accenturetechnology

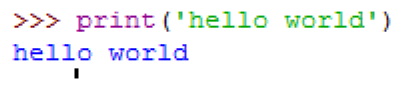
Learning and Knowledge Management

PYTHON TRAINING

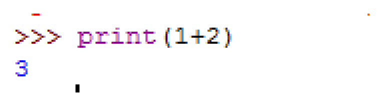
SUBTITLE LINE
SECOND LINE

Programming with Python

GETTING STARTED



```
>>> print('hello world')
hello world
```

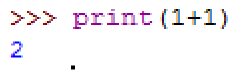


```
>>> print(1+2)
3
```

GETTING STARTED

Script Mode Programming:

Alternatively, you can write a program in a file and use the interpreter to execute the contents of the file. Such a file is called a script. For example, we used a text editor to create a file named firstprogram.py with the following contents.



```
>>> print(1+1)
2
```

There are many IDEs which can be used to write and execute python scripts:

- ❖ **Eclipse IDE**
- ❖ **Anaconda**
- ❖ **Pycharm**

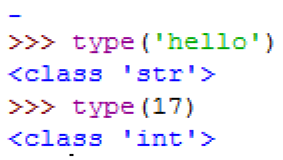
Variables, Expressions, Statements

VALUES AND DATA TYPES

A value is one of the fundamental things — like a letter or a number —that a program manipulates. Eg. 2 (the result of our previous program) and “Hello, World !”

These values belong to different data types: 2 is an *integer*, and"Hello, World!" is a *string*, so-called because it contains a string of letters. You (and the interpreter) can identify strings because they are enclosed in quotation marks.

If you are not sure what type a value has, the interpreter can tell you.



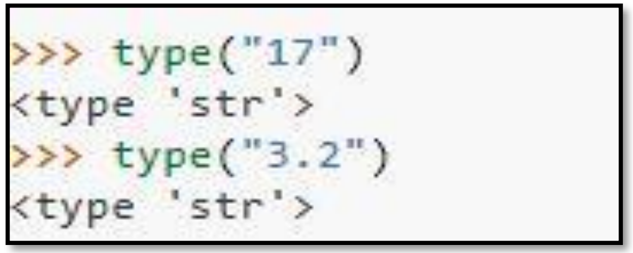
```
>>> type('hello')
<class 'str'>
>>> type(17)
<class 'int'>
```

VALUES AND DATA TYPES

Strings belong to the type str and integers belong to the type int. Less obviously, numbers with a decimal point belong to a type called float, because these numbers are represented in a format called *floating-point*.

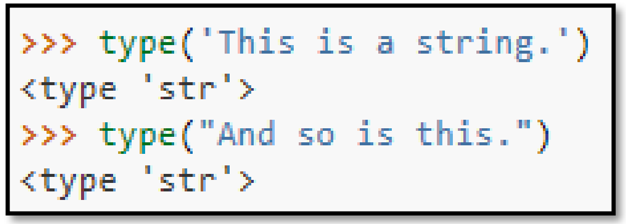
What about values like "17" and "3.2"? They look like numbers, but they are in quotation marks like strings.

- Strings in Python can be enclosed in either single quotes (‘) or double quotes (”).



```
>>> type("17")
<type 'str'>
>>> type("3.2")
<type 'str'>
```

- Double quoted strings can contain single quotes inside them, as in **"Bruce's beard"**



```
>>> type('This is a string.')
<type 'str'>
>>> type("And so is this.")
<type 'str'>
```

STANDARD DATA TYPES

- ❖ **Numbers**
- ❖ **String**
- ❖ **List**
- ❖ **Tuple**
- ❖ **Dictionary**

NUMBER DATA TYPE

- ❖Plain integers: Same precision as a C long, usually a 32bit binary number.
- ❖Long integers: Define with 100L. But, plain integers are automatically promoted when needed.
- ❖Floats: Implemented as a C double. Precision depends on your machine.
- ❖Complex numbers Define with, for example, 3j or complex(3.0, 2.0).

