwise Operators

```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15
```

Python Logical Operators

Assume variable a holds 60 and variable b holds 13, then

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

Python Membership Operators

Python’s membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below.

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Python Membership Operators

```
a = 10
b = 20
list = [1, 2, 3, 4, 5 ];

if ( a in list ):
   print "Line 1 - a is available in the given list"
else:
   print "Line 1 - a is not available in the given list"

if ( b not in list ):
   print "Line 2 - b is not available in the given list"
else:
   print "Line 2 - b is available in the given list"

a = 2
if ( a in list ):
   print "Line 3 - a is available in the given list"
else:
   print "Line 3 - a is not available in the given list"
```

Python Membership Operators

```
Line 1 - a is not available in the given list
Line 2 - b is not available in the given list
Line 3 - a is available in the given list
```

Python Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators as explained below.

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

Python Identity Operators

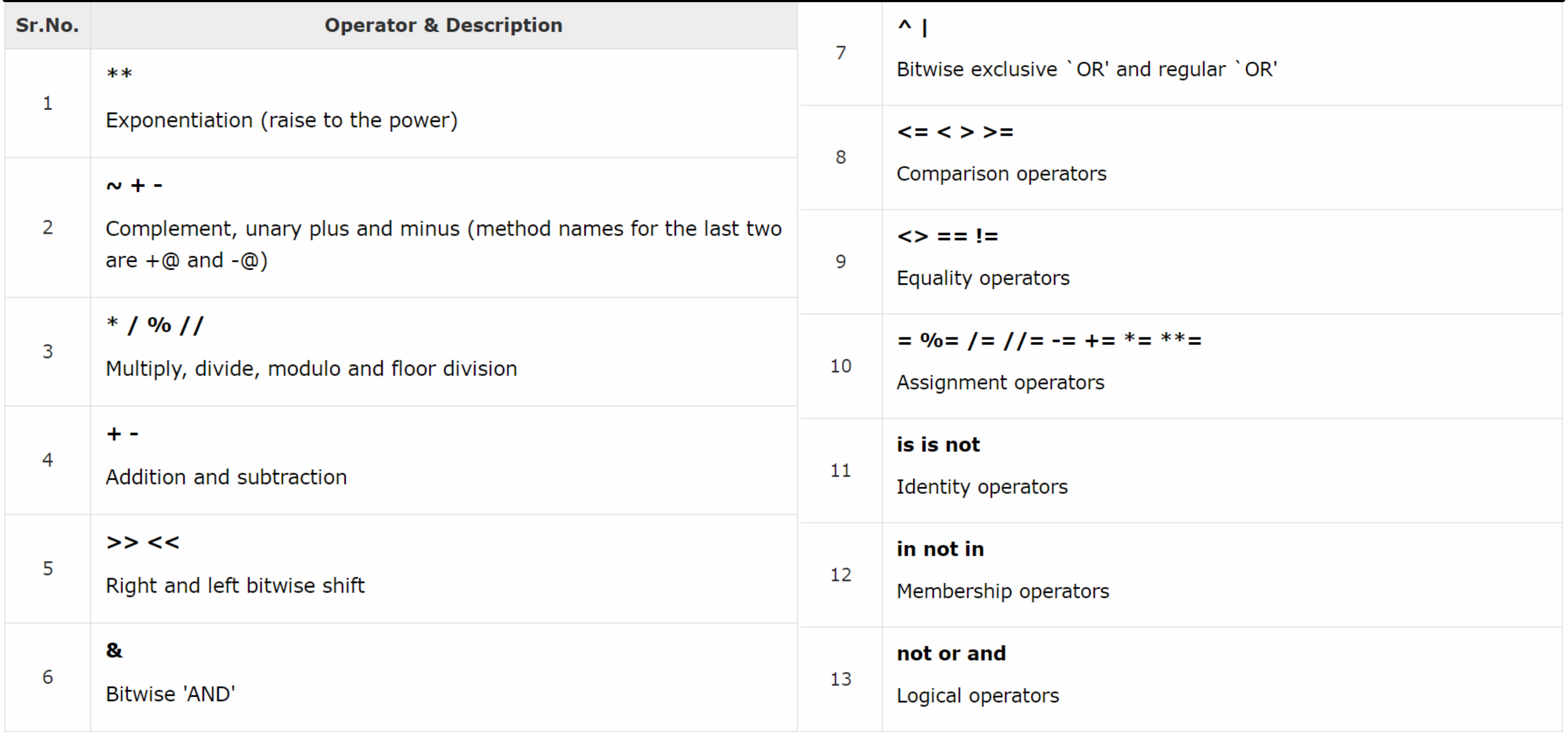
```
a = 20
b = 20

if ( a is b ):
   print "Line 1 - a and b have same identity"
else:
   print "Line 1 - a and b do not have same identity"

if ( id(a) == id(b) ):
   print "Line 2 - a and b have same identity"
else:
   print "Line 2 - a and b do not have same identity"

b = 30
if ( a is b ):
   print "Line 3 - a and b have same identity"
else:
   print "Line 3 - a and b do not have same identity"

if ( a is not b ):
   print "Line 4 - a and b do not have same identity"
else:
   print "Line 4 - a and b have same identity"
```

