Variables & Data Types -- 15 & 16

In [1]:
```
import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class',
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'globa
l', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise',
'return', 'try', 'while', 'with', 'yield']
```

In [2]:
```
len(keyword.kwlist)
```

Out[2]: 35

Differrent Data Types:

int ----->Numeric------->42 || float------>Numeric------>3.14 || complex-------->Numeric------>1 + 5j || bool--------->Boolean-------->True or False || str----------->"Text (Often treated as simple, though technically a sequence of characters)"--------->"""hello""" ||

In [3]:
```
Vn = 10
Vn #int
```

Out[3]: 10

In [4]:
```
flt = 2.5
flt #float
```

Out[4]: 2.5

In [5]:
```
strng = 'hello'
strng #string
```

Out[5]: 'hello'

In [6]:
```
active = True
active #bool
```

Out[6]: True

In [7]:
```
c = 10+20j
c
```

Out[7]: (10+20j)

In [8]:
```
c.real, c.imag
```

Out[8]: (10.0, 20.0)

In [9]:
```
d = 20+30j
print(c+d)
print(c-d)
print(c*d)
```

```
(30+50j)
(-10-10j)
(-400+700j)
```

In [10]:
```
a = 2.4
b =5.7
print(a+b)
```

```
8.1
```

In [11]:
```
percent = 70
percent
```

Out[11]: 70

String Concatenation

In [12]:
```
fn = 'Gudio'
mn = 'Van'
ln = 'Russom'
print(fn+mn,ln)
flln = fn+" "+mn+" "+ln
flln
```

```
GudioVan Russom
```

Out[12]: 'Gudio Van Russom'

In [13]:
```
print(flln)
```

```
Gudio Van Russom
```

In [14]:
```
type(flln)
```

Out[14]: str

In [15]:
```
print(type(percent))
```

```
<class 'int'>
```

In [16]:
```
# Variables with new assigned values

first_name = "ANIL"
last_name = "KUMAR"
country = "USA"
city = "NEW YORK"
age = 29
is_married = False
skills = ["Java", "SQL", "Spring Boot", "AWS", "Docker"]
person_info = {
    "firstname": "John",
    "lastname": "Doe",
```

```
    "country": "Canada",
    "city": "Toronto",
}

# --- Printing the new values stored in the variables ---

print("First name:", first_name)
print("First name length:", len(first_name))
print("Last name: ", last_name)
print("Last name length: ", len(last_name))
print("Country: ", country)
print("City: ", city)
print("Age: ", age)
print("Married: ", is_married)
print("Skills: ", skills)
print("Person information: ", person_info)
```

```
First name: ANIL
First name length: 4
Last name:  KUMAR
Last name length:  5
Country:  USA
City:  NEW YORK
Age:  29
Married:  False
Skills:  ['Java', 'SQL', 'Spring Boot', 'AWS', 'Docker']
Person information:  {'firstname': 'John', 'lastname': 'Doe', 'country': 'Canada',
'city': 'Toronto'}
```

In [17]:
```
first_name, last_name, country, age, is_married = ( "Durga", "Prasad", "Osaka", 250
print(first_name, last_name, country, age, is_married)
```

```
Durga Prasad Osaka 250 True
```

In [18]:
```
print(type(age), type(first_name), type(is_married))
```

```
<class 'int'> <class 'str'> <class 'bool'>
```

In [19]:
```
import sys
sys.version
```

Out[19]:
```
'3.13.7 | packaged by Anaconda, Inc. | (main, Sep  9 2025, 19:54:37) [MSC v.1929 6
4 bit (AMD64)]'
```

python variable = identifier = object

syntax (variable = value)

In [20]:
```
v@=16
v@ # Dont Use Special Variables
```

```
  Cell In[20], line 2
    v@ # Dont Use Special Variables
       ^
SyntaxError: invalid syntax
```

In [21]:
```
a_ = 10
a_
```

Out[21]: 10

In [22]:
```
1var = 45
1var
```

```
  Cell In[22], line 1
    1var = 45
    ^
SyntaxError: invalid decimal literal
```

In [23]:
```
def = 10 # Not Valid as def is used to define an user defined fucntion
```

```
  Cell In[23], line 1
    def = 10 # Not Valid as def is used to define an user defined fucntion
        ^
SyntaxError: invalid syntax
```

In [24]:
```
DEF = 10
DEF
```

Out[24]: 10

In [25]:
```
false = 10
false
```

Out[25]: 10

In [26]:
```
False = 10 # Not Valid as they are keywords
```

```
  Cell In[26], line 1
    False = 10 # Not Valid as they are keywords
    ^
SyntaxError: cannot assign to False
```

In [27]:
```
s = 'hello'
s
```

Out[27]: 'hello'

In [28]:
```
s1 = "hello"
s1
```

Out[28]: 'hello'

In [29]:
```
s2 = ''' hello '''
s2
```

Out[29]: ' hello '

In [30]:
```
s3 = '''Hi
            Guys'''
```

```
s3
```

Out[30]: 'Hi \n   Guys'

