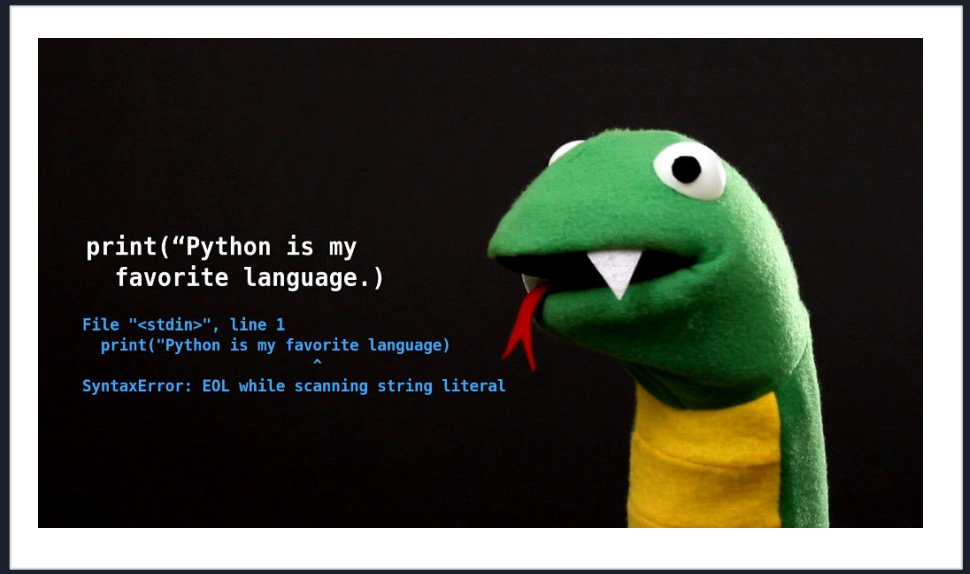
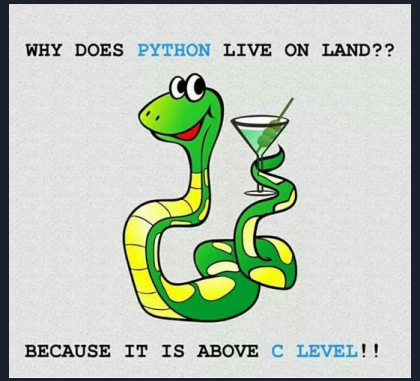
PYTHON

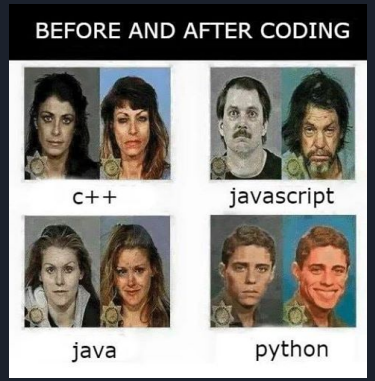
(A High level Programming language)

- Meenu Sowjanya . C .



FUNNY MEMES ON PYTHON





PYTHON FACTS !!!

- Python was developed in the year 1991 , by Guido Van Rossum .

- This language was named so , because the developer of this language loved the PYTHON FLY CIRCUS held at his state . So he named the language as PYTHON .

- The logo of this language is designed so , because Python generally refers to a creature similar to a dragon snake .

So, it is said that this logo has 2 snakes , left and right with eyes at the edge.

Also , if you look into the logo deeply , the blue snake at the left slightly looks like P and the next looks like Y which represents the extension of a python file .py

Why Python ?

- **Easy to Read, Learn and Write .....** Python is a high-level programming language that has English-like syntax. This makes it easier to read and understand the code.

- **Improved Productivity .....** Python is a very productive language. Due to the simplicity of Python, developers can focus on solving the problem. They don't need to spend too much time in understanding the syntax or behavior of the programming language. You write less code and get more things done.

- **Interpreted Language .....** Python is an interpreted language which means that Python directly executes the code line by line. In case of any error, it stops further execution and reports back the error which has occurred.Python shows only one error even if the program has multiple errors. This makes debugging easier.

- **Dynamically Typed .....** Python doesn't know the type of variable until we run the code. It automatically assigns the data type during execution. The programmer doesn't need to worry about declaring variables and their data types.