Here are some examples of 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

DATA TYPE CONVERSION

Function	Description
int(x [,base])	Converts x to an integer. base specifies the base if x is a string.
long(x [,base])	Converts x to a long integer. base specifies the base if x is a string.
float(x)	Converts x to a floating-point number.
complex(real [,imag])	Creates a complex number.
str(x)	Converts object x to a string representation.
repr(x)	Converts object x to an expression string.
eval(str)	Evaluates a string and returns an object.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set.
dict(d)	Creates a dictionary. d must be a sequence of (key,value) tuples.
frozenset(s)	Converts s to a frozen set.
chr(x)	Converts an integer to a character.
unichr(x)	Converts an integer to a Unicode character.
ord(x)	Converts a single character to its integer value.
hex(x)	Converts an integer to a hexadecimal string.

OPERATORS

- **Arithmetic Operators(+,-,*,/,%.**,//)**
- **Comparison (Relational) Operators(==,!=,>,<)**
- **Assignment Operators(=,+= etc..)**
- **Logical Operators**
- **Bitwise Operators(&,|,~,<<,>>)**
- **Membership Operators(in, not in)**
- **Identity Operators(is, is not)**

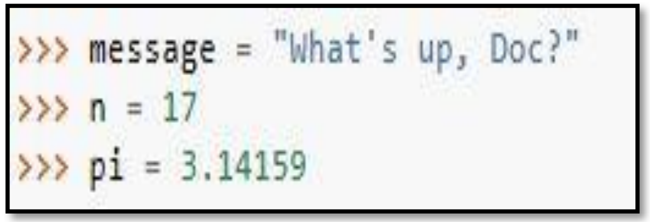
OPERATOR PRECEDENCE-HIGHEST TO LOWEST

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators

OPERATOR PRECEDENCE-HIGHEST TO LOWEST

Operator	Description
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

VARIABLE



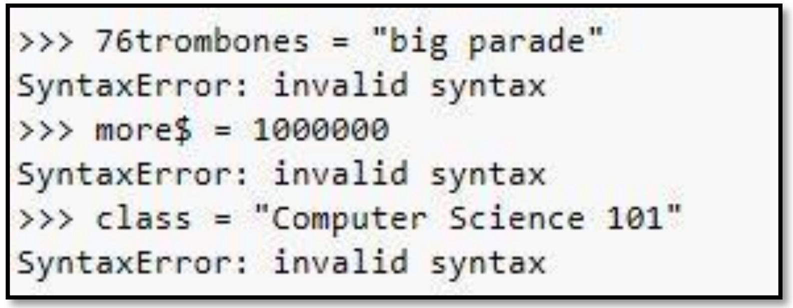
```
>>> message = "What's up, Doc?"
>>> n = 17
>>> pi = 3.14159
```

VARIABLE NAMES

Variable names can be arbitrarily long. They can contain both letters and numbers, but they have to begin with a letter. Although it is legal to use uppercase letters, by convention we don’t. Remember that case matters: Bruce and bruce are different variables.

The underscore character ‘_’ can appear in a name. It is often used in names with multiple words, such as my_name or price_of_tea_in_china.

If you give a variable an illegal name, you get a syntax error:



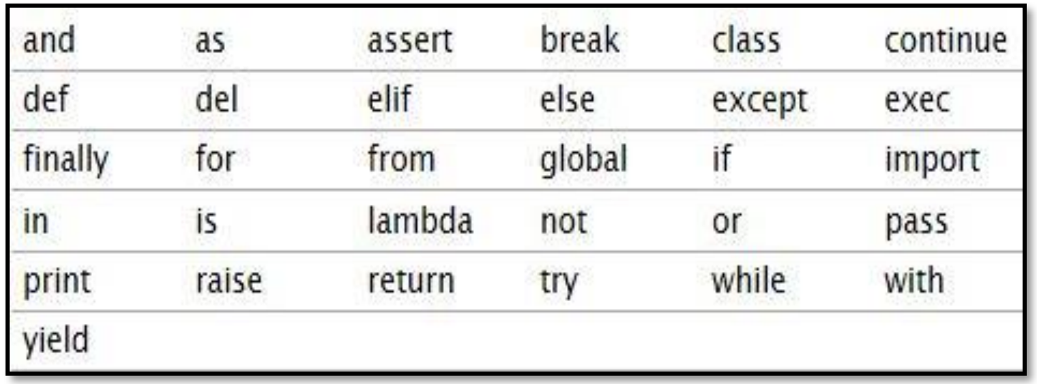
```
>>> 76trombones = "big parade"
SyntaxError: invalid syntax
>>> more$ = 1000000
SyntaxError: invalid syntax
>>> class = "Computer Science 101"
SyntaxError: invalid syntax
```

KEYWORDS

What’s wrong with Class ?

