Python – S2

Contents

1. Python Resserved words and Naming Convention
2. Comparison between Python and Java
3. Basic Python Syntax
4. Variables, Expressions and Statements
5. Simple Input and Output
6. The Format method
7. Strings

Python Reserved words

- The following identifiers are used as reserved words, or *keywords* of the language, and cannot be used as ordinary identifiers.

['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

- You can generate this keyword list as :

```
>>> import keyword
>>> keyword.kwlist
```

- So the point here is : Please DO NOT use any of the above words in your program for naming your variables/functions, etc.

Naming Convention in Python

- For Authentic Guideline : PLEASE consult PEP 8 documentation :

https://www.python.org/dev/peps/pep-0008/

- You can also consult Google Python Style Guide :

https://google.github.io/styleguide/pyguide.html

- The basic idea is :

Variable Names : lower_with_underscores

Constants : UPPER_WITH_UNDERSCORES

Function Names : lower_with_underscores

Function Parameters : lower_with_underscores

Class Names : CapitalWords

Method Names : lower_with_underscores

Method Parameters and Variables : lower_with_underscores

  - Always use self as the first parameter to a method

Comparison between Python & Java

Key Features of Java:

- Strong Corporate support from Sun Microsystems/Oracle
- Static typing
- Learning curve is long, and you need to understand concepts thoroughly
- Good for Large Programs and Features Runs on Java Virtual Machine (JVM)
- Works on any Operating System
- Cannot change data types of variables
- Pure Object Oriented Programming (OOP) language
- Easy to get programmers as they are in thousands

Comparison between Python & Java

Key Features of Python:

- Apart from Object Oriented Programming, it supports imperative and functional programming.
- Strongly typed language
- Choice of Startups and freelance web designers and freelance developers
- It uses whitespace to denote the beginning and end of blocks of code.
- Good for smaller programs but can be expanded to large projects
- Less coding required with easy syntax
- Compiles native bytecode
- You can change the data type of variables
- Easier to read and understand with short learning curve Is not supported across a wide variety of platforms
- Finding good Python programmers is not easier as the number is still small

Comparison between Python & Java

- The biggest similarity in these two languages is related to their object-oriented design.
- They are both reputed for cross-platform support though Java is better in this case.
- Java has always had a single large corporate sponsor; either Sun Microsystems or Oracle.
- Python has distributed support and its whitespace set it a little apart from the mainstream than Java.
- Both languages are compiled down to bytecodes that run on VMs, although Python does this automatically at runtime and Java has a separate program for doing this task.
- Neither Java nor Python are suited for high-performance computing.
- Both Java and Python have a vast supply of open-source libraries.

Comparison between Python & Java

Typed vs. dynamically typed:

- The main difference between these two languages is that Java is a statically typed and Python is a dynamically typed.
- Python is strongly but dynamically typed meaning the names in the code are bound to strongly typed objects at runtime.
- The static type inference in Python is a known as a hard problem.
- The downside of Python is of not having type information.
- It is hard to tell what is going on at any given place in the code, particularly when the variable names are ambiguous.
- Java is a statically typed language.
- The names in Java are bound to types at compile time via explicit type declaration.
- You can tell easily, what type of object has an association with which name in Java, which makes analysis task easier for humans as well as compilers.

Comparison between Python & Java

Get the Work done !!

- Python lets that happen: it is a language that gets out of the way and lets you get the job done.
- You will get a feeling that it is a helpful assistant handing you tools.
- When used with appropriate discipline and testing, Python can easily scale to large applications and high-powered Web services.
- Python requires no set-up on most systems.
- A full Python environment is already on every Linux machine, and on Macs and you can immediately start using the language.
- Java, on the other hand, needs a proper set-up with a specific version.
- Java support is also expensive, and configuration on Python is comparatively very easy.