Why Python ?

- **Free and Open Source .....** Python comes under the OSI approved open-source license. This makes it free to use and distribute. You can download the source code, modify it and even distribute your version of Python. This is useful for organizations that want to modify some specific behavior and use their version for development.

- **Vast Libraries Support .....** The standard library of Python is huge, you can find almost all the functions needed for your task. So, you don't have to depend on external libraries.

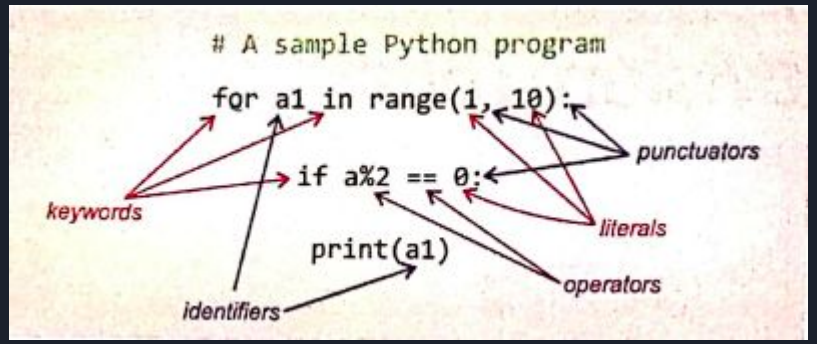
- **Portability .....** In many languages like C/C++, you need to change your code to run the program on different platforms. That is not the same with Python. You only write once and run it anywhere.

LET'S GET INTO PYTHON PROGRAMMING..

WHAT ALL A PYTHON PROGRAM CONTAINS ?

TOKENS :

A smallest individual unit of a python program is called a token.



TOKENS :

Python has five types of tokens :

1. Keywords
2. Identifiers (Names)
3. Literals
4. Operators
5. Punctuators

1 . KEYWORDS :

Keywords are predefined words with the special meaning to the language compiler or interpreter.

These are reserved for special purpose and must not be used as normal identifiers.

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

Keywords in Python

2 . IDENTIFIERS

Identifiers are the names , given to different parts of the program like Variables, Objects, Classes, Functions, Lists, Dictionaries and so on.

Naming Rules of Python Identifiers :

- Variable names must only be a *non-keyword word* with no spaces in between.
- Variable names must be made up of only letters, numbers, and underscore (_).
- Variable names cannot begin with a number, although they can contain numbers.

The following are some *valid* identifiers :

Myfile	DATE9_7_77
MYFILE	_DS
_CHK	FILE13
Z2T0Z9	_HJI3_JK

The following are some *invalid* identifiers :

DATA-REC	contains special character - (hyphen) (other than A - Z, a - z and _ (underscore))
29CLCT	Starting with a digit
break	reserved keyword
My.file	contains special character dot (.)

Note : PYTHON is CASE SENSITIVE , as it treats Uppercase and Lowercase letters differently

3 . LITERALS

Literals are the data items that has a constant / fixed value .

Python allows 4 main literals :

1. **String Literals**
2. **Numeric Literals**
3. **Boolean Literals**
4. **Special Literal NONE**

Examples of Literals :

(i) String Literals

A *string literal* is a sequence of characters surrounded by quotes (single or double or triple quotes). String literals can either be ***single line strings*** or ***multi-line strings***.

- **Single line strings** must terminate in one line *i.e.*, the closing quotes should be on the same line as that of the opening quotes. (See below)
- **Multiline strings** are strings spread across multiple lines. With single and double quotes, each line other that the concluding line has an end character as \ (backslash) but with triple quotes, no backslash is needed at the end of intermediate lines. (see below) :

```
>>> Text1 = "Hello World"      ← Single line string

>>> Text2 = "Hello\            ← Multi-line string
    World "

Text3 = '''Hello               ← No backslash needed
    World'''
```

In strings, you can include non-graphic characters through escape sequences. Escape sequences are given in following table :

Escape sequence	What it does [Non-graphic character]	Escape sequence	What it does [Non-graphic character]
\\	Backslash (\)	\r	Carriage Return (CR)
\'	Single quote (')	\t	Horizontal Tab (TAB)
\"	Double quote (")	\uxxxx	Character with 16-bit hex value xxxx (Unicode only)
\a	ASCII Bell (BEL)	\Uxxxxxxxx	Character with 32-bit hex value xxxxxxxx (Unicode only)
\b	ASCII Backspace (BS)	\v	ASCII Vertical Tab (VT)
\f	ASCII Formfeed (FF)	\ooo	Character with octal value ooo
\n	New line character	\xhh	Character with hex value hh
\N{name}	Character named name in the Unicode database (Unicode only)		

(ii) Numeric Literals

Numeric literals are numeric values and these can be one of the following types :

(a) int **(signed integers)** often called just **integers** or **ints**, are positive or negative whole numbers with no decimal point.