It turns out that class is one of the Python keywords. Keywords define the language’s rules and structure, and they cannot be used as variable names.

Python has thirty-one keywords:



and	as	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	with
yield					

- You might want to keep this list handy. If the interpreter complains about one of your variable names and you don’t know why, see if it is on this list.

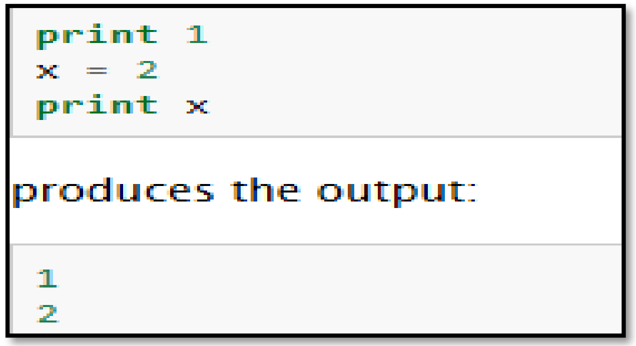
STATEMENT

A statement is an instruction that the Python interpreter can execute. We have seen two kinds of statements: print and assignment.

When you type a statement on the command line, Python executes it and displays the result, if there is one. The result of a print statement is a value. Assignment statements don't produce a result.

A script usually contains a sequence of statements. If there is more than one statement, the results appear one at a time as the statements execute.

For example, the script



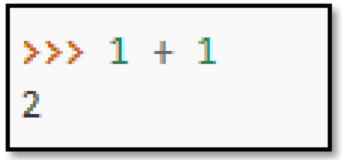
```
print 1
x = 2
print x
```

produces the output:

```
1
2
```

EXPRESSIONS

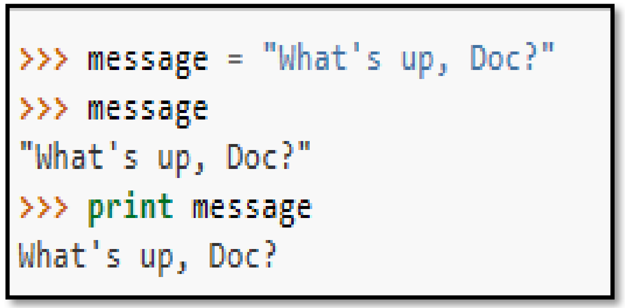
An expression is a combination of values, variables, and operators. If you type an expression on the command line, the interpreter evaluates it and displays the result:



```
>>> 1 + 1
2
```

A value all by itself is a simple expression, and so is a variable.

Confusingly, evaluating an expression is not quite the same thing as printing a value.

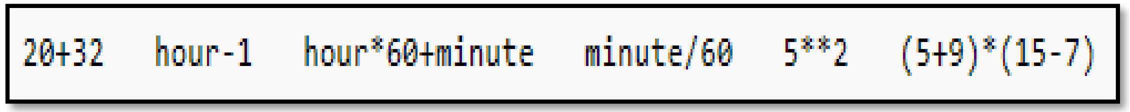


```
>>> message = "What's up, Doc?"
>>> message
"What's up, Doc?"
>>> print message
What's up, Doc?
```

OPERATORS AND OPERANDS

Operators are special symbols that represent computations like addition and multiplication. The values the operator uses are called operands.

The following are all legal Python expressions whose meaning is more or less clear:



```
20+32   hour-1   hour*60+minute   minute/60   5**2   (5+9)*(15-7)
```

The symbols +, -, and /, and the use of parenthesis for grouping, mean in Python what they mean in mathematics. The asterisk (*) is the symbol for multiplication, and ** is the symbol for exponentiation.

When a variable name appears in the place of an operand, it is replaced with its value before the operation is performed.

