Python : * Inventor : Guido van Rossum

* Year and place of invention : 1989, Centrum Wiskunde and Informatica (CWI), Amsterdam, Netherlands

* Rules for naming an identifier :
  - → An identifier must start with letter or underscore.
  - → After the first character, they can include letters, digits and underscores.
  - → Python is case-sensitive.

* Types of comments :
  - * Single-line comments starts with a hash (#) and end at the end of the line.
  - * Multi-line comments starts with three double quotes (""") and end with three double quotes (""").

* Features :
  - → Python is Interpreted language.
  - → Python is dynamically typed language.
  - → Python is general purpose language, so it can be used to write variety of programs.
  - → Python is powerful and expressive language, and used often in data science, machine learning and web development.

Error in Python

① **Syntax** :- When the proper syntax of lang. is not followed

- When the interpreter is unable to parse the code due to violation of rules of python prog. lang.

Ex:-
```
x = 0
if (x == 2)        <-- ! error.
    print ("Hi")
```

② **Runtime Error** :- The error's that are encountered during the execution of program & doesn't allow the prog. to continue the execution of prog.

Ex:- Divide by zero, index out of bound (array size is 5) arr[6], error accessing undefined func. & variables.

③ **Semantic Error (Logical)** :- When the code runs without any syntax or runtime error, but produces incorrect result due to flawed logic in code.

Ex:-
```
def add():
    a = 2
    b = 3
    return a*b;
print(add());
```

output = 6

Expected = 5

Variables :-

① Can be defined with assigning any datatype

② Can also assign datatype using casting

Ex :- x = str(3)
y = int(3)

③ To get the type of variable we use type() func.

Ex :- x = 5
print (type(x)).

④ Variables are case sensitive.

Keywords :- • Predefined words in python that has some special meaning & purpose to that lang.

• Cant be used as variables with same nam

Python Type Conversion :-

* Implicit type conversion :- Done by language itself during the execution.

Ex. ⇒

```
x = 10
print (type(x))
x = 15.6
print (type(x))
```

Output :-
<class 'int'>
<class 'float'>

* Explicit :- Done by the user.

~~(1) Converting~~

Ex:-

```
s = "10010"
x = int(s, 2)
print(x)
y = float(s)
print(y)
z = int(s)
print(z)
```

output :-

```
18
10010.0
10010
```

* ascii value of character

Ex:-

```
s = '4'
print(ord(s))
```

output

```
52
```

* int to hex or oct

Ex:-

```
print(hex(56))
print(oct(56))
```

output

```
0x38
0o70
```

* list(), tuple(), set()

```
s = "abcddea"
x = tuple(s)
y = set(s)
z = list(s)
print(x)
print(y)
print(z)
```

output :-

```
('a', 'b', 'c', 'd', 'd', 'e', 'a')
{'a', 'b', 'c', 'd', 'e'}
['a', 'b', 'c', 'd', 'd', 'e', 'a']
```

Name : Lavnish Kumar Branch : CSE Sem : 5th

Roll No. : 12111018

Assignment #2

Operators are symbols that perform operations on one or more operands (variables or values), there are different types of operators used in Python:

1) Arithmetic Operators:

'+' (Addition) : Adds two operands

'-' (Subtraction) : Subtracts the right operand from left operand.

'*' (Multiplication) : Multiplies two operands.

'/' (Division) : Divides the left operand by the right operand (results in a float).

'//' (Floor Division) : Divides the left operand by the right operand and returns the floor value (results in an integer).

'%' (Modulus) : Returns the remainder of the division between left operand and right operand.

'**' (Exponentation) : Raises the left operand to the power of right operand.

2) Assignment Operators :

'=' (Assignment) : Assigns the value of the right operand to the left operand.

'+=', '-+', '*=', '/=', '//=', '%=' : Perform the operations on the left operand with the right operand and assign the result back to the left operand.

3) Comparison Operators :

'==' (Equal) : Check if two operands are equal.

'!=' (Not Equal) : Check if two operands are not equal.

'<' (Less Than) : Checks if the left operand is less than the right operand.

'>' (Greater Than) : Checks if the left operand is greater than the right operand.

'<=' (Less Than or Equal) : Check if the left operand is less than or equal to the right operand.

'>=' (Greater Than or Equal) : Check if the left operand is greater than or equal to right operand.

4) Logical Operators :

'and' : Logical AND operator, returns TRUE if both operands are TRUE.

'or' : Logical OR operator, returns TRUE if atleast one operand is TRUE.

'not' : Logical NOT operator, returns opposite of operand's truth value.

5) Bitwise Operators :

'&' (Bitwise AND) : Performs Bitwise AND operation b/w operands.

'|' (Bitwise OR) : Performs Bitwise OR operation b/w operands.

'^' (Bitwise XOR) : Performs Bitwise exclusive OR operation b/w operands.

'~' (Bitwise NOT) : Flips the bits of the operand, changing 0s to 1s and vice versa.

'<<' (Left Shift) : Shifts the bits of the left operand to the left by the number of positions specified by right operand.

'>>' (Right Shift) : Shifts the bits of the left operand to the right by the number of positions specified by right operand.