The integer literals can be written in :

- **Decimal form** : an integer beginning with digits 1-9. *e.g.*, 1234, 4100 etc.
- **Octal form** : an integer beginning with **0o** (zero followed by letter o) *e.g.*, 0o35, 0o77 etc. Here do remember that for Octal, **8** and **9** are invalid digits.
- **Hexadecimal form** : an integer beginning with **0x** (zero followed by letter X) *e.g.*, 0x73, 0xAF etc. Here remember that valid digits/letters for hexadecimal numbers are *0-9* and *A-F*.

(b) **Floating Point Literals. Floating point literals** or **real literals floats** represent real numbers and are written with a decimal point dividing the integer and fractional parts are numbers having fractional parts. These can be written in fractional form *e.g.*, –13.0, .75, 7. etc. or in **Exponent form** e.g., *0.17E5*, *3.E2*, *.6E4* etc.

(c) **Complex number literals** are of the form $a + bJ$, where a and b are *floats* and J (or j) represents $\sqrt{-1}$, which is an imaginary number). a is the real part of the number, and b is the imaginary part.

(iii) Boolean Literals

A Boolean literal in Python is used to represent one of the two Boolean values *i.e.*, **True** (Boolean true) or **False** (Boolean false). A Boolean literal can either have value as *True* or as *False*.

(iv) Special Literal None

Python has one special literal, which is **None**. The **None** literal is used to indicate absence of value.

Python can also store literal collections, in the form of **tuples** and **lists** etc.

4 . OPERATORS

Operators are tokens that trigger some computation / action when applied to variables and in other objects in an expression .

The operators can be arithmetic operators (+, –, *, /, %, **, //), bitwise operators (&, ^, |), shift operators (<<, >>), identity operators (is, is not), relational operators (>, <, >=, <=, =, !=), logical operators (and, or), assignment operator (=), membership operators (in, not in), and arithmetic-assignment operators (/=, +=, –=, */, %=, **=, //=).

Table operator precedence

Operator	Description
()	Parentheses (grouping)
**	Exponentiation
~x	Bitwise nor
+x, –x	Positive, negative (unary + , –)
*, /, //, %	Multiplication, division, floor division, remainder
+, –	Addition, subtraction
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
<, <=, >, >=, <>, !=, ==, is, is not	Comparisons (Relational operators), identity operators
not x	Boolean NOT
and	Boolean AND
or	Boolean OR

Highest

Lowest

5 . PUNCTUATORS

Punctuators are symbols used in programming languages to organise sentence structures.

Most common punctuators of Python programming language are :

```
' " #\ ( ) [ ] { } @ , : . ` =
```

BASIC STRUCTURE OF A PYTHON PROGRAM :

BAREBONES OF A PYTHON PROGRAM :

- **Expressions**, which are any legal combination of symbols that *represents a value*.
- **Statements**, which are programming instructions.
- **Comments**, which are the additional readable information to clarify the source code. Comments can be **single line comments**, that start with # and **multi-line comments** that can be either triple-quoted strings or multiple # style comments.
- **Functions**, which are named code-sections and can be reused by specifying their names (*function calls*).
- **Block(s) or suite(s)**, which is a group of statements which are part of another statement or a function. All statements inside a block or suite are indented at the same level.

```
# This program shows a program's components
# Definition of function SeeYou() follows
def SeeYou() :
    print ("Time to say Good Bye !!")
# Main program-code follows now
a = 15
b = a - 10
print (a + 3)
if b > 5 :                    # colon means it's a block
    print (" Value of 'a' was more than 15 initially.")
else :
    print ("Value of 'a' was 15 or less initially.")
SeeYou()                      # calling above defined function SeeYou()
```

