Basic Python Syntax

By Md Saif Hossain

Saif.ndub@gmail.com

Python Operators

Python Operators: Operators are used to performing operations on variables and values. In the example below, we use the + operator to add together two values:

Example

```
print(10 + 5)
```

Python Operators

Python divides the operators into the following groups:

- ❑ Arithmetic operators
- ❑ Assignment operators
- ❑ Comparison operators
- ❑ Logical operators
- ❑ Identity operators
- ❑ Membership operators
- ❑ Bitwise operators

Python Operators

Arithmetic operators

Operator	Name	Description	Example
+	Addition	Add two operands	x + y
-	Subtraction	Subtracts two operands	x - y
*	Multiplication	Multiplies two operands	x * y
/	Division (float)	Divides the first operand by the second	x / y
//	Floor division	Divides the first operand by the second	x // y
%	Modulus	Returns the remainder when division	x % y
**	Exponentiation	Power: Returns first raised to power second	x ** y

Python Operators

Arithmetic operators

```
a = 32     # Initialize the value of a
b = 6       # Initialize the value of b
print('Addition of two numbers:', a+b)
print('Subtraction of two numbers:', a-b)
print('Multiplication of two numbers:', a*b)
print('Division of two numbers:', a/b)
print('Reminder of two numbers:', a%b)
print('Exponent of two numbers:', a**b)
print('Floor division of two numbers:', a//b)
```

Operator Precedence

Operators	Meaning
()	Parentheses
**	Exponent
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction

```
print(10+3*2**2+45)
```

Python Operators

Assignment Operators

Operator	Name	Example	Same As
=	Assignment	x = 5	x = 5
+=	Addition Assignment	x += 3	x = x + 3
-=	Subtraction Assignment	x -= 3	x = x - 3
*=	Multiplication Assignment	x *= 3	x = x * 3
/=	Division Assignment	x /= 3	x = x / 3
//=	Floor Division Assignment	x //= 3	x = x // 3
%=	Modulus Assignment	x %= 3	x = x % 3
**=	Exponentiation Assignment	x **= 3	x = x ** 3

Python Operators

Assignment Operators

```
x = 5
print(x)

x = 5
x += 3      #same as, x=x+3
print('x=x+3',x)

x = 5
x -= 3      #same as, x=x-3
print('x=x-3',x)
```

```
x = 5
x /= 3      #same as, x=x/3
print('x=x/3',x)

x = 5
x%=3        #same as, x=x%3
print('x=x%3',x)

x = 5
x//=3       #same as, x=x//3
print('x=x//3', x)

x = 5
x **= 3     #same as, x=x**3
print('x=x**3', x)
```

Python Operators

Bitwise Assignment Operators

Operator	Name	Example	Same As
&=	Bitwise AND Assignment	x &= 3	x = x & 3
|=	Bitwise OR Assignment	x |= 3	x = x | 3
^=	Bitwise XOR Assignment	x ^= 3	x = x ^ 3
<<=	Bitwise Left Shift Assignment	x <<= 3	x = x << 3
>>=	Bitwise Right Shift Assignment	x >>= 3	x = x >> 3

Python Operators

Bitwise Assignment Operators

```
x = 5
x &= 3      # x=x&3
print('x=x&3', x)

x = 5
x |= 3      # x=x|3
print('x=x|3', x)

x = 5
x ^= 3      # x=x^3
print('x=x^3', x)
```

```
x = 5
x >>= 3     # x=x>>3
print('x=x>>3', x)

x = 5
b = 3
x <<= b     # x=x<<b
print('x=x<<b', x)
```

Python Operators

❑ Comparison Operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal to	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Operators

- Comparison Operators

```
x = 5
y = 3
print(x == y)
# returns False because 5 is not equal to 3

x = 5
y = 3
print(x != y)
# returns True because 5 is not equal to 3
```

