PYTHON FOR BEGINNERS 1.2

NEHAL JAIN

QCFINANCE.IN

CONTENT

- How to print without newline in Python?
- Decision Making in Python
- Python Functions
- Python Lambda
- Python Arrays

How to print without newline in Python?

- Python has a predefined format if you use print(a_variable) then it will **go to next line automatically.**
- But sometime it may happen that we don’t want to go to next line but want to print on the same line. So what we can do?

How to print without newline in Python?

How to print without newline(i.e. in the same line) in Python?

In [5]:
```
print("hi")
print("world")
```

```
hi
world
```

In [2]:
```
# array
a = [1, 2, 3, 4]

# printing a element in same
# line
for i in range(4):
    print(a[i])
```

```
1
2
3
4
```

In [3]:
```
print("hi"),
print("world")
```

```
hi world
```

In [1]:
```
# array
a = [1, 2, 3, 4]

# printing a element in same
# line
for i in range(4):
    print(a[i]),
```

```
1 2 3 4
```

Python Conditions

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.

Decision Making in Python

- Decision making statements in programming languages decides the direction of flow of program execution. Decision making statements available in python are:

- if statement
- if..else statements
- nested if statements
- if-elif ladder
- While loop
- For loop

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.

- **Syntax**:

```
if condition:
    # Statements to execute if
    # condition is true
```

Indentation

- Python relies on indentation (whitespace at the beginning of a line) to define scope in the code.
- Other programming languages often use curly-brackets for this purpose.

Indentation error due to no space after starting a loop

In [1]:
```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

```
  File "<ipython-input-1-4276c1871af7>", line 4
    print("b is greater than a") # you will get an error
        ^
IndentationError: expected an indented block
```

In [3]:
```
a = 33
b = 200
if b > a:
    print("b is greater than a") # you will not get an error
```

```
b is greater than a
```

if- else

- The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't.
- But what if we want to do something else if the condition is false.
- Here comes the *else* statement. We can use the *else* statement with *if* statement to execute a block of code when the condition is false.

Syntax:

```
if (condition):
    # Executes this block if
    # condition is true
else:
    # Executes this block if
    # condition is false
```

nested-if

- A nested if is an if statement that is the target of another if statement.
- Nested if statements means an if statement inside another if statement.
- Yes, Python allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.
- **Syntax**:

```
if (condition1):
   # Executes when condition1 is true
   if (condition2):
      # Executes when condition2 is true
   # if Block is end here
# if Block is end here
```

code

```
# python program to illustrate nested If statement

i = 10
    if (i == 10):
# First if statement
    if (i < 15):
        print ("i is smaller than 15")
# Nested - if statement Will only be executed if statement above it is true
    if (i < 12):
        print ("i is smaller than 12 too")
    else:
        print ("i is greater than 15")
```

```
i is smaller than 15
i is smaller than 12 too
```

code

if-else-if loop program

```
#taking input from user and assigning to variable x

    x = int(input("Please enter an integer: "))

# if-else-if loop

    if x < 0:
        x = 0
        print('Negative changed to zero')
    elif x == 0:
        print('Zero')
    elif x == 1:
        print('Single')
    else:
        print('greater than zero')
```

```
Please enter an integer: 8
greater than zero
```

While loop

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

```
while loop
```

In [5]:
```
# Fibonacci series:
 # the sum of two elements defines the next

    a, b = 0, 1
while a < 10:
    print(a)
        a, b = b, a+b
```

```
0
1
1
2
3
5
8
```

for Statements

- Python's ***for*** statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.

for loop

```
# Measure some strings:

words = ['cat', 'window', 'defenestrate']
for w in words:
    print(w, len(w))
```

```
('cat', 3)
('window', 6)
('defenestrate', 12)
```

for

- If you need to modify the sequence you are iterating over while inside the loop (for example to duplicate selected items), it is recommended that you first make a copy.
- Iterating over a sequence does not implicitly make a copy. The slice notation makes this especially convenient:

In [25]:
```
>>> for w in words[:]:  # Loop over a slice copy of the entire list.
...     if len(w) > 6:
...         words.insert(0, w)
...
>>> words
```

Out[25]:
```
['defenestrate', 'cat', 'window', 'defenestrate']
```

The range() Function

- If you do need to iterate over a sequence of numbers, the built-in function range() comes in handy. It generates arithmetic progressions:

range function