Comments (*begin with #*)

Function

Statements

Expressions

Blocks

Indentation (*see indented lines*)

Inline comments (*comment beginning in the middle of a line*)

Function call

File Edit View Navigate Code Refactor Run Tools VCS Window Help

pythonProject 〉 HelloWorld.py

Project

pythonProject [Python] ~/pythonProject
- bin
- lib
- .gitignore
- HelloWorld.py
- pyvenv.cfg

External Libraries

Scratches and Consoles

HelloWorld.py

```python


# We use .py extension to save a python file

print("Hello World")

# We use the keyword "print" to get the output printed in the shell screen
```

A Sample Hello World Program

Run: HelloWorld

```
/home/meenusowjanya/pythonProject/bin/python /home/meenusowjanya/pythonProject/HelloWorld.py
Hello World

Process finished with exit code 0
```

Version Control | Find | Run | TODO | Problems | Terminal | Python Packages | Python Console | Services

Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes // Always download // Download once // Don't show again // Config... (today 10:34 AM) 4:1 LF UTF-8 4 spaces Python 3.8 (pythonProject)

DATA TYPES IN PYTHON :

Datatypes helps us to identify the type of the data and set of operations for it .

The datatypes offered in Python :

1. Numbers
2. Strings
3. Lists
4. Tuples
5. Dictionaries

(i) Data types for Numbers

Python offers following data types to store and process different types of numeric data :

(a) Integers

- Integers (signed)
- Booleans

(b) Floating-Point Numbers

(c) Complex Numbers

(a) **Integers.** There are *two* types of integers in Python :

(i) **Integers (signed).** It is the normal integer representation of whole numbers. *Python 3.x* provides single data type (***int***) to store any integer, whether *big* or *small*.

It is signed representation, *i.e.*, the integers can be positive as well as negative.

(ii) **Booleans.** These represent the truth values ***False*** and ***True***. The Boolean type is a subtype of plain integers, and Boolean values *False* and *True* behave like the values 0 and 1, respectively.

(b) **Floating Point Numbers.** In Python, floating point numbers represent machine-level **double precision floating point numbers (15 digit precision)**. The range of these numbers is limited by underlying machine architecture subject to available (virtual) memory.

(c) **Complex Numbers.** Python represents complex numbers in the form $A + Bj$. Complex numbers are a composite quantity made of two parts : *the real part* and the *imaginary part*, both of which are represented internally as ***float*** values (floating point numbers).

You can retrieve the two components using attribute references. For a complex number **z** :

- **z.real** gives the *real part*.
- **z.imag** gives the *imaginary part* as a float, not as a complex value.

NOTE

Python represents complex numbers as a pair of floating point numbers.

(ii) Data Type for Strings

All strings in Python 3.x are sequences of *pure Unicode characters*. *Unicode* is a system designed to represent every character from every language. A string can hold any type of known characters *i.e.*, *letters*, *numbers*, and *special characters*, of any known scripted language.

Following are all legal strings in Python :

"abcd" , "1234", '$%^&', '????', "ŠÆËá" , "αβγ", 'इक',
"बस", "ت ح ذ ث ج"

A Python string is a sequence of characters and each character can be individually accessed using its **index**.

NOTE

Valid string indices are *0, 1, 2* ... upto *length-1* in forward direction and *–1, –2, –3*... *– length* in backward direction.

(iii) Lists

A List in Python represents a group of comma-separated values of any datatype between square brackets *e.g.*, following are some lists :

```
[1, 2, 3, 4, 5]
['a', 'e', 'i', 'o', 'u']
['Neha', 102, 79.5]
```

In list too, the values internally are numbered from 0 (zero) onwards *i.e.*, first item of the list is internally numbered as 0, second item of the list as 1, 3rd item as 2 and so on.

(iv) Tuples

Tuples are represented as group of comma-separated values of any date type within parentheses, *e.g.*, following are some tuples :

```
p = (1, 2, 3, 4, 5)
q = (2, 4, 6, 8)
r = ('a', 'e', 'i', 'o', 'u')
h = (7, 8, 9, 'A', 'B', 'C')
```

NOTE

Values of type **list** are *mutable i.e.*, changeable – one can change / add / delete a list's elements. But the values of type **tuple** are *immutable i.e.*, non-changable ; one cannot make changes to a tuple.