Python Operators

❑ Comparison Operators

```
x = 5
y = 3
print(x > y)
# returns True because 5 is greater than 3

x = 5
y = 3
print(x < y)
# returns False because 5 is not less than 3
```

Python Operators

❑ Comparison Operators

```
x = 5
y = 3
print(x >= y)
# returns True because five is greater, or equal, to 3

x = 5
y = 3
print(x <= y)
# returns False because 5 is neither less than or equal to 3
```

Python Operators

Logical Operators

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, and returns False if the result is true	not False

Python Operators

Logical Operators

```
x = 5
print(x > 3 and x < 10)
# returns True because 5 is greater than 3 AND 5 is
less than 10

x = 5
print(x > 3 or x < 4)
# returns True because one of the conditions is
true (5 is greater than 3, but 5 is not less than
4)

x = 5
print(not(x > 3 and x < 10))
# returns False because not is used to reverse the
result
```

Python Operators

- Identity operators

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Python Operators

❑ Identity operators

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is z)
# returns True because z is the same object as x

print(x is y)
# returns False because x is not the same object as y, even if they have
the same content

print(x == y)
# to demonstrate the difference betweeen "is" and "==": this comparison
returns True because x is equal to y
```

Python Operators

❑ Identity operators

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x
print(x is not z)
# returns False because z is the same object as x

print(x is not y)
# returns True because x is not the same object as y, even if
they have the same content

print(x != y)
# to demonstrate the difference between "is not" and "!=":
this comparison returns False because x is equal to y
```

Python Operators

❑ Membership operators

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Python Operators

❑ Membership operators

```
x = ["apple", "banana"]
print("banana" in x)
# returns True because a sequence with the value "banana" is in
the list

x = ["apple", "banana"]
print("pineapple" not in x)
# returns True because a sequence with the value "pineapple" is
not in the list
```

Python Operators

❑ Bitwise operators

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits (Return one’s complement)
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Python Operators

❑ Bitwise operators

```python
a = 10          # initialize the value of a
b = 2          # initialize the value of b
print('Result of a&b:', a&b)
print('Result of a|b:', a|b)
print('Result of a^b:', a^b)
print('Result of ~a:', ~a)
print('Result of a<<b:', a<<b)
print('Result of a>>b:', a>>b)
```

```
Result of a&b: 2
Result of a|b: 10
Result of a^b: 8
Result of ~a: -11
Result of a<<b: 40
Result of a>>b: 2
```

```
a = 10 = 1010 (Binary)

In computers we usually represent numbers using 32 bits,
so binary representation of 10 is (....0000 1010)[32 bits]

~a is basically 1's complement of a
i.e ~a should be ~10 = ~(....0000 1010) = (....1111 0101) = intermediate-result

Since bitwise negation inverts the sign bit,
we now have a negative number. And we represent a negative number
using 2's complement.

2's complement of intermediate-result is:
intermediate-res =  0101      //....1111 0101

                     1010      //....0000 1010 -(1's complement)
                         +1
                 -----------
                   =  1011      //....0000 1011
                  -----------
                   =   -11 (Decimal)

thus ~a = -11
```

Math Functions/Module

Python math Functions

Python Math functions is one of the most used functions in Python Programming. In python there are different built-in math functions. Beside there is also a math module in python.

```
x=2.9
print(round(x))
print(abs(-2.9))
```

Python math Module

Python has a built-in module that you can use for mathematical tasks. The math module has a set of methods and constants.

```
import math

x=2.9
print(math.ceil(x))
print(math.floor(x))
```

https://www.w3schools.com/python/module_math.asp

Random Number/Module

Example: Single Random

```
import random
# Random Number between 1 to 5
x=random.randint(1,5)
print(x)

# Random Number between 0 to 1
y=random.random()
print(y)
```