Basic Python

Variables, expressions and statements

Values and types

- A value is one of the basic things a program works with, like a letter or a number.
- The values can be : 1, 2, , 'Hello, World!'.
- These values belong to different types: 2 is an integer, and 'Hello, World!' is a string, so-called because it contains a “string” of letters.
- The print statement also works for integers.

```
print (4)
```

Variables, expressions and statements

Variables

- A variable is a name that refers to a value.
- An assignment statement creates new variables and gives them values:

```
>>> message = 'And now for something completely different'
>>>n=17
>>> pi = 3.14159265359
```

- The above example makes three assignments.
- The first assigns a string to a new variable named message; the second assigns the integer 17 to n; the third assigns the (approximate) value of π to pi.
- We can print the value of a variable :

```
>>> print (n)
17
>>> print (pi)
3.14159265359
```

Variables, expressions and statements

Variable names and Keywords

- Programmers generally choose names for their variables that are meaningful— they document what the variable is used for.
- Variable names can be arbitrarily long.
- They can contain both letters and numbers, but they have to begin with a letter. The underscore character (_) can appear in a name.
- It is often used in names with multiple words, such as my_name or airspeed_of_airplane.
- You should choose mnemonic [memory aid] variable names to help us remember why we created the variable in the first place.
- Keywords are reserved words and CANNOT be used as a variable name.

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

Variables, expressions and statements

Statements

- A statement is a unit of code that the Python interpreter can execute.
- We have seen [so far] two kinds of statements: print and assignment.
- When you type a statement in interactive mode, the interpreter executes it and displays the result, if there is one.
- 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
```

- The assignment statement produces no output.

Variables, expressions and statements

Operators and operands

- Operators are special symbols that represent computations like addition and multiplication.
- The values the operator is applied to are called operands.

- The operators +, -, *, / and ** perform addition, subtraction, multiplication, division and exponentiation, as in the following examples:

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

- Note : When both of the operands are integers, the result is also an integer; If either of the operands is a floating-point number, Python performs floating-point division, and the result is a float

Variables, expressions and statements

Expressions

- An expression is a combination of values, variables, and operators. E.g.

17

X

X + 17

- When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence.
- For mathematical operators, Python follows mathematical convention.
- The acronym PEMDAS is a useful way to remember the rules:
- Parentheses, Exponentiation, Multiplication and Division, Addition and Subtraction
- Operators with the same precedence are evaluated from left to right.
- When in doubt always put parentheses in your expressions to make sure the computations are performed in the order you intend.

Variables, expressions and statements

Modulus Operator

- The modulus operator works on integers and yields the remainder when the first operand is divided by the second.
- In Python, the modulus operator is a percent sign (%). The syntax is the same as for other operators:

```
>>>quotient=7/3
>>> print (quotient)
2
>>> remainder = 7 % 3
>>> print (remainder )
1
```

- So 7 divided by 3 is 2 with 1 left over.

Variables, expressions and statements

String Operations

- The + operator works with strings, but it is not addition in the mathematical sense.
- Instead it performs concatenation, which means joining the strings by linking them end-to-end. For example:

```
>>> first = 10
>>> second = 15
>>> print (first+second )
25
>>> first = '100'
>>> second = '150'
>>> print (first + second )
100150
```

The output of this program is 100150.

Variables, expressions and statements

Comments

- As programs get bigger and more complicated, they get more difficult to read.
- For this reason, it is a good idea to add notes to your programs to explain in natural language what the program is doing.
- These notes are called comments, and they start with the # symbol:

```
# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60
```

- In this case, the comment appears on a line by itself.
- You can also put comments at the end of a line:

```
percentage = (minute * 100) / 60 # percentage of an hour
```

- Everything from the # to the end of the line is ignored—it has no effect on the program.

Variables, expressions and statements

Asking the user for input

- Sometimes we would like to take the value for a variable from the user via their keyboard.
- Python provides a built-in function called *input* that gets input from the keyboard.
- When this function is called, the program stops and waits for the user to type something.
- When the user presses Return or Enter, the program resumes and *input* returns what the user typed as a string.

```
>>> myinput = input()
My name is Prasun [ ← this is what you are typing, say ]

>>> print (myinput)
My name is Prasun
```

Variables, expressions and statements

- Before getting input from the user, it is a good idea to print a prompt telling the user what to input.
- You can pass a string to input to be displayed to the user before pausing for input:

```
>>> name = input('What is your name?\n')
What is your name?
Gautam

>>> print (name )
Gautam
```

- The sequence \n at the end of the prompt represents a newline, which is a special character that causes a line break.
- That's why the user's input appears below the prompt.

Example

- Let’s see a complete Python program :

```
# This program adds two numbers provided by the user
# Store input numbers
num1 = input('Enter first number :  ')
num2 = input('Enter second number :  ')
# Add two numbers
sum = float(num1) + float(num2)
# Display the sum
print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
```

- The output will be :

```
Enter first number : 1.5
Enter second number : 6.3
The sum of 1.5 and 6.3 is 7.8
```

Example

- Let’s see another complete Python program :