(v) Dictionaries

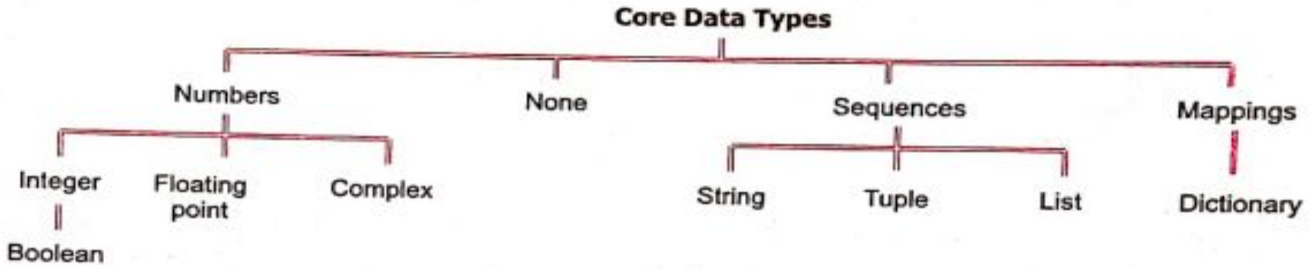
The *dictionary* is an unordered set of comma-separated **key : value** pairs, within { }, with the requirement that within a dictionary, no two keys can be the same (*i.e.*, there are unique keys within a dictionary). For instance, following are some dictionaries :

```
{'a' : 1 , 'e' : 2, 'i' : 3, 'o' : 4, 'u' : 5}
```

```
>>> vowels = {'a' : 1, 'e' : 2, 'i' : 3, 'o' : 4, 'u' : 5}
>>> vowels['a']
1
```

Here 'a', 'e', 'i', 'o' and 'u' are the keys of dictionary vowels; 1, 2, 3, 4, 5 are values for these keys respectively.

Following figure summarizes the core data types of Python.



File Edit View Navigate Code Refactor Run Tools VCS Window Help

pythonProject 〉 DatatypesAndOperators.py

DatatypesAndOperators

Project

pythonProject [Python] ~/pythonProject
- bin
- lib
- .gitignore
- DatatypesAndOperators.py
- HelloWorld.py
- pyvenv.cfg

External Libraries

Scratches and Consoles

HelloWorld.py | DatatypesAndOperators.py

```python
# A sample program for Datatypes , Operators
number1 = 12                # Integer
float1 = 12.123             # Floating point
complex1 = 12 + 13j         # Complex numbers
name = "Meenu"              # Single line String
sentence = '''
       I love Python
           '''              # Multi line String
boolean1 = True             # Boolean
list1 = [1,"Meenu",12.2472,True]      # List
tuple1 = (1,"Meenu",12731.2376,False) # Tuple
dict1 = {"name": "Meenu", "age": 18} # Dictionary

print (number1 + float1)  # Arithmetic operator
print (name + sentence)   # String concatenation
print (number1 > float1)  # Comparison Operator
```

Run: DatatypesAndOperators

```
/home/meenusowjanya/pythonProject/bin/python /home/meenusowjanya/pythonProject/DatatypesAndOperators.py
24.122999999999998
Meenu
       I love Python

False

Process finished with exit code 0
```

Version Control | Find | Run | TODO | Problems | Terminal | Python Packages | Python Console | Services

Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes // Always download // Download once // Don't show again // Con... (today 10:34 AM)

16:48 LF UTF-8 4 spaces Python 3.8 (pythonProject)

1.7 MUTABLE AND IMMUTABLE TYPES

The Python data objects can be broadly categorized into *two – mutable* and *immutable* types, in simple words changeable or modifiable and non-modifiable types.

