Python Identifiers

Rules for writing identifiers:

1. Use only alphanumeric characters and underscores.
2. Never start with a number
3. Use underscores to separate two words
4. Keywords cannot be identifiers

Python Keywords

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

Python Identifiers Test

first_name

while

name1

1name

AGE

_id, _

age$

3

Python Data Types

Integer and Long

Float / Floating Point

Complex Numbers

Boolean

None

Sequence

Sets

Mappings

Python Data Types Test

Write a program to save the following values of a User:

- Name
- Age
- Year of birth
- Is New User?
- List of his/her favorite food

Input and Output

`input()`

`print()`

Python User input Test

Write a Program to take marks of a User in English, Science and Maths and print the average of these marks.

Comments in Python

Single line comments start with **hash symbol (#)**

Multi - line comments start and end with **triple quoted strings (’’’)**

Python Operators

Arithmetic Operators → `+ , - , / , * , % , // , **`

Assignment Operators → `+= , -= , *= , /= , %= , //= , **=`

Relational / Comparison Operators → `==, != , < , > , <= , >=`

Logical Operators → `and , not , or`

Truth Tables

NOT

x	x'
0	1
1	0

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

Python Arithmetic Operators Test

Write a Program to take the average of three numbers.

if

```
if test expression:
    statement(s)
```

In Python, the body of the `if` statement is indicated by the indentation. Body starts with an indentation and the first unindented line marks the end.

The body of `if` is executed only if test expression evaluates to `True`.

if...else

```
if test expression:
    Body of if
else:
    Body of else
```

The if..else statement evaluates test expression and will execute body of if only when test condition is True.

If the condition is False, body of else is executed. Indentation is used to separate the blocks.

if...elif...else

```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

Python nested if statements

We can have a `if...elif...else` statement inside another `if...elif...else` statement. This is called nesting in computer programming.

Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. This can get confusing, so must be avoided if we can.

Python range() function

We can generate a sequence of numbers using `range()` function. `range(10)` will generate numbers from 0 to 9 (10 numbers). → `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`

We can also define the start, stop and step size as `range(start,stop,step size)`. step size defaults to 1 if not provided.

Python for loop

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called **traversal**.

```
for val in sequence:

      Body of for
```

Python while loop

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

We generally use this loop when we don't know beforehand, the number of times to iterate.

```
while test_expression:
    Body of while
```

break & continue

The `break` statement terminates the loop containing it.

The `continue` statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

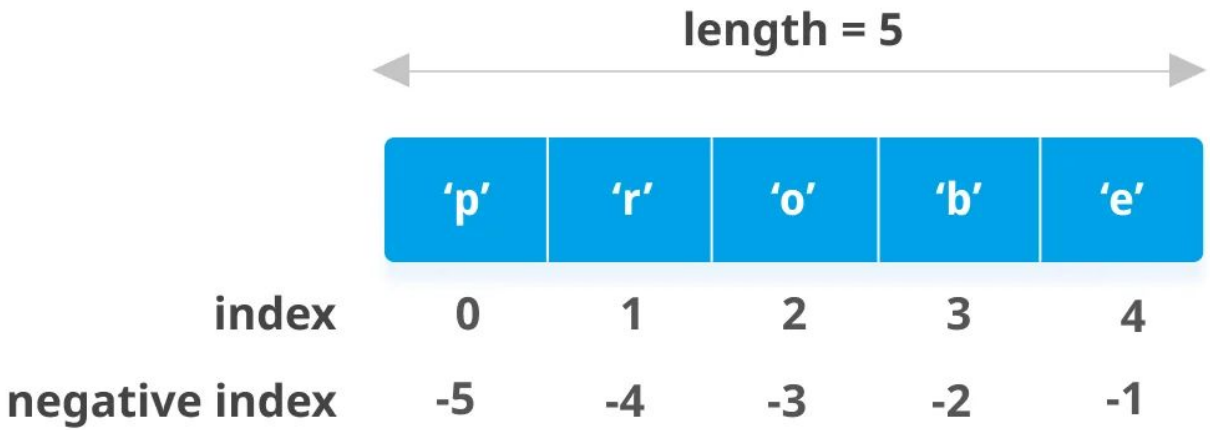
Python Strings

A string is a sequence of characters

Strings can be created by enclosing characters inside a single quote or double quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

Strings are immutable. This means that elements of a string cannot be changed once it has been assigned. We can simply re-assign different strings to the same name.

Python Strings indexing



Accessing characters in a String (slicing)

We can access individual characters using indexing and a range of characters using slicing. Index starts from 0. Trying to access a character out of index range will raise an `IndexError`. The index must be an integer.

Python allows negative indexing for its sequences.

The index of -1 refers to the last item, -2 to the second last item and so on. We can access a range of items in a string by using the slicing operator (colon).

Using + and * operators with Strings

Joining of two or more strings into a single one is called concatenation.

The + operator does this in Python. Simply writing two string literals together also concatenates them.

The * operator can be used to repeat the string for a given number of times.

for loops with Strings

```
for letter in 'Hello':
    print(letter*2, end = " ")
```

Prints:

HH ee ll ll oo

Python String methods

- `isalpha()`
- `isdigit()`
- `islower()`
- `lower()`
- `upper()`
- `isupper()`
- `lstrip()`
- `rstrip()`

Python Lists

In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.

It can have any number of items and they may be of different types (integer, float, string etc.).