```
# Python Program to find the area of triangle
# Three sides of the triangle a, b and c are provided by the user
a = float(input('Enter first side : '))
b = float(input('Enter second side : '))
c = float(input('Enter third side : '))
# calculate the semi-perimeter
s = (a + b + c) / 2
# calculate the area
area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
print('The area of the triangle is %0.2f'  % area)
```

The .format() method

- The .format() method of the str type is an extremely convenient way to format text exactly the way you want it.
- Quite often, we want to embed data values in some explanatory text.
- For example, if we are displaying the number of nematodes in a hectare, it is a lot more meaningful to display it as "There were 37.9 nematodes per hectare" than just "37.9".
- So what we need is a way to mix constant text like "nematodes per hectare" with values from elsewhere in your program.

- Here is the general form: *template*.format(p_0, p_1, ..., k_0=v_0, k_1=v_1, ...)
- The *template* is a string containing a mixture of one or more *format codes* embedded in constant text. The format method uses its arguments to substitute an appropriate value for each format code in the template.

The .format() method

- The arguments to the .format() method are of two types.
- The list starts with zero or more positional arguments p_i, followed by zero or more keyword arguments of the form k_i=v_i, where each k_i is a name with an associated value v_i.

- Just to give you the general flavor of how this works, here's a simple conversational example.
- In this example, the format code "{0}" is replaced by the first positional argument (49), and "{1}" is replaced by the second positional argument, the string ”wheat"

```
>>> "We have {0} hectares planted to {1}.".format(49, ”wheat")
'We have 49 hectares planted to wheat.'
>>>
```

The .format() method

- In the next example, we supply the values using keyword arguments. The arguments may be supplied in any order. The keyword names must be valid Python names.

```
>>> "{monster} has now eaten {city}".format(
 ...   city='Tokyo', monster=’Godzilla')
’Godzilla has now eaten Tokyo'
```

- You may mix references to positional and keyword arguments:

```
>>> "The {structure} sank {0} times in {1} years.".format(
 ...          3, 2, structure='castle')
'The castle sank 3 times in 2 years.'
```

The .format() method

- To do a nice formatting, let's see this example.
- Say we have the following data

```
>>>id = 'IAD'
>>>location = "Dulles Intl Airport
>>>max_temp = 32
>>> min_temp = 13
>>> precipitation = 0.4
```

- And we want the output to look like :

```
IAD : Dulles Intl Airport : 32 / 13 / 0.40
```

- Let's see how we can do that
- Create a template string from the result, replacing all of the data items with {} placeholders.
- Inside each placeholder, put the name of the data item.

```
'{id} : {location} : {max_temp} / {min_temp} / {precipitation}'
```

The .format() method

- To do a nice formatting, let's see this example. [contd.]

- For each data item, append :data type information to the placeholders in the template string.
- The basic data type codes are:
  - s for string
  - d for decimal number
  - f for floating-point number
- [Note : There are other format conversions available]

- It would look like this:

```
'{id:s} : {location:s} : {max_temp:d} / {min_temp:d} / {precipitation:f}'
```

The .format() method

- To do a nice formatting, let's see this example. [contd.]
- Add length information where required.
- Length is not always required, and in some cases, it's not even desirable.
- In this example, though, the length information assures that each message has a consistent format.
- For strings and decimal numbers, prefix the format with the length like this: 19s or 3d.
- For floating-point numbers use a two part prefix like this: 5.2f to specify the total length of five characters with two to the right of the decimal point.
- Here's the whole format:

```
'{id:3d} : {location:19s} : {max_temp:3d} / {min_temp:3d} / {precipitation:5.2f}'
```

The .format() method

- To do a nice formatting, let’s see this example. [contd.]
- Use the format() method of this string to create the final string:

```
>>> '{id:3s} : {location:19s} : {max_temp:3d} / {min_temp:3d} / {precipitation:5.2f}'.format(
... id=id, location=location, max_temp=max_temp,
... min_temp=min_temp, precipitation=precipitation
... )
```

- The ouput that you would get is :

```
'IAD : Dulles Intl Airport : 32 / 13 / 0.40'
```

Strings

Strings

- A string is a sequence of characters.
- You can access the characters one at a time with the bracket operator:

```
>>> fruit = 'banana’
 >>> letter = fruit[1]
```

- The second statement extracts the character at index position 1 from the fruit variable and assigns it to letter variable.
- The expression in brackets is called an index. The index indicates which character in the sequence you want (hence the name).

