Python : All Syntax & Short Note

Variable & Data Types

Variables:

Variable is a name given to a memory location in a program. [**name** = "Tamim", **age** = 22]

Rules for Variables or Identifiers:

1. Identifiers can be combination of uppercase and lowercase letters, digits or an underscore (_). [**variable**, **variable1, variable_1**]
2. An Identifier can not start with digit. So while **variable1** is valid, **1variable** is not valid.
3. We can't use special symbols like !, #, @, %, $ etc in our identifiers.
4. Identifier can be of any length

Data Types:

- Integers
- String
- Float
- Boolean
- None

Keywords:

To be added later . . .

Comments:

- Single Line

```
# Single Line Comment
```

- Multiple Line

```
"""
Multi Line
Comment
"""
```

Types of Operator:

An operator is a symbol that performs a certain operation between operands

- Arithmetic Operators [+, -, *, /, %, **]
- Relational / Comparison Operators [==, ≠, >, <, ≥, ≤]
- Assignment Operators [=, +=, -=, *=, /=, %=, **=]
- Logical Operator [not, and, or]

Print → print()

Type Conversion

conversion → automatically

```
a = 2
b = 3.25
c = "2"
sum = a + b
print(sum)               #output: 5.25
print(a+c)               #error: unsupported operand type for +: 'int' and 'str'
```

Type Casting

casting → manually

```
a = 2
c = "2"
c = int(c)
print(a+c)               #output: 4
```

Input

input() statement is used to accept values (using keyboard) from user

input() always take as string, so always use data type

```
name = input("Enter your name: ")
age = int(input("Enter your age: "))
```

Strings & Conditional Statements

Strings [Immutable]

String is data type that stores a sequence of characters

String writing

- "This is a string"
- 'String in single quote'
- """This is also a string"""

String Concatenation → str_1 + str_2

String Length → len()

Indexing

Indexing starts from 0 & Can't be modified

```
idx = "Bangladesh"
print(idx[0])                              #output : B
#idx[1] = "A"                              # Not allowed
```

slicing → [ending idx is not included]

string_name[starting_idx : ending_idx]   **str[1 : 4], str[-4 : -1]**

String Functions

Original variable remain unchanged

```
str = "I am a Coder"
```

str.**endswith(** "er" **)**	returns true if strings ends with substr(**er**)
str.**capitalize()**	capitalize 1st char
str.**replace(** "o", "a" **)**	replaces all occurrence of **o** with **a**
str.**find(** "am" **)**	returns 1st index of 1st occurrence
str **.find(** "b" **)**	if -1, means b doesn't exist
str.**count(** "a" **)**	counts the occurrence of substr(**a**)

Conditional Statement

if-elif-else syntax

```
if(condition):
   statement1
elif(condition):
   statement2
else:
   statement3
```

Lists & Tuples

List [Mutable]

A built-in data type that stores set of values

It can store elements of different types [**integer**, **float**, **string** etc]

list can store element of different types

```
student = ["Arjun", 94.5, 22, "Dhaka"]
student[0] = "Akash"                            #change value akash from arjun
```

list slicing → [similar to string]

list_name[starting_idx : ending_idx]

List Methods

```
num = [3, 1, 2]
```

num.**append(** el **)**	adds element (**4**) in the end
num.**sort()**	sorts in ascending order
num.**sort(** reverse=True **)**	sorts in descending order
num.**reverse()**	reverse list
num.**insert(** idx, el **)**	insert element at index
num.**remove(** el **)**	remove 1st occurrence of element
num.**pop(** idx **)**	remove element at index

num.copy()	copy list

Tuple [Immutable]

A build-in data type that lets us create **immutable** sequence of values

Tuple →

```
num = (3, 1, 2)
print(num[1])                #output: 1
#num[1] = 4                   #error: tuple object doesn't support item assignment
```

tuple slicing → [same as string & list]

tuple_name[starting_idx : ending_idx]

Tuple Methods

```
num = (2, 1, 3, 1)
```

num.index(el)	returns index of first occurrence
num.count(el)	counts total occurrences

Dictionary & Set

Dictionary

Dictionaries are used to store data values in **key:value** pairs

They are **unordered**, **mutable**(changeable) & **don't allow duplicate keys**

```
info = {
    "name" : "Tamim",                      #string
    "age" : 22,                            #int
    "marks" : [94.4, 89.7],                #list
    "subjects" : ("python", "c"),          #tuple
}
```

Change value → info["name"] = "Iqbal"

Add new value → info["country"] = "Bangladesh"

Empty dictionary:

```
hello = {}
```

Nested dictionary:

```
student = {
    "name" : "Saimon",
    "marks" : {
        "phy" : 99,
        "chem" : 89
    }
}
print(student)     #output: {'name': 'Saimon', 'marks': {'phy': 99, 'chem': 89}}
print(student["marks"])                    #output: {'phy': 99, 'chem': 89}
print(student["marks"]["phy"])             #output: 99
```

Dictionary Methods

```
info = {
    "name" : "Tamim",                     #string
    "age" : 22,                           #int
    "marks" : [94.4, 89.7],               #list
    "subjects" : ("python", "c"),         #tuple
}
```

info.**keys()**	returns all keys
info.**values()**	returns all values
info.**items()**	returns all (key, val) pairs as tuple
info.**get(** "key" **)**	returns the value according to key
info.**update()**	insert the specified items to the dictionary

Set

Set is the collection of the **unordered items**

Each element in the set must be **unique** & **immutable**

List & dictionary can't be stored in set because they are mutable

Set → Mutable,   Element of set → Immutable

```
collection = {1, 2, 3, 4, "hello", 2}
print(collection)     #output: {1, 2, 3, 4, 'hello'} : Duplicate 2 doesn't print
```

Empty Set :

```
num = set()
```

Set Methods

```
collection = {1, 2, 3, 4, "hello", 2}
```

collection.**add(** el **)**	adds an element
collection.**remove(** el **)**	removes the element
collection.**clear()**	empties the set
collection.**pop()**	removes a random value

```
set = {1, 3, 5}
```

collection.**union(** set **)**	combines both set values & returns new
collection.**intersection(** set **)**	combines common values & returns new

Loops

Loops are used to repeat instruction

While Loop

while condition:

  code block to be executed

Break

Used to terminate the loop when encountered

```
i = 1
while i <= 10:
    print(i)
    if(i == 5):
        break
    i += 1                           # returns 1 to 5
```

Continue

Terminates execution in the current iteration & continues execution of the loop with the next iteration

```
i = 1
while i <= 10:
    if(i == 5):
        i += 1
        continue
    print(i)
    i += 1                           # returns 1 to 10 without 5
```

For Loop

For loops are used for sequential traversal. For traversal **list**, **string**, **tuples**

for var in sequence:

  code block to be executed

For - Else

Applied when break is used

```
string = "Bangladesh"
for char in string:
    if(char == 'x'):
        print('x is found')
        break
    print(char)
else:
    print("Not Found")
```

Range

Range functions returns a sequence of number, string from 0 by default, and increment by 1 (by default), and stops before a specified number

range(start, stop, step **)**

```
for i in range(5):
    print(i)                          #returns 0 to 4

for i in range(1, 5):
    print(i)                          #returns 1 to 4

for i in range (1, 5, 2):
    print(i)                          #returns 1 to 3

for i in range (5, 0, -1):
    print(i)                          #returns 5 to 1
```

Pass

Pass is a null statement that does nothing. It is used as a placeholder for future code.