In [31]:
```
print(3 ** 2)  # exponential(**  -- square)
print(6 % 2)   # modulus(%)  -- remainder
print(6 // 2)  # Floor division operator(//)  -- quotient
```

```
9
0
3
```

Type Casting == Convert one Dataype to Other Dataype

All other data types to Int

In [32]:
```
int(2.4)
```

Out[32]: 2

In [33]:
```
int(True)
```

Out[33]: 1

In [34]:
```
int(true) #just variable is not considered
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[34], line 1
----> 1 int(true) #just variable is not considered

NameError: name 'true' is not defined
```

In [36]:
```
int (True, False) # Can't take more than two arguments
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[36], line 1
----> 1 int (True, False) # Can't take more than two arguments

TypeError: int() can't convert non-string with explicit base
```

In [37]:
```
int('10') # number based string can be converted
```

Out[37]: 10

In [38]:
```
int('ten') #text based string can't be converted
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[38], line 1
----> 1 int(      ) #text based string can't be converted

ValueError: invalid literal for int() with base 10: 'ten'
```

In [39]:
```
int(20+3j) # can't convert to complex
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[39], line 1
----> 1 int(20+3j) # can't convert to complex

TypeError: int() argument must be a string, a bytes-like object or a real number, no
t 'complex'
```

All other data types to float

In [40]:
```
float(20000)
```

Out[40]: 20000.0

In [41]:
```
float(2,3) #can't take more than two arguments
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[41], line 1
----> 1 float(2,3) #can't take more than two arguments

TypeError: float expected at most 1 argument, got 2
```

In [42]:
```
float(True)
```

Out[42]: 1.0

In [43]:
```
float(False)
```

Out[43]: 0.0

In [44]:
```
float('10')
```

Out[44]: 10.0

In [45]:
```
float('ten') # text based string can't be converted
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[45], line 1
----> 1 float(      ) # text based string can't be converted

ValueError: could not convert string to float: 'ten'
```

In [46]:
```
float(10+20j)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[46], line 1
----> 1 float(10+20j)

TypeError: float() argument must be a string or a real number, not 'complex'
```

In [47]:
```
f = 1e0
f
```

Out[47]: 1.0

In [48]:
```
f1 = 2e1
f1
```

Out[48]: 20.0

In [49]:
```
f2 = 2.4e2
f2
```

Out[49]: 240.0

In [50]:
```
f3 = 2.5e3
f3
```

Out[50]: 2500.0

In [51]:
```
type(f3)
```

Out[51]: float

All other data types to string

In [52]:
```
str(48)
```

Out[52]: '48'

In [53]:
```
str('48')
```

Out[53]: '48'

In [54]:
```
str(4.8)
```

Out[54]: '4.8'

In [55]:
```
str(10+20j)
```

Out[55]: '(10+20j)'

In [56]:
```
str(True)
```

Out[56]: 'True'

In [57]:
```
str(True,False) #cant take two arguments
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[57], line 1
----> 1 str(True,False) #cant take two arguments

TypeError: str() argument 'encoding' must be str, not bool
```

In [58]:
```
str(False)
```

Out[58]: 'False'

All other data types to boolean

In [59]:
```
bool(10)
```

Out[59]: True

In [60]:
```
bool(2.3)
```

Out[60]: True

In [61]:
```
bool(0)
```

Out[61]: False

In [62]:
```
bool()
```

Out[62]: False

In [63]:
```
bool(10+20j)
```

Out[63]: True

In [64]:
```
bool('10')
```

Out[64]: True

In [65]:
```
bool('ten')
```

Out[65]: True

All other data types to complex

In [66]:
```
complex(10)
```

Out[66]: (10+0j)

In [67]:
```
complex(10.20,30)
```

Out[67]: (10.2+30j)

In [68]:
```
complex(False)
```

Out[68]: 0j

In [69]:
```
complex(True)
```

Out[69]: (1+0j)

In [70]:
```
complex('10')
```

Out[70]: (10+0j)

In [71]:
```
complex(True,False)
```

Out[71]: (1+0j)

In [72]:
```
complex('ten') # can't take text based string as arg
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[72], line 1
----> 1 complex('ten') # can't take text based string as arg

ValueError: complex() arg is a malformed string
```

In [73]:
```
complex(10,20,30) #only 2 args needed given 3 args
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[73], line 1
----> 1 complex(10,20,30) #only 2 args needed given 3 args

TypeError: complex() takes at most 2 arguments (3 given)
```

In [74]:
```
complex('10', '20') #two string arg are not considered
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[74], line 1
----> 1 complex('10', '20') #two string arg are not considered

TypeError: complex() can't take second arg if first is a string
```

String Functions and Indexing and slicing and other

In [75]:
```
i = 12
i
id(i)
```

Out[75]: 140711978509576

In [76]:
```
j = 12
q = 12
print(id(i),id(j),id(q))
'''if(id(i) == id(j) == id(q)):
    print("Memory allocated at the same loction, pointers pointing to same location
else:
    print("Something went wrong!!")'''
```

```
140711978509576 140711978509576 140711978509576
```