```
>>> print (letter)
a
```

- <u>Please note that in Python, the index is an offset from the beginning of the string and the offset of the first letter is zero</u>.
- You can use any expression, including variables and operators, as an index, but the value of the index has to be an integer.

Strings

- Getting the length of a string using len
- len is a built-in function that returns the number of characters in a string:

```
>>> fruit = 'banana'
>>> len(fruit)
6
```

- A lot of computations involve processing a string one character at a time.
- Often they start at the beginning, select each character in turn, do something to it, and continue until the end.
- This pattern of processing is called a traversal.
- One way to write a traversal is with a while loop:

```
index = 0
while index < len(fruit):
    letter = fruit[index]
    print (letter)
    index = index + 1
```

Strings

String slices

- A segment of a string is called a slice.
- Selecting a slice is similar to selecting a character:

```
>>> s = 'Monty Python'
>>> print  (s[0:5])
Monty

>>> print (s[6:12] )
Python
```

- The operator [n:m] returns the part of the string from the "n-eth" character to the "m-eth" character, including the first but excluding the last.

Strings

String slices

- If you omit the first index (before the colon), the slice starts at the beginning of the string.
- If you omit the second index, the slice goes to the end of the string:

```
>>> fruit = 'banana'
>>> fruit[ :3 ]
'ban'
>>> fruit[ 3: ]
'ana'
```

- If the first index is greater than or equal to the second the result is an empty string, represented by two quotation marks:

```
>>> fruit = 'banana’
>>> fruit[ 3:3 ]
''
```

- An empty string contains no characters and has length 0, but other than that, it is the same as any other string.

Strings

Strings are Immutable

- It is tempting to use the [] operator on the left side of an assignment, with the intention of changing a character in a string. For example:

```
>>> greeting = 'Hello, world!'
>>> greeting[0] = 'J'
TypeError: object does not support item assignment
```

- The reason for the error is that strings are immutable, which means you can't change an existing string.

Strings

Strings are Immutable

- The best you can do is create a new string that is a variation on the original:

```
>>> greeting = 'Hello, world!'
>>> new_greeting = 'J' + greeting[1:]
>>> print (new_greeting)
Jello, world!
```

- This example concatenates a new first letter onto a slice of greeting.
- Please note : It has no effect on the original string.

- Also note : you can reassign a totally different string to an existing variable, like : greeting = ‘Hello Netaji’

Strings

<u>Strings are Immutable</u>

- Another interesting way to do string slicing is :

```
>>> mystr = “Tip : Rewriting and Immutable String”
```

- Let us slice the above into 2 parts, one part before ‘:’ and the rest

```
>>> colon_pos = mystr.index(‘:’)
>>> text1, post_colon_text = mystr[:colon_position],mystr[colon_position+1:]

>>>text1
‘Tip’
>>>post_colon_text
‘Rewriting and Immutable String’
```

Strings

Strings are Immutable

- The same [as in last slide] can also be done like below :

```
>>> mystr = ‘Tip:Rewriting and Immutable String’
>>> pre_colon_text, _ , post_colon_text = mystr.partition(‘:’)
>>>pre_colon_text
‘Tip’
>>>post_colon_text
‘Rewriting and Immutable String’
```

Note : The partition function returns three things: the part before the target, the target, and the part after the target. We used multiple assignment to assign each object to a different variable. We assigned the target to a variable named _ because we're going to ignore that part of the result. This is a common idiom for places where we must provide a variable, but we don't care about using the object.

Strings

Looping and Counting

- The following program counts the number of times the letter a appears in a string:

```
word = 'banana'
count = 0
for letter in word:
    if letter == 'a':
    count = count + 1
print (count)
```

- This program demonstrates another pattern of computation called a counter.
- The variable count is initialized to 0 and then incremented each time an a is found.
- When the loop exits, count contains the result—the total number of a’s.

Strings

The in Operator

- The word ‘in’ is a boolean operator that takes two strings and returns True if the first appears as a substring in the second:

```
>>> 'a' in 'banana'
True
>>> 'seed' in 'banana'
False
```

String Comparison

- The comparison operators work on strings.
- To see if two strings are equal:

```
if word == 'banana':
    print ('All right, bananas.’)
```

Strings

String Methods

- Strings are an example of Python objects.
- An object contains both data (the actual string itself) as well as methods, which are effectively functions which that are built into the object and are available to any instance of the object.
- Python has a function called 'dir' that lists the methods available for an object.
- The type function shows the type of an object and the dir function shows the available methods.

Strings

String Methods