6) Membership Operators :

'in' : Returns TRUE if a value is found in a sequence (eg a list, tuple or string)

'not in' : Returns TRUE if value is not found in a sequence.

7) Identity Operators :

'is' : Returns TRUE if two operands refer to same object in memory.

'is not' : Returns TRUE if two operands do not refer to same object in memory.

8) Unary Operators :

'-' (Negation) : Negates the values of the operand.

'+' (Unary Plus) : Represents the value of the operand (not commonly used)

* Interactive Mode :- When we directly type your code into the python interpreter directly. (Python IDLE)
- Here we write single line of code & press enter to get instant output.
- Useful for small programs, or for testing things out.

* Script Mode :- Script mode is where you write your code in .py file and then run it with python commands.
- Used to write large programes & it is the most common way to write.

Operator Precedence in descending order (highest to low)

Operators	Meaning
()	Parenthesis (Highest precedence)
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor Division, Modulus
+, -	Addition, Subtraction
<< , >>	Bitwise Shift Operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in	Comparison, Identity, Membership operators
NOT	Logical Not
AND	Logical And
OR	Logical OR (Least precedence)

Name: Lavnish Kumar Roll No.: 12111018 CSE (5th Sem)

Assignment #3

Different types of jump statements in python are:

1) Break Statement:

The break statement is used to exit a loop prematurely, before its normal completion. It is commonly used in loops like 'for' and 'while' to terminate the loop's execution once a specific condition is met. When a 'break' statement is encountered, the program jumps out of the loop, and control moves to the statement immediately following the loop.

eg

```
for i in range (5):
    if i == 3:
        break
    print (i)
```

2) Continue Statement:

The 'continue' statement is used to skip the remaining part of the current iteration of a loop and move on to the next iteration of the loop. It doesn't exit the loop entirely, rather, it jumps to the next iteration. This can be useful when you want to skip certain values or conditions in the loop.

eg.
```
for i in range (5):
    if (i == 2)
        continue
    print (i)
```

3) Pass statement :

The 'pass' statement is used as placeholder in situations where code is syntactically required but no action is intended or necessary. It is often used as a placeholder for future code or in situations where you don't want to execute any specific action, but you need a valid code block to maintain the program's structure.

eg
```
def placeholder_function ():
    pass    # No implementation yet

class EmptyClass :
    pass    # No attributes or methods defined yet
```

Name: Lavnish Kumar Roll No.: 12111018 CSE (5th Sem)

Assignment #4

Python functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again. The benefits of using functions is to increase code readability and reusability

Different types of user-defined functions:

① No argument, no return type:

```
def add():
    a = int(input())
    b = int(input())
    print(a+b)
```

② No argument, but return type

```
def add():
    a = int(input())
    b = int(input())
    return a+b
x = add()
print(x)
```

③ Arguments but no return type :

```
def add (a, b) :
    print (a+b)
add (1,2)
```

④ Arguments and return type

```
def add (a,b) :
    return a+b
x = add (2,3)
print (x)
```

Function Composition in Python

Prerequisite: reduce(), lambda

Function composition is the way of combining two or more functions in such a way that the output of one function becomes the input of the second function and so on. For example, let there be two functions “F” and “G” and their composition can be represented as F(G(x)) where “x” is the argument and output of G(x) function will become the input of F() function.

Example:

```
# Function to add 2
# to a number
def add(x):
    return x + 2

# Function to multiply
# 2 to a number
def multiply(x):
    return x * 2

# Printing the result of
# composition of add and
# multiply to add 2 to a number
# and then multiply by 2
print("Adding 2 to 5 and multiplying the result with 2: ",
      multiply(add(5)))
```

Output:

```
Adding 2 to 5 and multiplying the result with 2: 14
```

Harshal Rajput
12111010
CSE 5th Sem

Question - (5)

What is String? Explain any 5 string library function used in python with example

String :-

* In Python, string is a sequence of characters enclosed in either single ('') or double quotes (" ").

* Strings are a fundamental data type used to represent text or sequences of characters.

* 5 commonly used String Library Functions :-

(1) len() :-

The len() function returns the length (number of characters) of a string.

Eg:

```
text = "Hello World"
length = len(text)
print(length)      # Output: 11
```

(2) str.upper() :- and str.lower() :-

These methods are used to convert a string to uppercase or lowercase respectively.

Eg:

```
text = "Hello World"
uppercase_text = text.upper()
lowercase_text = text.lower()
print(uppercase_txt)    # Output: HELLO WORLD
print(lowercase_txt)    # Output: hello world
```

(3) str.strip() :-

The strip() method removes leading and trailing whitespace (spaces, tabs and newline characters) from a string.

Eg :

```
text = "   Hello, world   "
new_text = text.strip()
print(new_text)        # Output : Hello, world
```

(4) str.split() :-

The split() method splits a string into a list of substrings based on a specific delimiter. By default, it splits on spaces.

Eg :

```
text = "Hello, world!"
words = text.split()
print(words)        # Output : ["Hello," "world"]
```

(5) str.replace() :-

The replace() method replaces all occurences of a specific string with another substring.

Eg :

```
text = "Hello, world"
new_text = text.replace("world", "Python")
print(new_text)        # Output : "Hello, Python"
```