In case of division of an integer by an integer, the quotient is also an integer. Eg. 59/60 = 0 and NOT 0.9833

ORDER OF OPERATIONS

When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence. Python follows the same precedence rules for its mathematical operators that mathematics does. The acronym PEMDAS is a useful way to remember the order of operations:

P – Parentheses

E – Exponentiation

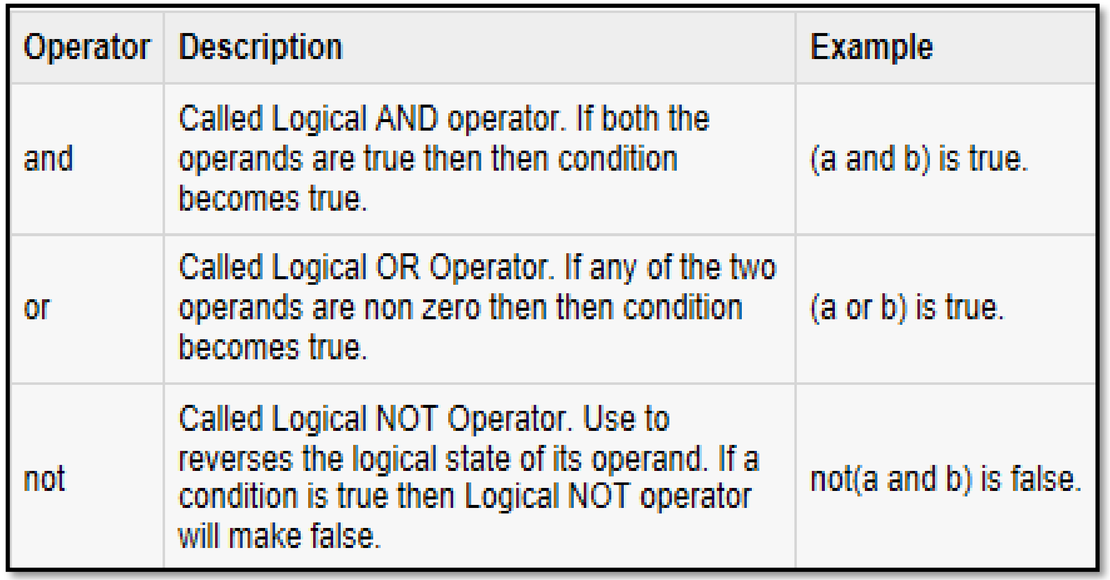
M – Multiplication

D – Division

A – Addition

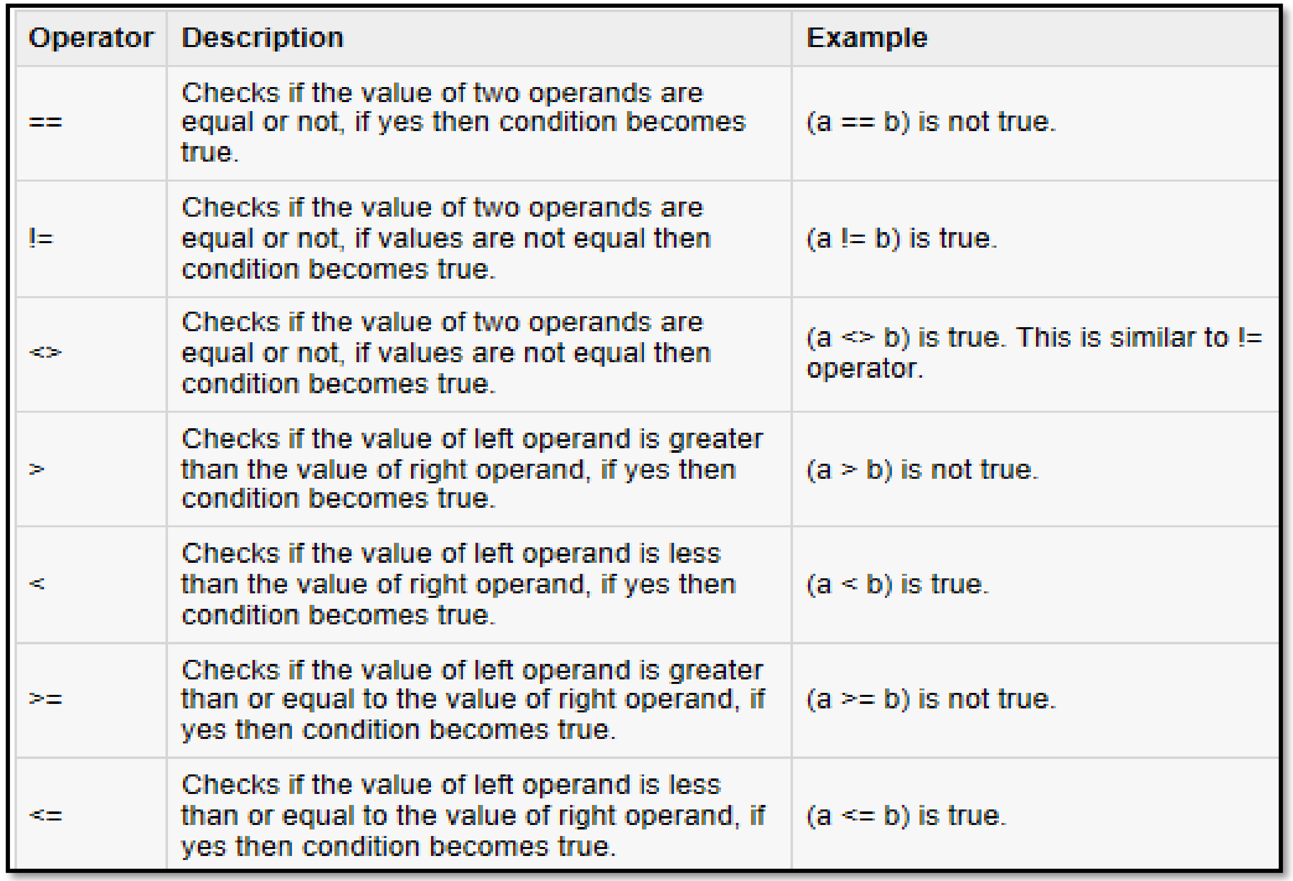
S – Subtraction

OPERATORS



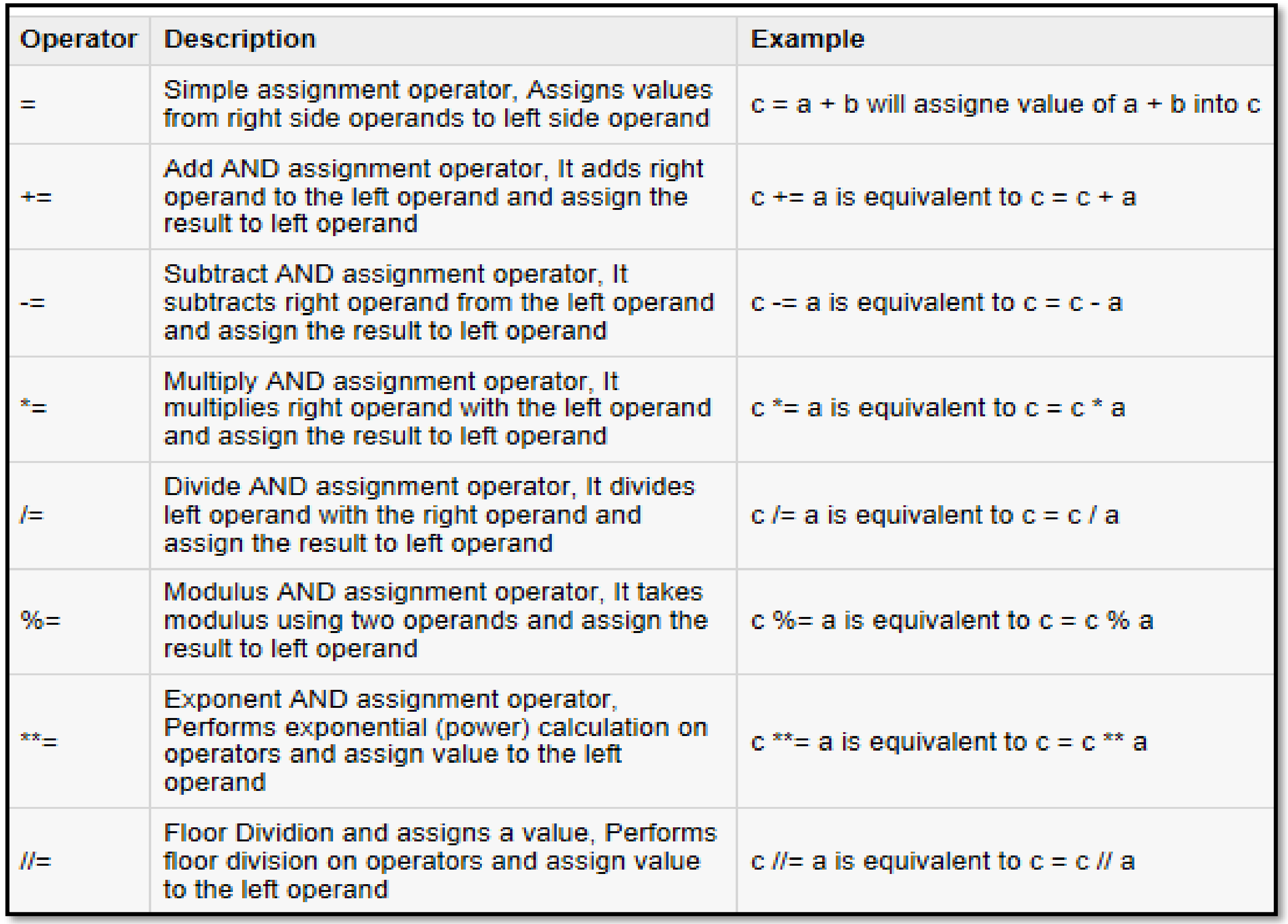
Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(a and b) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(a or b) is true.
not	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	not(a and b) is false.