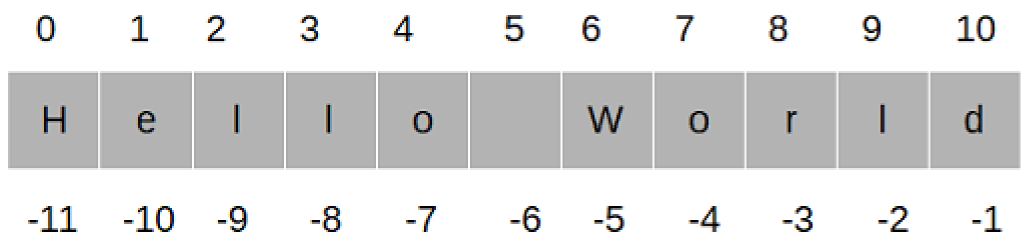
Example: Multiple random

```
m=[]
# Five Random Number between 1 to 50
for i in range(0,5):
   n=random.randint(1,50)
   m.append(n)
   print(m)

# Five Unique Random Number between 1 to 40
y=random.sample(range(1,40),5)
print(y)
```

Strings

Strings are List like many other popular programming languages. Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.



```
a = "Hello World"
print(a)
print(a[0])
print(a[-1])
print(a[0:3])
print(a[0:])
print(a[1:])
print(a[:4])
print(a[0:-1])
print(a[2:-3])
```

Formatted Strings

f-strings are a great new way to format strings. Not only are they more readable, more concise, and less prone to error than other ways of formatting, they are also faster!

```
name = "Eric"
age = 74
# Hello, Eric. You are 74.
print("Hello, " + name + ". You are " + str(age) + ".")
print(f "Hello, {name}. You are {age}.")
```

String Methods/Functions

https://www.w3schools.com/python/python_ref_string.asp

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
count()	Returns the number of times a specified value occurs in a string
find()	Searches the string for a specified value and returns the position of where it was found
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces

String Methods/Functions

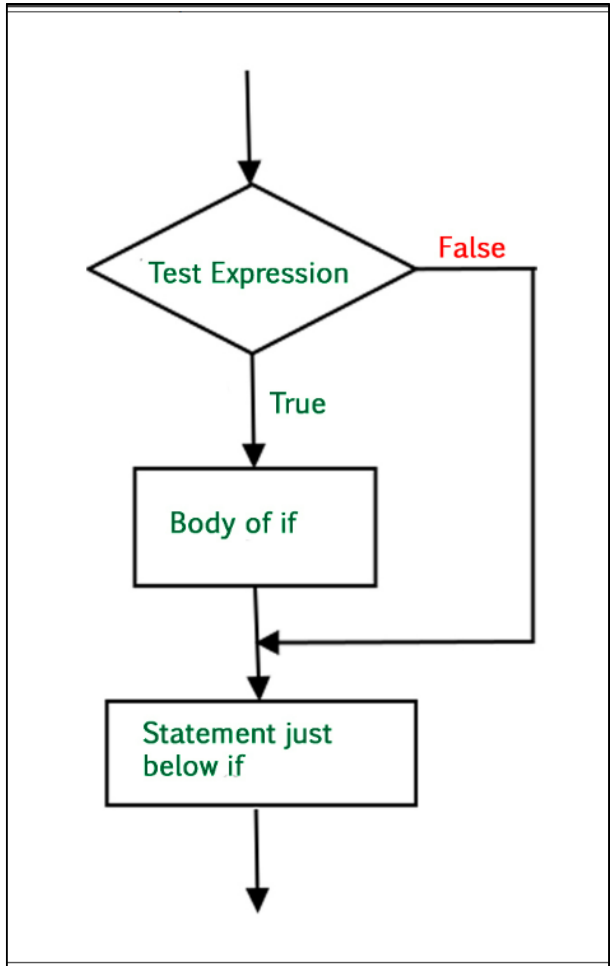
Method	Description
isupper()	Returns True if all characters in the string are upper case
join()	Converts the elements of an iterable into a string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