```
>>> for i in range(5):
...     print(i)
...
```

```
0
1
2
3
4
```

```
for i in range(1):
    print(i)
```

```
0
```

range

- The given end point is never part of the generated sequence;
- range(10) generates 10 values, the legal indices for items of a sequence of length 10.
- It is possible to let the range start at another number, or to specify a different increment (even negative; sometimes this is called the 'step'):

```
In [35]: range(2,9)
Out[35]: [2, 3, 4, 5, 6, 7, 8]

In [36]: range(0,10,2)
Out[36]: [0, 2, 4, 6, 8]

In [38]: range(-10, -20, -2)
Out[38]: [-10, -12, -14, -16, -18]
```

range() and len()

- To iterate over the indices of a sequence, you can combine range() and len() as follows:

In [39]:
```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print(i, a[i])
...
```

```
(0, 'Mary')
(1, 'had')
(2, 'a')
(3, 'little')
(4, 'lamb')
```

break and continue Statements, and else Clauses on Loops

- The *break* statement, breaks out of the innermost enclosing for or while loop.
- Loop statements may have an else clause; it is executed when the loop terminates through exhaustion of the list (with for) or when the condition becomes false (with while), but not when the loop is terminated by a break statement.
- This is exemplified by the following loop, which searches for prime numbers:

code

In [43]:
```
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print(n, 'equals', x, '*', n//x)
            break
    else:
        # loop fell through without finding a factor
        print(n, 'is a prime number')
```

```
(2, 'is a prime number')
(3, 'is a prime number')
(4, 'equals', 2, '*', 2)
(5, 'is a prime number')
(6, 'equals', 2, '*', 3)
(7, 'is a prime number')
(8, 'equals', 2, '*', 4)
(9, 'equals', 3, '*', 3)
```

Python Functions

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.

Creating and Calling a Function

- In Python a function is defined using the ***def*** keyword:
- To call a function, use the function name followed by parenthesis:

Python functions

creating a function

In [5]:
```
def my_function():
    print("Hello from a function")
```

calling a funtion

In [6]:
```
def my_function():
  print("Hello from a function")

my_function()
```

```
Hello from a function
```

Function Parameters

- Information can be passed to functions as parameter.
- Parameters are specified after the function name, inside the parentheses.
- You can add as many parameters as you want, just separate them with a comma.
- The following example has a function with one parameter (fname). When the function is called, we pass along a first word, which is used inside the function to print the full sentence:

code

function parameters

In [9]:

```
def my_function(fname):
  print(fname + " function")

my_function("hi")
my_function("hello")
my_function("welcome to new")
```

```
hi function
hello function
welcome to new function
```

Default Parameter Value

- The following example shows how to use a default parameter value.
- If we call the function without parameter, it uses the default value:

default parameter value

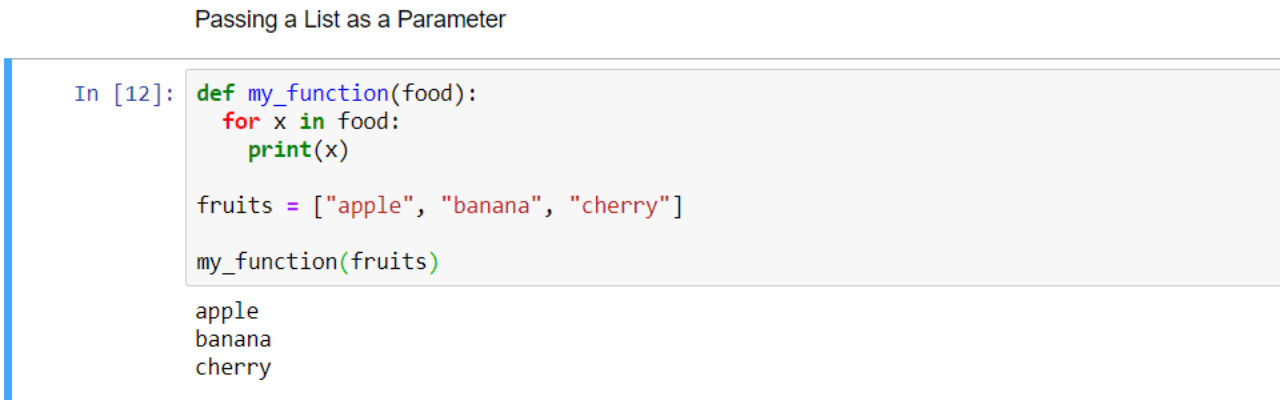
In [11]:
```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("India")
my_function("New York")
my_function()
my_function("Brazil")
```