OPERATORS



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

OPERATORS



Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	c = a + b will assigne value of a + b into c
+=	Add AND assignment operator, 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 assignment operator, 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 assignment operator, 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 assignment operator, It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / a
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**=	Exponent AND assignment operator, Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//=	Floor Dividion and assigns a value, Performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

BOOLEAN VALUES

BOOLEAN EXPRESSION

DECISION MAKING STATEMENTS

IF STATEMENT

IF.. ELSE STATEMENT

IF.. ELSE STATEMENT

THE *ELIF* STATEMENT

NESTED IF STATEMENT

EXAMPLE

MORE ON KEYBOARD INPUT

MORE ON KEYBOARD INPUT

MORE ON KEYBOARD INPUT

MORE ON KEYBOARD INPUT

TYPE CONVERSION

Each Python type comes with a built-in command that attempts to convert values of another type into that type. The int(ARGUMENT) command, for example, takes any value and converts it to an integer, if possible, or complains otherwise:

```
>>> int("32") 32

>>> int("Hello")

ValueError: invalid literal for int() with base 10: 'Hello'
```

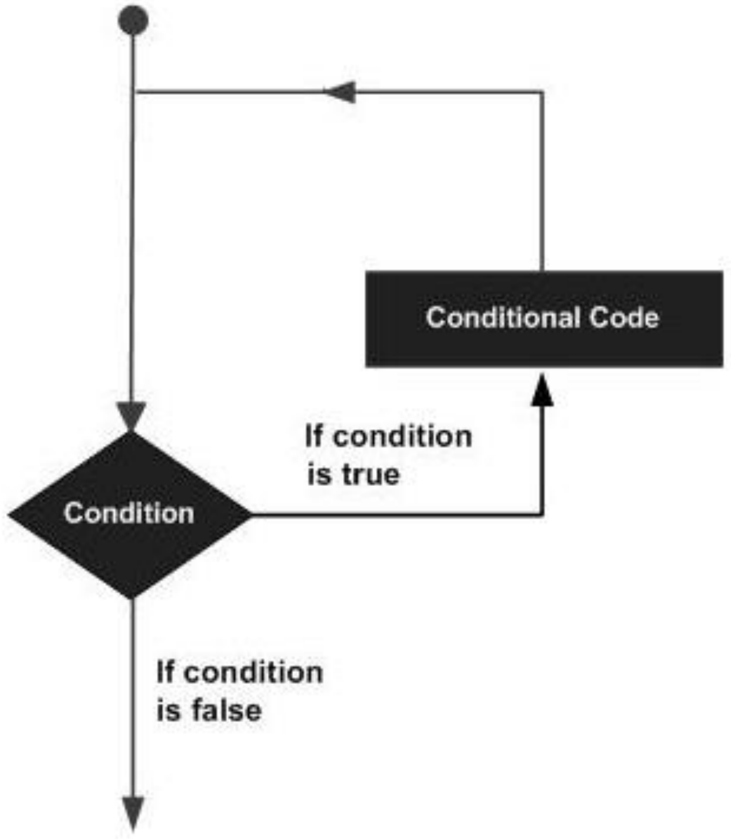
int can also convert floating-point values to integers, but remember that it truncates the fractional part:

```
>>> int(-2.3)

-2

>>> int(3.99999)

3
```