If Statements

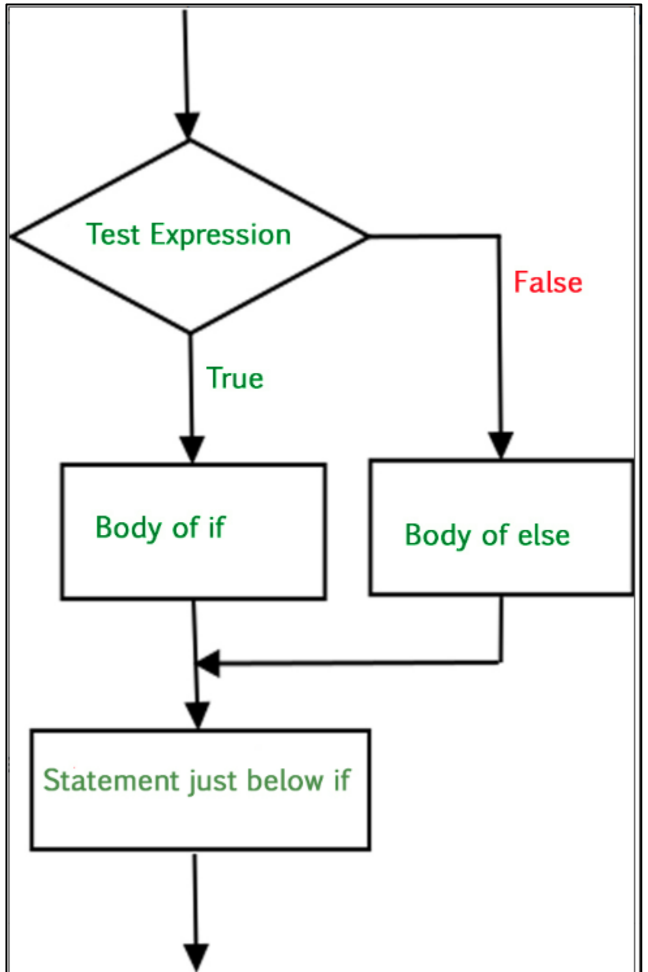
If statement

If statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

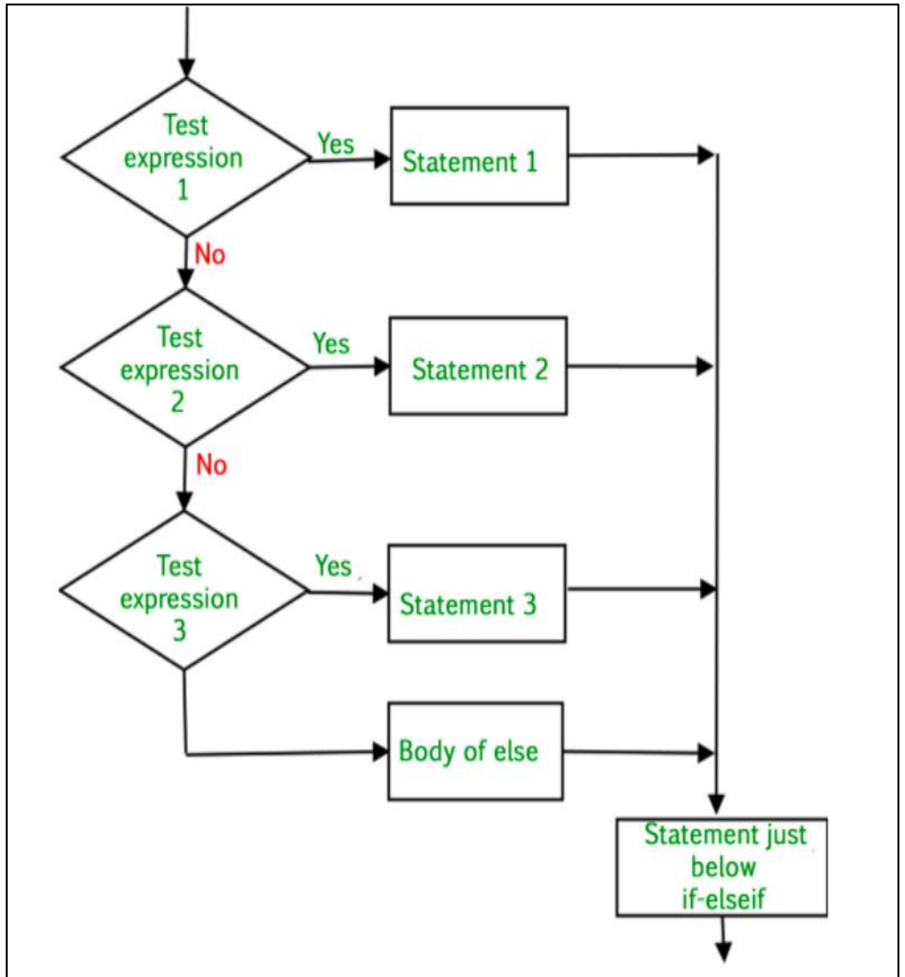
if



if……else



if……elif……else



If Statements

Example

if

```
i = 10

if i > 15:
   print("10 is less than 15")
print("I am Not in if")
```

If……else

```
i = 20
if i < 15:
   print("i is smaller than 15")
   print("i'm in if Block")
else:
   print("i is greater than 15")
   print("i'm in else Block")
print("i'm not in if and not in else Block")
```

If……elif……else

```
i = 20
if i == 10:
   print("i is 10")
elif i == 15:
   print("i is 15")
elif i == 20:
   print("i is 20")
else:
   print("i is not present")
```

Letter Grade Program

```
print("Enter Marks Obtained in 5 Subjects: ")
markOne = int(input())
markTwo = int(input())
markThree = int(input())
markFour = int(input())
markFive = int(input())

tot = markOne+markTwo+markThree+markFour+markFive
avg = tot/5

if avg>=90 and avg<=100:
    print("Your Grade is A+")
elif avg>=80 and avg<90:
    print("Your Grade is A")
elif avg>=70 and avg<80:
    print("Your Grade is B")
elif avg>=60 and avg<70:
    print("Your Grade is C")
elif avg>=50 and avg<60:
    print("Your Grade is F")
```

Leap Year Program

```
yr=int(input("Enter a year:"))

if yr % 4 == 0:
   print("It's a leap year...!!!!")
else:
   print("It's not a leap year....!!!")
```

Ternary Operators

```
num1=20
num2=15

if num1>num2:
    print(num1)
else:
    print(num2)

print(num1 if num1>num2 else num2)
```