```
I am from India
I am from New York
I am from Norway
I am from Brazil
```

Passing a List as a Parameter

- You can send any data types of parameter to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.
- E.g. if you send a List as a parameter, it will still be a List when it reaches the function:



Return Values

- To let a function return a value, use the ***return*** statement:

Return Values

```
In [13]: def my_function(x):
           return 5 * x

         print(my_function(3))  # print 5*3 i.e. 15
         print(my_function(5))  # print 5*5
         print(my_function(9))  # print 5*9
```

```
15
25
45
```

Keyword Arguments

- You can also send arguments with the *key* = *value* syntax.
- This way the order of the arguments does not matter.

Keyword Arguments

```
In [15]: def my_function(child3, child2, child1):
           print("The youngest child is " + child3)

         my_function(child1 = "Alex", child2 = "John", child3 = "Niti")

         The youngest child is Niti
```

- The phrase *Keyword Arguments* are often shortened to *kwargs* in Python documentations.

Arbitrary Arguments

- If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.
- This way the function will receive a *tuple* of arguments, and can access the items accordingly:

Arbitrary Arguments

In [20]:
```
def my_function(*kids):
    print("The youngest child is " + kids[2])
    print("The eldest child is " + kids[0])

my_function("Alex", "John", "Niti")
```

```
The youngest child is Niti
The eldest child is Alex
```

Recursion

- Python also accepts function recursion, which means a defined function can call itself.
- It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.
- The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power.
- However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.
- In this example, tri_recursion() is a function that we have defined to call itself ("recurse"). We use the k variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).
-

Code

Recursion Example

In [33]:
```
def tri_recursion(k):
  if(k>0):
    result = k+tri_recursion(k-1)
    print(result)
  else:
    result = 0
  return result

print("Recursion Example Results when k>0")
tri_recursion(6)

print("Recursion Example Results when k<0")
tri_recursion(-5)
```

```
Recursion Example Results when k>0 1
3
6
10
15
21
Recursion Example Results when k<0
```

Out[33]:
```
0
```

Python Lambda

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.

- Syntax
- lambda *arguments* : *expression*

- The expression is executed and the result is returned:

- A lambda function that adds 10 to the number passed in as an argument, and print the result:

Python Lambda

```
In [35]: x = lambda a : a + 10
         print(x(5))

         15
```

- Lambda functions can take any number of arguments:

Example

A lambda function that multiplies argument a with argument b and print the result:

```
In [36]: x = lambda a, b : a * b
         print(x(5, 6))

         30
```

- A lambda function that sums argument a, b, and c and print the result:

```
In [37]: x = lambda a, b, c : a + b + c
         print(x(5, 6, 2))

         13
```

Why Use Lambda Functions?

- The power of lambda is better shown when you use them as an anonymous function inside another function.
- Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

```
In [38]: def myfunc(n):
           return lambda a : a * n
```

- Use that function definition to make a function that always doubles the number you send in:

In [39]:
```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

```
22
```

- Or, use the same function definition to make a function that always *triples* the number you send in:

In [40]:
```
def myfunc(n):
  return lambda a : a * n

mytripler = myfunc(3)

print(mytripler(11))
```

```
33
```

- Or, use the same function definition to make both functions, in the same program:

In [41]:
```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

```
22
33
```

- Note - Use lambda functions when an anonymous function is required for a short period of time.

Python Arrays

- Arrays are used to store multiple values in one single variable
- An array is defined as a collection of items that are stored at contiguous memory locations.
- It is a container which can hold a fixed number of items, and these items should be of the same type.
- Array is an idea of storing multiple items of the same type together and it makes easier to calculate the position of each element by simply adding an offset to the base value.
- A combination of the arrays could save a lot of time by reducing the overall size of the code.
- It is used to store multiple values in single variable.

- If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:
- car1 = "Ford"
  car2 = "Volvo"
  car3 = "BMW“
- However, what if you want to loop through the cars and find a specific one?
- And what if you had not 3 cars, but 300? The solution is an array!
- An array can hold many values under a single name, and you can access the values by referring to an index number.

- Following are the terms to understand the concept of an array:

Element - Each item stored in an array is called an element.

Index - The location of an element in an array has a numerical index, which is used to identify the position of the element.

Array Representation

An array can be declared in various ways and different languages. The important points that should be considered are as follows:

- Index starts with 0.
- We can access each element via its index.
- The length of the array defines the capacity to store the elements.

Array

In [43]:
```
cars = ["Ford", "Volvo", "BMW"]