Immutable Types

The immutable types are those that can never change their value *in place*. In Python, the following types are immutable : *integers, floating point numbers, Booleans, strings, tuples.*

In immutable types, the **variable names** are stored as references to a value-object. Each time you change the value, the variable's reference memory address changes. See following explanation for sample code given below :

```
P = 5
q = P
r = 5
      ⋮
P = 10
r = 7
q = r
```

- Initially these three statements are executed :

```
p = 5
q = p
r = 5
```

All variables having same value reference and the same value object i.e., p, q, r will all reference same integer

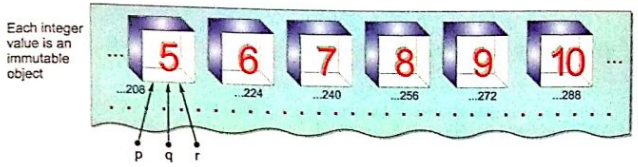


Figure 1.4

- When the next set of statements execute, *i.e.*,

```
p = 10
r = 7
q = r
```

then these variable names are made to point to different integer objects.

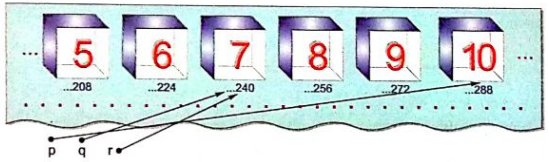


Figure 1.5

Mutable Types

Mutability means that in the same memory address, new value can be stored as and when you want. The types that do not support this property are **immutable types**.

The mutable types are those whose values can **be changed in place**. Only three types are mutable in Python.

These are : *lists, dictionaries* and *sets*.

To change a member of a list, you may write :

```
Chk = [2, 4, 6]
Chk[1] = 40
```

It will make the list namely **Chk** as [2, 40, 6].

NOTE

Python frontloads some commonly used values in memory. Each variable referring to that value actually stores that memory address of the value. Multiple variables/identifiers can refer to a value. Internally Python keeps count of how many identifiers/ variables are referring to a value.

NOTE

Mutable objects are :
list, dictionary, set

Immutable objects:
int, float, complex, string, tuple

PYTHON RUNTIME INPUT :

input( ) command

```
name = input("Enter your name : ")            # String input
age = int(input("Enter your age : "))         # Integer input
ratings = float(input("Enter your ratings : "))  # Float input
interests = eval(input("Enter your hobbies : "))# Objects input

print(name)
print(age)
print(ratings)
print(interests)
```

```
/home/meenusowjanya/pythonProject/bin/python /home/meenusowjanya/pythonProject/RuntimeInputs.py
Enter your name : Meenu
Enter your age : 18
Enter your ratings : 16521.23
Enter your hobbies : ["Singing","Dancing"]
Meenu
18
16521.23
['Singing', 'Dancing']

Process finished with exit code 0
```

1.8.4 Type Casting (Explicit Type Conversion)

An explicit type conversion is user-defined conversion that forces an expression to be of specific type. The explicit type conversion is also known as **Type Casting**.

Type casting in Python is performed by <type>( ) function of appropriate data type, in the following manner :

```
<datatype> (expression)
```

where <datatype> is the data type to which you want to type-cast your expression.

For example, if we have ($a = 3$ and $b = 5.0$), then

```
int(b)
```

will cast the data-type of the expression as **int**.

TYPE CASTING

The explicit conversion of an operand to a specific type is called type casting.

CONTROL STATEMENTS :

There are 3 main types of control statements in Python :

1. Loops
2. Conditions
3. Jumps

FOR AND FOR - EACH LOOPS

ForLoop.py

```python
# For Loop
print("For Loop starts")
for i in range(1,10):
    print("Hello World")
    print(i)
print(" For Loop ended")

# For Each Loop
print("For Each Loop 1 starts")
for i in "Meenu":
    print(i)
print("For Each Loop 1 Ends")
print("For Each Loop 2 starts")
for i in [1,12,"Meenu"]:
    print(i)
print("For Each Loop 2 Ends")
```

Run: ForLoop

```
/home/meenusowjanya/pythonProject/bin/python /home/meenusowjanya/pythonProject/Loops/ForLoop.py
For Loop starts
Hello World
1
Hello World
2
Hello World
3
Hello World
4
Hello World
5
Hello World
6
Hello World
7
Hello World
8
Hello World
9
 For Loop ended
For Each Loop 1 starts
M
e
e
n
u
For Each Loop 1 Ends
For Each Loop 2 starts
1
12
Meenu
For Each Loop 2 Ends

Process finished with exit code 0
```

Version Control | Run | Python Packages | TODO | Python Console | Problems | Terminal | Services