Weight Converter Program

```
weight = float(input("Weight: "))
unit = input("(K)g or (L)bs: ")

if unit.upper() == "K":
    converted = weight / 0.4
    print(f"Your weight is {converted} pounds")
else:
    converted = weight * 0.4
    print(f"Your weight is {converted} kilos")
```

Python Strings

Strings: Strings in python are surrounded by either single quotation marks or double quotation marks.

'hello' is the same as "hello". You can display a string literal with the print() function:

Example

```
print("Hello")
print('Hello')
```

Python Strings

Assign String to a Variable: Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

```
Example
a = "Hello"
print(a)
```

Python Strings

Multiline Strings: You can assign a multiline string to a variable by using three quotes:

```
Example
a = """Lorem ipsum dolor sit
amet,consectetur adipiscing elit,sed
do eiusmod tempor incididuntut
labore et dolore magna aliqua."""
print(a)
```

Python Strings

Strings are Arrays: Like many other popular programming languages, strings in Python are arrays of bytes representing Unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

Example

```
a = "Hello, World!"
print(a[1])
```

Python Strings

Looping Through a String: Since strings are arrays, we can loop through the characters in a string, with a for loop.

```
Example
for x in "banana":
  print(x)
```

String Length: To get the length of a string, use the len() function. The len() function returns the length of a string:

```
Example
a = "Hello, World!"
print(len(a))
```

