Agenda

- A) Python - Keywords
- B) Python - Identifiers
- C) Python - I/O statements

A) Python Keywords

- Keywords are the reserved words in Python.
- We cannot use a keyword as
  - a variable name,
  - function name,
  - any other identifier.
- They are used to define the syntax and structure of the Python language.
- In Python, keywords are case sensitive.
- All the keywords except True, False and None are in lowercase and they must be written as they are.

Keyword	Description
and	A logical operator
as	To create an alias
assert	For debugging
break	To break out of a loop

class	To define a class
continue	To continue to the next iteration of a loop
def	To define a function
del	To delete an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements
except	Used with exceptions, what to do when an exception occurs
False	Boolean value, result of comparison operations
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
for	To create a for loop
from	To import specific parts of a module
global	To declare a global variable
if	To make a conditional statement
import	To import a module
in	To check if a value is present in a list, tuple, etc.
is	To test if two variables are equal
lambda	To create an anonymous function
None	Represents a null value

nonlocal	To declare a non-local variable
not	A logical operator
or	A logical operator
pass	A null statement, a statement that will do nothing
raise	To raise an exception
return	To exit a function and return a value
True	Boolean value, result of comparison operations
try	To make a try...except statement
while	To create a while loop
with	Used to simplify exception handling
yield	To end a function, returns a generator

B) Python Identifiers

- An identifier is a name given to entities like class, functions, variables, etc.
- It helps to differentiate one entity from another.

Rules for writing identifiers

1. Identifiers can be a combination of letters in lowercase **(a to z)** or uppercase **(A to Z)** or digits **(0 to 9)** or an underscore _. Names like myClass, var_1 and print_this_to_screen, all are valid example.
2. An identifier cannot start with a digit. 1variable is invalid, but variable1 is a valid name.
3. Keywords cannot be used as identifiers.

Example:

```
else = 1

Output

 File "<interactive input>", line 1

   else = 1

        ^

SyntaxError: invalid syntax
```

4. We cannot use special symbols like **!**, **@**, **#**, **$**, **%** etc. in our identifier.

```
a@ = 0
```

Output

```
  File "<interactive input>", line 1
    a@ = 0
     ^
SyntaxError: invalid syntax
```

Things to Remember

- Python is a case-sensitive language.
- This means, Variable and variable are not the same.
- Always give the identifiers a name that makes sense.
    - While `c = 10` is a valid name, writing `count = 10` would make more sense,
    - and it would be easier to figure out what it represents when you look at your code after a long gap.
- Multiple words can be separated using an underscore, like `this_is_a_long_variable`.

C) Python I/O Statements

- We use the print() function to output data to the standard output device (screen)
- An example of its use is given below.

```
print('This sentence is output to the screen')
```

Output

```
This sentence is output to the screen
```

- Another example is given below:

```
a = 5

print('The value of a is', a)
```

Output

```
The value of a is 5
```

- The actual syntax of the print() function is:
  - print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
  - Here, objects is the value(s) to be printed.
  - The sep separator is used between the values. It defaults into a space character.
  - After all values are printed, end is printed. It defaults into a new line.
  - The file is the object where the values are printed and its default value is sys.stdout (screen).

- Here is an example to illustrate this.
    - `print(1, 2, 3, 4)`
    - `print(1, 2, 3, 4, sep='*')`
    - `print(1, 2, 3, 4, sep='#', end='&')`
    - **Output**
    - `1 2 3 4`
    - `1*2*3*4`
    - `1#2#3#4&`

Output formatting

- Sometimes we would like to format our output to make it look attractive.
- This can be done by using the `str.format()` method.
- This method is visible to any string object.
    - `x = 5; y = 10`
    - `print('The value of x is {} and y is {}'.format(x,y))`
    - `The value of x is 5 and y is 10`
- Here, the curly braces `{}` are used as placeholders.
- We can specify the order in which they are printed by using numbers (tuple index).
    - `print('I love {0} and {1}'.format('bread','butter'))`
    - `print('I love {1} and {0}'.format('bread','butter'))`
    - **Output**
    - `I love bread and butter`
    - `I love butter and bread`

- We can even use keyword arguments to format the string.
    - `print('Hello {name}, {greeting}'.format(greeting = 'Goodmorning', name = 'John'))`
    - `Hello John, Goodmorning`

Python Input

- Up until now, our programs were static.
- The value of variables was defined or hard coded into the source code.
- To allow flexibility, we might want to take the input from the user.
- In Python, we have the `input()` function to allow this.
- The syntax for `input()` is:

```
input([prompt])
```

    - where `prompt` is the string we wish to display on the screen.
    - It is optional.
- Example:
    - `num = input('Enter a number: ')`
    - `Enter a number: 10`
    - `print(num)`
    - `Output: 10`
- Here, we can see that the entered value `10` is a string, not a number.
- To convert this into a number we have to typecast by using `int()` or `float()` functions.

Typecasting

1. **Typecasting the input to Integer:** There might be conditions when you might require integer input from user/Console, the following code takes two input(integer/float) from console and typecasts them to integer then prints the sum.

```
# input
num1 = int(input())
num2 = int(input())

# printing the sum in integer
print(num1 + num2)
```

2. **Typecasting the input to Float:** To convert the input to float the following code will work out.

```
# input
num1 = float(input())
num2 = float(input())

# printing the sum in float
print(num1 + num2)
```

3. **Typecasting the input to String:** All kind of input can be converted to string type whether they are float or integer. We make use of keyword str for typecasting.

```
# input
string = str(input())

# output
print(string)
```

Example code – without type casting:

```
a = input("Enter the value for a: ")
print(a)
b = input("Enter the value for b: ")
print(b)

if a > b:
 print("a is greater than b.")
 print("I am in a block.")
 print("a block is bigger than b block.")

else:
 print("b is greater than a.")
 print("I am in b block.")
 print("b block is bigger than a block.")
```

Example code – with type casting:

```
a = int(input("Enter the value for a: "))
print(a)
b = int(input("Enter the value for b: "))
print(b)

if a > b:
 print("a is greater than b.")
 print("I am in a block.")
 print("a block is bigger than b block.")

else:
 print("b is greater than a.")
 print("I am in b block.")
     print("b block is bigger than a block.")
```