```
>>> stuff = 'Hello world'
>>> type(stuff)
<type 'str'>

>>> dir(stuff)
['capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs',
'find', 'format', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle',
'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip',
'swapcase', 'title', 'translate', 'upper', 'zfill']
```

Strings

String Methods

- Some examples:

```
>>> word = 'banana'
>>> new_word = word.upper()
>>> print (new_word)
BANANA
```

- This form of dot notation specifies the name of the method, upper, and the name of the string to apply the method to, word.
- The empty parentheses indicate that this method takes no argument.
- A method call is called an invocation; in this case, we would say that we are invoking upper on the word.

```
>>> word = 'banana'
>>> index = word.find('a')
>>> print (index)
1
```

Strings

String Methods [Please see documentation for complete list]

- Some examples:

```
>>> line = 'Please have a nice day'
>>> line.startswith('Please')
True
>>> line.startswith('p')
False
```

```
>>> line = 'Please have a nice day’
 >>> line.startswith('p')
False
>>> line.lower()
'please have a nice day'
>>> line.lower().startswith('p')
True
```

Strings

String Methods [Please see documentation for complete list]

- Some examples:

```
>>> line = 'Please have a nice day'
>>> line.replace(‘ ’, ‘_’)
‘Please_have_a_nice_day’
```

Note : The actual string ‘line’ has NOT Changed!!!!

```
>>>line
'Please have a nice day'
```

If you want the change, you can write:

```
>>> line=line.replace(‘ ’, ‘_’)
>>> line
‘Please_have_a_nice_day’
```

Strings

Parsing Strings

- Often, we want to look into a string and find a substring.
- For example if we were presented a series of lines formatted as follows:

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2014

- And we wanted to pull out only the second half of the address (i.e. uct.ac.za) from each line.
- We can do this by using the find method and string slicing.
- First, we will find the position of the at-sign in the string.
- Then we will find the position of the first space *after* the at-sign.
- And then we will use string slicing to extract the portion of the string which we are looking for.

- [see the next slide]

Strings

Parsing Strings

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print (atpos)
21
>>> sppos = data.find(' ',atpos)
>>> print (sppos)
31
>>> host = data[atpos+1:sppos]
>>> print (host)
uct.ac.za
```

- We use a version of the find method which allows us to specify a position in the string where we want find to start looking.
- When we slice, we extract the characters from "one beyond the at-sign through up to *but not including* the space character".

Strings – Example programs

Some example program:

```
# Program to check if a string  is palindrome or not
# take input from the user
my_str = input("Enter a string: ")

# make it suitable for caseless comparison
my_str = my_str.casefold()

# reverse the string
rev_str = reversed(my_str)

# check if the string is equal to its reverse
if list(my_str) == list(rev_str):
	print("It is palindrome")
else:
	print("It is not palindrome")
```

Strings– Example programs

Another very useful program:

```
# Program to remove all punctuation from the string provided by the user
# define punctuation
punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''

# take input from the user
my_str = input("Enter a string: ")

# remove punctuation from the string
no_punct = ""
for char in my_str:
    if char not in punctuations:
        no_punct = no_punct + char

# display the unpunctuated string
print(no_punct)
```

Strings– Example programs

```
# Purpose: Example: finding a string within a string
s1 = 'spamandeggs'
x = s1.find('and')
print (x)



# Purpose: Example: replacing string by a string


s1 = 'spam and eggs'
s1.replace('and','without')
 spam without egss
print (s1)
spam and eggs
# the above shows that strings are immutable (cannot change)
s2 = s1.replace('and','without')
print (s2)
spam without eggs
```

Strings– Example programs

```
# Palindrome checking – another way
s = input("Please enter a string: ")
z = s[::-1]
if s == z:
        print("The string is a palindrome")
else:
        print("The string is not a palindrome")
```

```
# In this example we will count the number of words in a given line
s = input("Enter a line: ")
print("The number of words in the line are %d"  %  (len(s.split(" "))))
```

Reference

- Python for Informatics – C. Severance
- Think Python – A. B. Downey - [O'Reilly]
- Python Crash Course – Eric Matthes [No starch Press]
- A Byte of Python – Swaroop C H
- Introducing Python – Bill Lubanovic [O'Reilly]

- Learning Python – 5th Ed – Mark Lutz - [O'Reilly]

End of Presentation

Python – S2