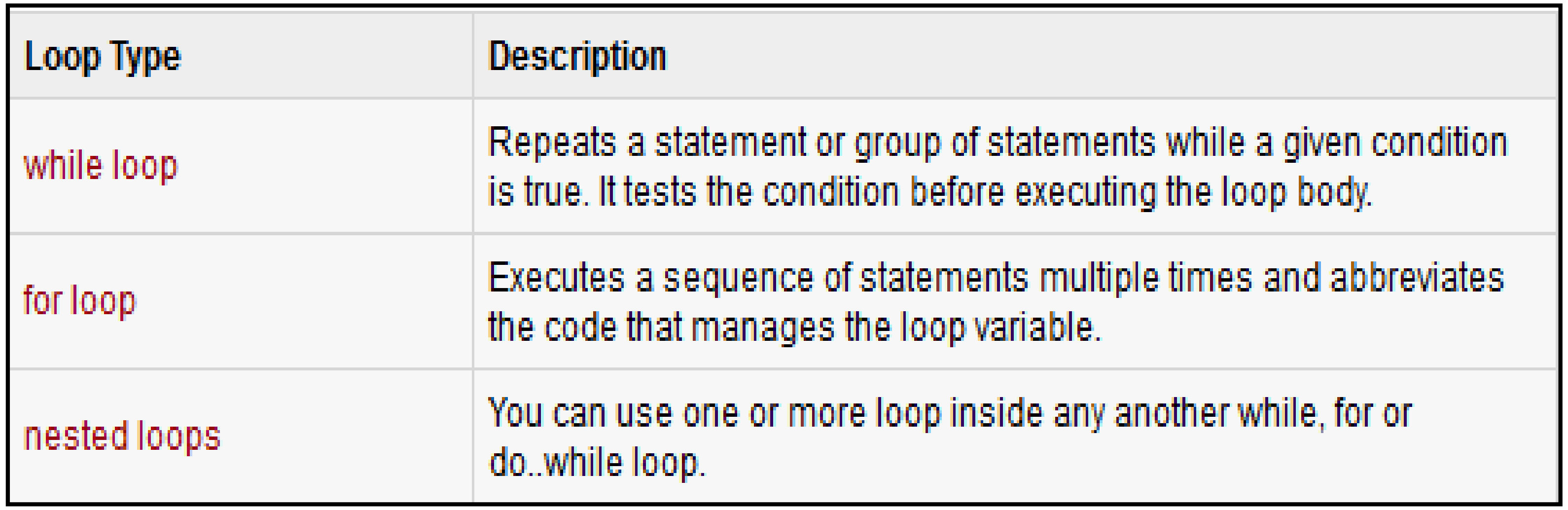
Iteration

WHAT IS A LOOP ?

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages:



TYPES OF LOOPS



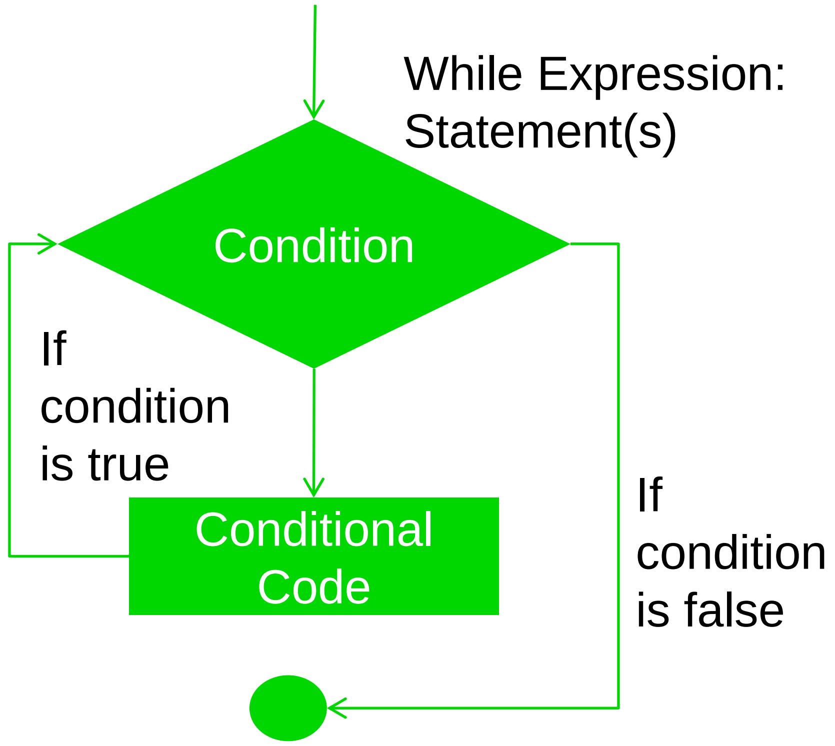
Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loop inside any another while, for or do..while loop.