```
1.  my_list = []
2.  my_list = [1, 2, 3]
3.  my_list = [1, "Hello", 3.4]
```

Lists comprehension

List comprehension is an elegant and concise way to create a new list from an existing list in Python.

List comprehension consists of an expression followed by for statement inside square brackets.

new_list = [expression for item in list if condition]

```
pow2 = [2 ** x for x in range(10)]

# Output: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
print(pow2)
```

Lists methods

- `append`
- `insert`
- `sort`
- `pop`
- `clear`
- `reverse`
- `index`
- `count`

Lists functions

- `len(list)`
- `max(list)`
- `min(list)`
- `list(seq)`
- `sum(list)`

for loops with Lists

```
for car in ['swift', 'ritz', 'ferrari']:
    print(car)
```

Prints:

swift

ritz

ferrari

Tuples

A tuple is created by placing all the items (elements) inside parentheses (), separated by commas. The parentheses are optional, however, it is a good practice to use them.

Tuples are immutable. This means that elements of a tuple cannot be changed once it has been assigned.

```
1.  my_tuple = ()
2.  my_tuple = (1, 2, 3)
3.  my_tuple = (1, "Hello", 3.4)
```

Tuple operations

Accessing element from a tuple

1. Positive and negative indexing → like **Lists**
2. Slicing → like **Lists**
3. Iterating through a tuple → like **Lists**

Tuple functions

- `len(tuple)`
- `max(tuple)`
- `min(tuple)`
- `tuple(seq)`
- `sum(tuple)`

Dictionary

Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair.

```
1.  my_dict = {}
2.  my_dict = {1: 'apple', 2: 'ball'}
3.  my_dict = {'name': 'John', 1: [2, 4, 3]}
4.  my_dict = dict({1:'apple', 2:'ball'})
```

Iterating through a Dictionary

Using a `for` loop we can iterate through each key in a dictionary.

```
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

for i in squares:

    print(squares[i])
```

Updating Dictionary items

You can also update a dictionary by modifying existing key-value pair or by merging another dictionary to an existing one.

We can add new items or change the value of existing items using assignment operator.

If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.

```
my_dict = {'name':'Jack', 'age': 26}
```

```
my_dict['age'] = 27
```

Python functions

In Python, function is a group of related statements that perform a specific task.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Categories of functions

1. Built-in
2. Modules
3. User-defined

Built-in functions

`int()` / `float()` / `eval()`

`input()`

`min()` / `max()`

`abs()`

`type()`

`len()`

`round()`

`range()`

Modules

A module is a file containing functions and variables defined in separate files. A module is simply a file that contains Python code. When we break a program into modules, each module should contain functions that perform related tasks.

Importing from modules

- We can import the definitions inside a module to another module or the interactive interpreter in Python.

  We use the `import` keyword to do this.

  ```
  import math
  ```

- We can rename modules using `as`
- We can import specific names `from` a module without importing the module as a whole using `from`.

Important modules

math

random

string

User-defined functions

Syntax of Function

```
def function_name(parameters):
    """docstring"""
    statement(s)
```

Calling the functions

Once we have defined a function, we can call it from another function, program or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

```
>>> greet('Anuj')

  Hello, Anuj. Good morning!
```

Returning from the functions

The `return` statement is used to exit a function and go back to the place from where it was called.

```
return [expression_list]
```

Python Functions Test

Write a function which takes a list of names as argument and greets “Hello” to everyone

What is a file?

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).

in Python, a file operation takes place in the following order.

1. Open a file
2. Read or write (perform operation)
3. Close the file

How to Open a file?

Python has a built-in function `open()` to open a file. This function returns a file object.

We can specify the mode while opening a file. In mode, we specify whether we want to read `'r'`, write `'w'` or append `'a'` to the file. We also specify if we want to open the file in text mode or binary mode.

We can also use with statement instead of open() and close():

```
with open('filename.txt', 'w') as fileObject:
```

Modes while opening a file

Python File Modes

Mode	Description
'r'	Open a file for reading. (default)
'w'	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode.
'+'	Open a file for updating (reading and writing)

Reading from a file

Open the file using ‘r’ mode. We get a fileObject that can be used to read from the file.

We can read a file line-by-line using a for loop. This is both efficient and fast.

```
>>> for line in f:
...     print(line, end = '')
```

Alternately, we can use readline() method to read individual lines of a file. This method reads a file till the newline, including the newline character.

Writing to a file

Open the file using ‘w’ mode. We get a fileObject that can be used to write strings into the file.

```
fileObject.write(string)
```

```
fileObject.writelines(sequence)
```