Python Identity Operators

```
Line 1 - a and b have same identity
Line 2 - a and b have same identity
Line 3 - a and b do not have same identity
Line 4 - a and b do not have same identity
```

Python Operators Precedence

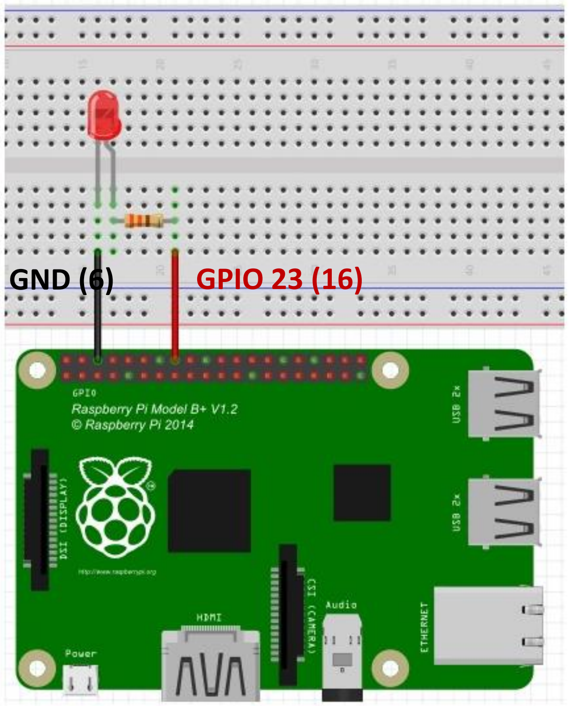
The following table lists all operators from highest precedence to lowest.

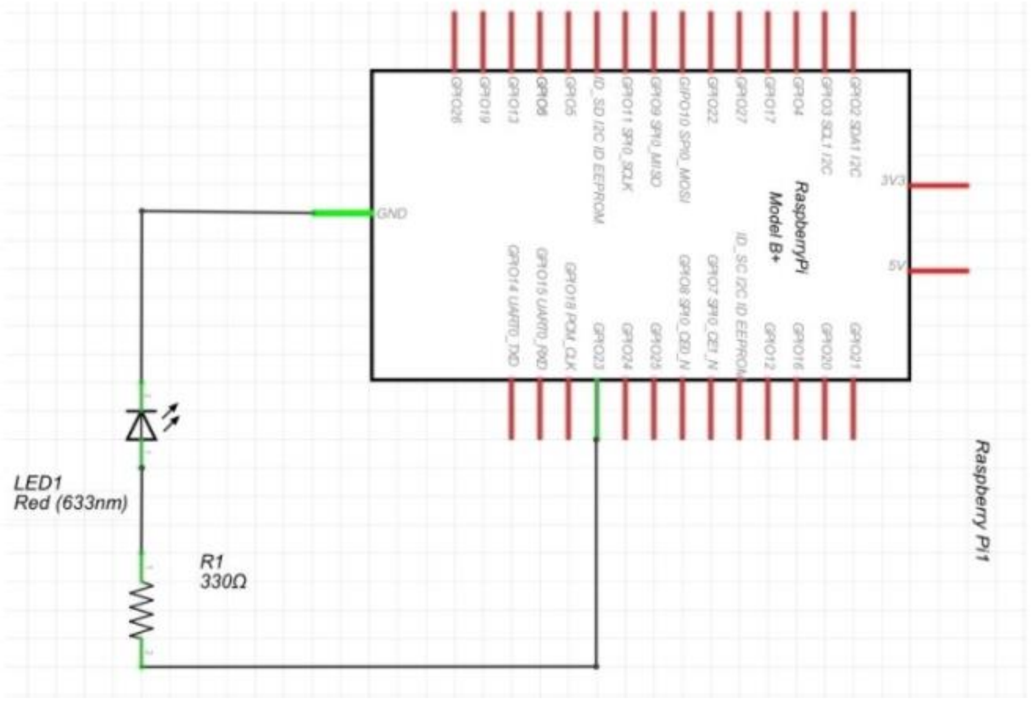


Sr.No.	Operator & Description
1	****** Exponentiation (raise to the power)
2	**~ + -** Complement, unary plus and minus (method names for the last two are +@ and -@)
3	*** / % //** Multiply, divide, modulo and floor division
4	**+ -** Addition and subtraction
5	**>> <<** Right and left bitwise shift
6	**&** Bitwise 'AND'
7	**^ |** Bitwise exclusive `OR' and regular `OR'
8	**<= < > >=** Comparison operators
9	**<> == !=** Equality operators
10	**= %= /= //= -= += *= **=** Assignment operators
11	**is is not** Identity operators
12	**in not in** Membership operators
13	**not or and** Logical operators

LED Control

- Circuit configuration





LED Control

- Py script implementation

Source code input

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
print "Setup LED pins as outputs"

GPIO.setup(23, GPIO.OUT)
GPIO.output(23, False)
GPIO.output(23, True)

time.sleep(1)

GPIO.output(23, False)

raw_input('press enter to exit program')

GPIO.cleanup()
```

Terminal console output

```
Setup LED pins as outputs
```

```
Press enter to exit program
```