x = cars[0]

print(x)
```

```
Ford
```

Array operations

Some of the basic operations supported by an array are as follows:

- **Traverse -** It prints all the elements one by one.
- **Insertion** - It adds an element at the given index.
- **Deletion** - It deletes an element at the given index.
- **Search** - It searches an element using the given index or by the value.
- **Update -** It updates an element at the given index.

- The Array can be created in Python by importing the array module to the python program.

Accessing array elements

- We can access the array elements using the respective indices of those elements.

```
In [45]: import array as arr
         a = arr.array('i', [12,54, 36, 81])
         print("First element:", a[0])
         print("Second element:", a[1])
         print("Second last element:", a[-1])

         ('First element:', 12)
         ('Second element:', 54)
         ('Second last element:', 81)
```

- **Explanation:** In the above example, we have imported an array, defined a variable named as "a" that holds the elements of an array and print the elements by accessing elements through indices of an array.

How to change or add elements

- Arrays are mutable, and their elements can be changed in a similar way like lists.

```
In [46]: import array as arr
         numbers = arr.array('i', [1, 2, 3, 5, 7, 10])

         # changing first element
         numbers[0] = 0
         print(numbers)    # Output: array('i', [0, 2, 3, 5, 7, 10])

         # changing 3rd to 5th element
         numbers[2:5] = arr.array('i', [4, 6, 8])
         print(numbers)    # Output: array('i', [0, 2, 4, 6, 8, 10])

         array('i', [0, 2, 3, 5, 7, 10])
         array('i', [0, 2, 4, 6, 8, 10])
```

- **Explanation:** In the above example, we have imported an array and defined a variable named as "numbers" which holds the value of an array. If we want to change or add the elements in an array, we can do it by defining the particular index of an array where you want to change or add the elements.

Why to use arrays in Python?

- A combination of arrays saves a lot of time. The array can reduce the overall size of the code.

How to delete elements from an array?

- The elements can be deleted from an array using Python's **del** statement.
- If we want to delete any value from the array, we can do that by using the indices of a particular element.

```
In [47]: import array as arr

         number = arr.array('i', [1, 2, 3, 3, 4])

         del number[2]                    # removing third element
         print(number)                    # Output: array('i', [1, 2, 3, 4])

         array('i', [1, 2, 3, 4])
```

- **Explanation:** In the above example, we have imported an array and defined a variable named as "number" which stores the values of an array. Here, by using del statement, we are removing the third element [3] of the given array.

Finding the length of an array

- The length of an array is defined as the number of elements present in an array.
- It returns an integer value that is equal to the total number of the elements present in that array.

Syntax

- len(array_name)

```
In [49]: cars = ["Ford", "Volvo", "BMW","Audi"]

         x = len(cars)

         print(x)
```

```
4
```

Array Concatenation

- We can easily concatenate any two arrays using the + symbol.

```
In [50]: a=arr.array('d',[1.1 , 2.1 ,3.1,2.6,7.8])
         b=arr.array('d',[3.7,8.6])
         c=arr.array('d')
         c=a+b
         print("Array c = ",c)

         ('Array c = ', array('d', [1.1, 2.1, 3.1, 2.6, 7.8, 3.7, 8.6]))
```

- **Explanation**
- In the above example, we have defined variables named as "a, b, c" that hold the values of an array.

- **Explanation:** In the below example, first, we have imported an array and defined a variable named as "x" which holds the value of an array and then, we have printed the elements using the indices of an array.

In [51]:
```
import array as arr
x = arr.array('i', [4, 7, 19, 22])
print("First element:", x[0])
print("Second element:", x[1])
print("Second last element:", x[-1])
```

```
('First element:', 4)
('Second element:', 7)
('Second last element:', 22)
```

Looping Array Elements

- You can use the for in loop to loop through all the elements of an array.

```
In [52]: cars = ["Ford", "Volvo", "BMW"]

for x in cars:
  print(x)
```

```
Ford
Volvo
BMW
```

Array Methods

- Python has a set of built-in methods that you can use on lists/arrays.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

THANK YOU