Python tutorials

Sunday, 13 February 2022 9:06 pm

Questions :
(1) if __name__ == "__main__":

#	Name	Description
1	Intro	Python-programming language. Created by Guido van Rossum, and released in 1991. Used for web development(server side), s/w development, maths and scripting. Why python: - works on different platform (Windows, Linux, Mac, Rasberry, Pi, etc) - simple syntax, similar to English language - fewer lines of code and runs on an interpreter system (prototyping can be very quick). - python can be treated in procedural way, an object-oriented way or a functional way.

2	HelloWorld Program	def print_hi():
		    print("Hello World!!!!")
		
		if __name__ == "__main__":
		    print_hi()

3	Command line options	--python version python --version or py version --running python python <python file name> --exit command line exit()

4	Syntax/Indentation/Variable	Syntax:
		  syntax can be executed by writing directly in the Command Line
		  (Or), by executing python script
		   e.g. switch to the python exe location
		        > print("Hello World")
		        > python helloWorld.py
		Indentation:
		  Indentation refers to the spaces at the beginning of a code line.
		  indentation uses to indicate the block of code.
		  if 5< 2:
		        print("condition satisfied")
		Variable:
		  variables are created when you assign a value to it.
		  e.g. x = 5;
		     y =" Hello World"

5	Comment Single/Multi line	1. Using # 2. Using triple quotes
6	Variables	1. No declaration required for variable. 2. Variables are created the moment we first assign a value to it. 3. single/double quotes -> string variables can be declared either by single or double quotes. 4. case-sensitive - variable names are case sensitive. ---- Variable Name: must start with letter or "_". it can contain only alpha-numeric charracters and "_". its case sensitive.

```
Multi Words Variable Names:
Camel Case :
myVariableName = "John "
Pascal Case
MyVariableName = "John"
Snake Case
my_variable_name = "John"

---
#Many values to multiple variables
x, y, z = "Orange", "Apple", "Banana"
print ( "x = " + x + "  : y = " + y + "  :  z = " + z )

#same value to multiple variable
x=y=z ="same value"
print ( "x = " + x + "  : y = " + y + "  :  z = " + z )


#Unpack collections
fruits = ["Apple", "Orange", "Banana"]

a,b,c = fruits
print ( "a = " + a + "  : b = " + b + "  :  c = " + c )

#output variables
print("value of a is " + a )
print ("a + b  = "  + a+b)  #same type of variables can be added.
#but string and int CAN NOT be added.


#global variable
g_var = ""
def sample_fun():
    g_var = "global variable value"

def sample_fun2():
    global g_var2;
    g_var2 = "global variable value 2"

sample_fun()
sample_fun2()

print("global variable example")
print(g_var)
print(g_var2)
```

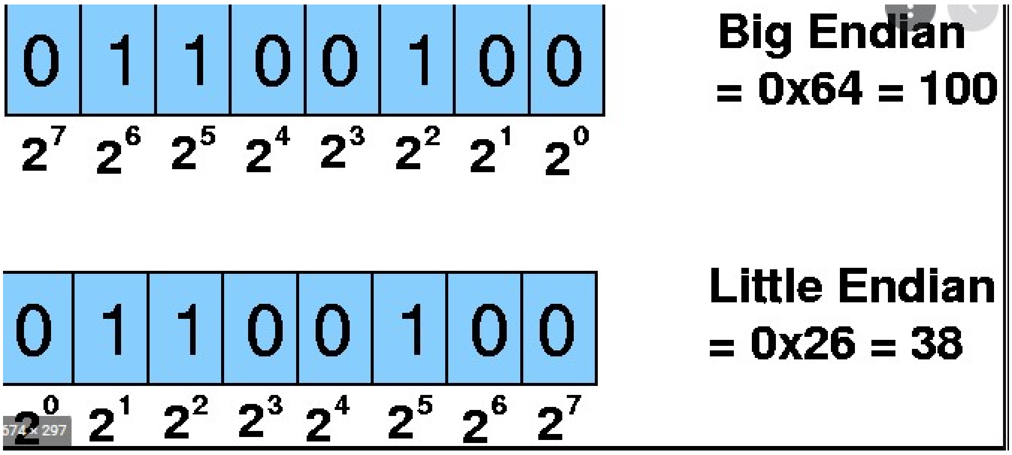
7 Data types

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview

(e.g)

x = "Hello World"	str
x = 20	int
x = 20.5	float

x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview



8 casting

int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals
List()
Tuple()

9 String
(all functions are not implemented)

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map	Formats specified values in a string

()	
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
isidentifier()	Returns True if the string is an identifier
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
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the 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