Python Strings

Check String: To check if a certain phrase or character is present in a string, we can use the keyword in.

```
Example
txt = "The best things in life are free!"
print("free" in txt)
```

Use it in an if statement. Print only if "free" is present:

```
Example
txt = "The best things in life are!"
if "free" in txt:
   print("Yes, 'free' is present.")
else:
   print("Not Present")
```

Python Strings

Slicing Strings: you can return a range of characters by using the slice syntax.
Specify the start index and the end index, separated by a colon, to return a part of the string.

```
Example
b = "Hello, World!"
print(b[2:5])
```

Slice From the Start: By leaving out the start index, the range will start at the first character:

```
Example
b = "Hello, World!"
print(b[:5])
```

Python Strings

Negative Indexing: Use negative indexes to start the slice from the end of the string:

```
Example
b = "Hello, World!"
print(b[-5:-2])
```

Python Strings

Modify Strings: Python has a set of built-in methods that you can use on strings.

Upper Case: The upper() method returns the string in upper case:

```
Example
a = "Hello, World!"
print(a.upper())
```

Lower Case: The lower() method returns the string in lower case:

```
Example
a = "Hello, World!"
print(a.lower())
```

Python Strings

Modify Strings: **Remove Whitespace:** Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

Example
```
a = "   Hello, World!     "
print(a.strip()) #returns "Hello, World!"
```

Replace String: The replace() method replaces a string with another string:

Example
```
a = "Hello, World!"
print(a.replace("H", "J"))
```

Python Strings

Modify Strings: **Split String:** The split() method returns a list where the text between the specified separator becomes the list items. The split() method splits the string into substrings if it finds instances of the separator:

Example
```
a = "Hello, World!"
print(a.split(","))
# returns ['Hello', ' World!']
```

String Concatenation: To concatenate, or combine, two strings you can use the + operator.

Example
```
a = "Hello"
b = "World"
c = a + b
print(c)
```

Example (Add “ ”)
```
a = "Hello"
b = "World"
c = a + " " + b
print(c)
```

Python Strings

String Format: As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

Example

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

But we can combine strings and numbers by using the format() method! The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

Example: Use the format() method to insert numbers into strings:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

Python Strings

String Format: The format() method takes an unlimited number of arguments, and are placed into the respective placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {}
Taka."
print(myorder.format(quantity, itemno, price))
```

Python Strings

String Format: You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

Python Strings

Escape Character: To insert characters that are illegal in a string, use an escape character. An escape character is a backslash \ followed by the character you want to insert. An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

Example: You will get an error if you use double quotes inside a string that is surrounded by double quotes:

```
txt = "We are the so-called "Vikings"
from the north."
```

#You will get an error if you use double quotes inside a string that are surrounded by double quotes:

Python Strings

Escape Character: To fix this problem, use the escape character \":

Example: The escape character allows you to use double quotes when you normally would not be allowed:

```
txt = "We are the so-called \"Vikings\" from the north."
```

Python Strings

Escape Characters:

Code	Result
\'	Single Quote
\\	Backslash
\n	New Line
\r	Carriage Return

```
txt = 'It\'s alright.'
print(txt)

txt = "This will insert one \\
(backslash)."
print(txt)

txt = "Hello\nWorld!"
print(txt)

txt = "Hello\rWorld!"
print(txt)
```

Python Strings

Escape Characters:

Code	Result
\t	Tab
\b	Backspace
\ooo	Octal value
\xhh	Hex value

```
txt = "Hello\tWorld!"
print(txt)

#This example erases one character
(backspace):
txt = "Hello \bWorld!"
print(txt)

#A backslash followed by three integers
will result in a octal value:
txt = "\110\145\154\154\157"
print(txt)

#A backslash followed by an 'x' and a hex
number represents a hex value:
txt = "\x48\x65\x6c\x6c\x6f"
print(txt)
```