WHILE LOOP

A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

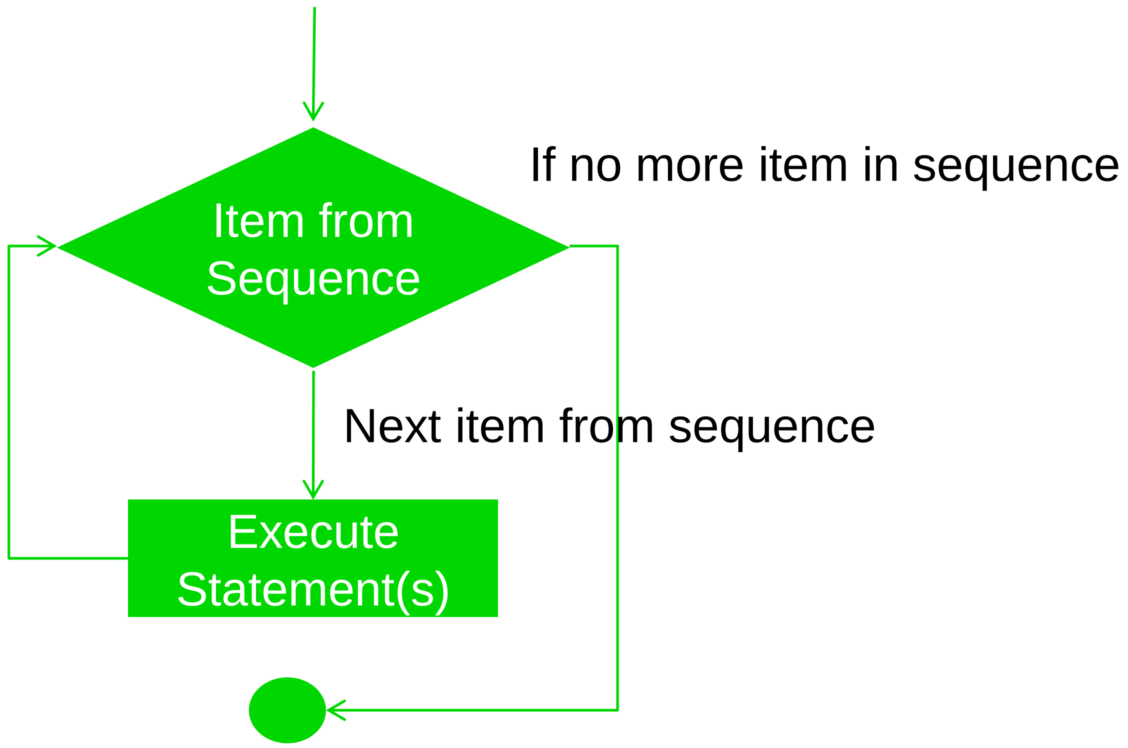
The syntax of a while loop in Python programming language is:

```
while expression:
statement(s)
```



EXAMPLE

FOR LOOP


If no more item in sequence
Item from Sequence
Next item from sequence
Execute Statement(s)

EXAMPLE

```
for letter in 'Python': # First Example
                        print 'Current Letter :', letter
fruits = ['banana', 'apple', 'mango']
for fruit in fruits: # Second Example
                        print 'Current fruit :', fruit
print "Good bye!“
```

OUTPUT:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

DECISION MAKING

Statement	Description
if statements	An **if statement** consists of a boolean expression followed by one or more statements.
if...else statements	An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is FALSE.
nested if statements	You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

PYTHON LOOPS

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loop inside any another while, for or do..while loop.

LOOP CONTROL STATEMENTS

Control Statement	Description
break statement	Terminates the loop statement and transfers execution to the statement immediately following the loop.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
pass statement	The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.