10 Operators

Arithmetic Operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y

**	Exponentiation	x ** y
//	Floor division	x // y

Assignment Operators

Operator	Example	Same As	
=	x = 5	x = 5	
+=	x += 3	x = x + 3	
-=	x -= 3	x = x - 3	
*=	x *= 3	x = x * 3	
/=	x /= 3	x = x / 3	
%=	x %= 3	x = x % 3	
//=	x //= 3	x = x // 3	
**=	x **= 3	x = x ** 3	
&=	x &= 3	x = x & 3	
|=	x |= 3	x = x | 3	
^=	x ^= 3	x = x ^ 3	Exclusive OR
>>=	x >>= 3	x = x >> 3	
<<=	x <<= 3	x = x << 3	

Comparison Operators

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

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, returns False if the result is true	not(x < 5 and x < 10)

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

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	

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
<<	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 bit

11 Lists

Lists
-used to store multiple items in a single variable.
- one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
-are created using square brackets:
- items are ordered, changeable, and allow duplicate values.
-items are indexed, the first item has index [0], the second item has index [1] etc.
- hetrogenious

<add few more notes from code>

#list comprehension condition syntax
List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

newlist = [expression for item in iterable if condition == True]
e.g
listNew = [item for item in listFruits if "a" in item]

#Customize Sort Function
You can also customize your own function by using the keyword argument key = function.

#Join list - join two list
#1. use + operator
#2. use append function
#3. use extend method

List methods

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 item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

12 Tuples

Tuples are used to store multiple items in a single variable. A tuple is a collection which is ordered and unchangeable. hetrogenious
Tuples are written with round brackets.

#updatetuplevaleus

#changetolist,updateandmaketuple

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

13 Sets

Sets are used to store multiple items in a single variable.
A set is a collection which is unordered, unchangeable*, and unindexed. duplicates not allowed.

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

14 Dictionaries

Dictionary
Dictionaries are used to store data values in key:value pairs.
A dictionary is a collection which is ordered*, changeable and do not allow duplicates (for keys).
(As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.)
Dictionaries are written with curly brackets, and have keys and values:
Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.
The values in dictionary items can be of any data type:

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key

<table>
<tr><th></th><th></th><th></th></tr>
<tr><td></td><td></td><td><table>
<tr><td>popitem()</td><td>Removes the last inserted key-value pair</td></tr>
<tr><td>setdefault()</td><td>Returns the value of the specified key. If the key does not exist: insert the key, with the specified value</td></tr>
<tr><td>update()</td><td>Updates the dictionary with the specified key-value pairs</td></tr>
<tr><td>values()</td><td>Returns a list of all the values in the dictionary</td></tr>
</table></td></tr>
<tr><td>15</td><td>If Else</td><td>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.</td></tr>
<tr><td>16</td><td>For/While Loop</td><td>Python has two primitive loop commands:
• while loops
• for loops</td></tr>
<tr><td>17</td><td>Function/Lambda function</td><td>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.

<b>Arbitrary Arguments : *args</b>
If we are not sure on the number of arguments, add * before the parameter name. In this way, function will receive tuple of arguments.

Keyword Arguments
we can send arguments with the key = value syntax. order of the arguments does not matter.

Arbitrary Keyword Arguments :**kwargs
if we do not know the number arguments and need to receive arguments as dictionary, prepend ** before the argument name.

Recursive Function
function calls itself. ensure function has the correct condition to terminate, otherwise function will never stops. Further, recursive function will consume much amount of memory as well as processing power.

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

Why lambda functions:

lambda function is very useful when we need to use an anonymous function in side another function. i.e. function will return another function as result. i.e. all the parameter values are not available in the earlier stage.</td></tr>
<tr><td>18</td><td>Class Objects</td><td>Python Classes/Objects
Python is an object oriented programming language.
Almost everything in Python is an object, with its properties and methods. A Class is a "blueprint" for creating objects.

The __init__() function is called automatically every time the class is being used to create a new object.

Object Methods
==============</td></tr>
</table>

<table>
<tr><td></td><td></td><td>The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.
It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:</td></tr>
<tr><td>19</td><td>Inheritance</td><td>Inheritance allows us to define a class that inherits all the methods and properties from another class.
Parent class is the class being inherited from, also called base class.
Child class is the class that inherits from another class, also called derived class.

#creating child class
while creating a class that inherits the functionality from another class, send the parent class as parameter when creating the child class.

List, tuples, dictionaries and sets are iterable objects. Since these are iterable objects, we can get the iterator from and all these objects have iter() method that will give iterator.</td></tr>
<tr><td>20</td><td>Iterator</td><td>An iterator is an object that can be iterated upon, meaning that you can traverse through all the values.