PEP 8: W292 no newline at end of file

PROGRAM **OUTPUT**

WHILE LOOP

File Edit View Navigate Code Refactor Run Tools VCS Window Help

pythonProject 〉 Loops 〉 WhileLoop.py

Project

pythonProject [Python] ~/pythonProject
- bin
- lib
- Loops
  - __init__.py
  - ForLoop.py
  - WhileLoop.py
- .gitignore
- DatatypesAndOperators.py
- HelloWorld.py
- pyvenv.cfg
- RuntimeInputs.py

External Libraries

Scratches and Consoles

WhileLoop.py

```python
# While Loop

age = 3

print("While Loop Starts")
while age <= 18:
    print("Not Eligible for Voting")
    age += 1
print("While Loops Ends")
```

Run: WhileLoop

```
/home/meenusowjanya/pythonProject/bin/python /home/meenusowjanya/pythonProject/Loops/WhileLoop.py
While Loop Starts
Not Eligible for Voting
Not Eligible for Voting
Not Eligible for Voting
Not Eligible for Voting
Not Eligible for Voting
Not Eligible for Voting
Not Eligible for Voting
Not Eligible for Voting
Not Eligible for Voting
Not Eligible for Voting
Not Eligible for Voting
Not Eligible for Voting
Not Eligible for Voting
Not Eligible for Voting
Not Eligible for Voting
Not Eligible for Voting
Not Eligible for Voting
While Loops Ends

Process finished with exit code 0
```

Version Control | Run | Python Packages | TODO | Python Console | Problems | Terminal | Services

PEP 8: W391 blank line at end of file

11:1 LF UTF-8 4 spaces Python 3.8 (pythonProject)

Conditional Statements :

IfCondition.py

```python
age = 18

# If Condition
print("If Condition Starts")
if age >= 18:
    print("Eligible for voting")
print("If Condition ends")

#  If - Else Condition
print("If - Else Condition starts")
if age >= 18:
    print("Eligible for voting")
else:
    print("Not eligible for voting")
print("If - Else Condition ends")

# If - Elif - Else Condition
print("If - Elif - Else Condition starts")
if age > 18:
    print("Eligible for voting")
elif age == 18:
    print("At the correct age for voting")
else:
    print("Not eligible for voting")
print("If - Elif - Else Condition ends")
```

Run: IfCondition

```
/home/meenusowjanya/pythonProject/bin/python /home/meenusowjanya/pythonProject/Conditions/IfCondition.py
If Condition Starts
Eligible for voting
If Condition ends
If - Else Condition starts
Eligible for voting
If - Else Condition ends
If - Elif - Else Condition starts
At the correct age for voting
If - Elif - Else Condition ends

Process finished with exit code 0
```

File Edit View Navigate Code Refactor Run Tools VCS Window Help

pythonProject 〉 JumpStatements 〉 JumpStatementsInLoops.py

JumpStatementsInLoops

Project

- pythonProject [Python] ~/pythonProject
  - bin
  - Conditions
    - __init__.py
    - IfCondition.py
  - JumpStatements
    - __init__.py
    - JumpStatementsInLoops.py
  - lib
  - Loops
    - __init__.py
    - ForLoop.py
    - WhileLoop.py
  - .gitignore
  - DatatypesAndOperators.py
  - HelloWorld.py
  - pyvenv.cfg
  - RuntimeInputs.py
- External Libraries
- Scratches and Consoles

JumpStatementsInLoops.py

```python

# A sample program which prints only odd numbers till 29
print("Loop starts")
for i in range(1,100):
    if (i % 2 == 0):
        continue          # Skips that particular Iteration value if the condition is true
    elif (i == 29):
        print("Loop broke")
        break             # Terminates the loop if the condition is true
    else:
        print(i)
print("Loop ends")

```

for i in range(1,100) 〉 elif (i == 29)

Run: JumpStatementsInLoops

```
/home/meenusowjanya/pythonProject/bin/python /home/meenusowjanya/pythonProject/JumpStatements/JumpStatementsInLoops.py
Loop starts
1
3
5
7
9
11
13
15
17
19
21
23
25
27
Loop broke
Loop ends
```

Jump Statements in Python

Version Control | Run | Python Packages | TODO | Python Console | Problems | Terminal | Services

Remove redundant parentheses

7:18 LF UTF-8 4 spaces Python 3.8 (pythonProject)

Thank you