In Python, an iterator is an object which implements the iterator protocol, which consist of the methods __iter__() and __next__().</td></tr>
<tr><td>21</td><td>Scope</td><td>A variable is only available from inside the region it is created. This is called scope.
local scope:
- variable declared inside the function can't be accessed outside of it
- local variable can be accessed from a function within the function
global scope:
A variable created in the main body of the Python code is a global variable and belongs to the global scope. Global variables are available from within any scope, global and local. If same variable initialized inside and outside of a function, python will threat them as two separate variables. use global keyword to refer the global variable inside the function for modification.</td></tr>
<tr><td>22</td><td>Modules</td><td>1. Create file with common functions
2. Import the modules in new file and re-use it</td></tr>
<tr><td>23</td><td>Dates</td><td>import a module named datetime to work with dates as date objects</td></tr>
<tr><td>24</td><td>Math</td><td></td></tr>
<tr><td>25</td><td>Json</td><td>
<table>
<tr><th>Python</th><th>JSON</th></tr>
<tr><td>dict</td><td>Object</td></tr>
<tr><td>list</td><td>Array</td></tr>
<tr><td>tuple</td><td>Array</td></tr>
<tr><td>str</td><td>String</td></tr>
<tr><td>int</td><td>Number</td></tr>
<tr><td>float</td><td>Number</td></tr>
<tr><td>True</td><td>true</td></tr>
<tr><td>False</td><td>false</td></tr>
<tr><td>None</td><td>null</td></tr>
</table>
</td></tr>
<tr><td>26</td><td>RegEx</td><td>A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern. RegEx can be used to check if a string contains the specified search pattern. Python has a built-in package called re, which can be used to work with Regular Expressions.
<table>
<tr><th>Function</th><th>Description</th></tr>
<tr><td>findall</td><td>Returns a list containing all matches</td></tr>
<tr><td>search</td><td>The search() function searches the string for a match, and returns a Match object if there is a match. If there is more than one match, only the first occurrence of the match will be returned. If no match, None will be returned.</td></tr>
<tr><td>split</td><td>Returns a list where the string has been split at each match</td></tr>
</table>
</td></tr>
</table>

sub	Replaces one or many matches with a string

The Match object has properties and methods used to retrieve information about the search, and the result:

.span() returns a tuple containing the start-, and end positions of the match.
.string returns the string passed into the function
.group() returns the part of the string where there was a match

Metacharacters

Metacharacters are characters with a special meaning:

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	String Starts with	"^hello"
$	String Ends with	"planet$"
*	Zero or more occurrences	"he.*o"
+	One or more occurrences	"he.+o"
?	Zero or one occurrences	"he.?o"
{}	Exactly the specified number of occurrences	"he.{2}o"
|	Either or	"falls|stays"

Special Sequences

A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

Character	Description	Example
\A	Returns a match if the specified characters are at the beginning of the string	"\AThe"
\b	Returns a match where the specified characters are at the beginning or at the end of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\bain" r"ain\b"
\B	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\Bain" r"ain\B"
\d	Returns a match where the string contains digits (numbers from 0-9)	"\d"
\D	Returns a match where the string DOES NOT contain digits	"\D"
\s	Returns a match where the string contains a white space character	"\s"
\S	Returns a match where the string DOES NOT contain a white space character	"\S"
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	"\w"
\W	Returns a match where the string DOES NOT contain any word characters	"\W"
\Z	Returns a match if the specified characters are at the end of the string	"Spain\Z"

<table>
<tr><td></td><td></td><td>

<b>Sets</b>
A set is a set of characters inside a pair of square brackets [] with a special meaning:
<table>
<tr><th>Set</th><th>Description</th></tr>
<tr><td>[arn]</td><td>Returns a match where one of the specified characters (a, r, or n) are present</td></tr>
<tr><td>[a-n]</td><td>Returns a match for any lower case character, alphabetically between a and n</td></tr>
<tr><td>[^arn]</td><td>Returns a match for any character EXCEPT a, r, and n</td></tr>
<tr><td>[0123]</td><td>Returns a match where any of the specified digits (0, 1, 2, or 3) are present</td></tr>
<tr><td>[0-9]</td><td>Returns a match for any digit between 0 and 9</td></tr>
<tr><td>[0-5][0-9]</td><td>Returns a match for any two-digit numbers from 00 and 59</td></tr>
<tr><td>[a-zA-Z]</td><td>Returns a match for any character alphabetically between a and z, lower case OR upper case</td></tr>
<tr><td>[+]</td><td>In sets, +, *, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string</td></tr>
</table>
</td></tr>
<tr><td>27</td><td>Try…Except</td><td>The try block lets you test a block of code for errors.
The except block lets you handle the error.
The else block lets you execute code when there is no error.
The finally block lets you execute code, regardless of the result of the try- and except blocks.</td></tr>
<tr><td>28</td><td>User Input and String formatting</td><td></td></tr>
<tr><td>29</td><td>File Handling</td><td>File Handling

"r" - Read - Default value. Opens a file for reading, error if the file does not exist
"w" - Write - Opens a file for writing, creates the file if it does not exist
"a" - Append - Opens a file for appending, creates the file if it does not exist
"x" - Create - Creates the specified file, returns an error if the file exists

"t" - Text - Default value. Text mode
"b" - Binary - Binary mode (e.g. images)</td></tr>
</table>