Out[76]:
```
'if(id(i) == id(j) == id(q)):\n    print("Memory allocated at the same loction, po
inters pointing to same location but different variables")\nelse:\n    print("Some
thing went wrong!!")'
```

In [77]:
```
q=30
```

In [78]:
```
id(q)
```

Out[78]:
```
140711978510152
```

In [79]:
```
str = 'Statistics'
str
```

Out[79]:
```
'Statistics'
```

In [80]:
```
str[0]
```

Out[80]:
```
'S'
```

In [81]:
```
str[1]
```

Out[81]:
```
't'
```

In [82]:
```
str[0:11]
```

Out[82]:
```
'Statistics'
```

In [83]:
```
print(str[-10])
print(str[-9])
print(str[-8])
print(str[-7])
print(str[-6])
print(str[-5])
print(str[-4])
print(str[-3])
print(str[-2])
print(str[-1])
```

```
S
t
a
t
i
s
t
i
c
s
```

In [84]:
```
str
```

Out[84]:
```
'Statistics'
```

In [85]:
```
len(str)
```

Out[85]:
```
10
```

In [86]:
```
str[3:11]
```

Out[86]:
```
'tistics'
```

In [87]:
```
str[0:6]
```

Out[87]:
```
'Statis'
```

In [88]:
```
str[0:11:2]
```

Out[88]:
```
'Saitc'
```

In [89]:
```
str[0:11:4]
```

Out[89]:
```
'Sic'
```

In [90]:
```
str[:8:2]
```

Out[90]:
```
'Sait'
```

In [91]:
```
str[0::4]
```

Out[91]:
```
'Sic'
```

In [92]:
```
str[::3]
```

Out[92]:
```
'Stts'
```

In [93]:
```
str[:]
```

Out[93]:
```
'Statistics'
```

In [94]:
```
print(s,str)
```

```
hello Statistics
```

In [95]:
```
print(s + str)
```

```
helloStatistics
```

Operators

TYPES OF OPERATORS: 1. Arithmatic Operator 2. Assignment Operator 3. Relational Operator 4. logical Operator 5. Unary Operator

In [96]:
```
a = 15
b = 4
print(f"a + b (Addition): {a + b}")
print(f"a - b (Subtraction): {a - b}")
print(f"a * b (Multiplication): {a * b}")
print(f"a / b (Division): {a / b}")
print(f"a // b (Floor Division): {a // b}")
print(f"a % b (Modulus/Remainder): {a % b}")
print(f"a ** 2 (Exponentiation): {a ** 2}")
```

```
a + b (Addition): 19
a - b (Subtraction): 11
a * b (Multiplication): 60
a / b (Division): 3.75
a // b (Floor Division): 3
a % b (Modulus/Remainder): 3
a ** 2 (Exponentiation): 225
```

In [97]:
```
x = 10
print(f"Initial x: {x}")
x += 5
print(f"x after x += 5: {x}")
x -= 2
print(f"x after x -= 2: {x}")
x *= 3
print(f"x after x *= 3: {x}")
x /= 13
print(f"x after x /= 13: {x}")
```

```
Initial x: 10
x after x += 5: 15
x after x -= 2: 13
x after x *= 3: 39
x after x /= 13: 3.0
```

In [98]:
```
print(f"a == b (Equal to): {a == b}")
print(f"a != b (Not equal to): {a != b}")
print(f"a > b (Greater than): {a > b}")
print(f"a < b (Less than): {a < b}")
print(f"a >= b (Greater than or equal to): {a >= b}")
print(f"a <= 15 (Less than or equal to): {a <= 15}")
```

```
a == b (Equal to): False
a != b (Not equal to): True
a > b (Greater than): True
a < b (Less than): False
a >= b (Greater than or equal to): True
a <= 15 (Less than or equal to): True
```

In [99]:
```python
p = True
q = False
print(f"p is {p}, q is {q}")
print(f"p and q (AND): {p and q}")
print(f"p or q (OR): {p or q}")
print(f"not p (NOT): {not p}")
```

```
p is True, q is False
p and q (AND): False
p or q (OR): True
not p (NOT): False
```

In [100…
```python
y = 7
z = -10
print(f"Initial y: {y}")
print(f"Initial z: {z}")
# Unary Plus: Returns the value unchanged
print(f"+y (Unary Plus): {+y}")  # 7
# Unary Minus: Negates the value
print(f"-y (Unary Minus): {-y}")  # -7
# Unary Minus can flip a negative to a positive
print(f"-z (Negation of -10): {-z}")
```

```
Initial y: 7
Initial z: -10
+y (Unary Plus): 7
-y (Unary Minus): -7
-z (Negation of -10): 10
```

In [101…
```python
# The complement operator (~) is applied to an integer.
number = 5
complement_result = ~number
# complement formula: -(number + 1)
print(f"Original number: {number}")
print(f"Complement (~) result: {complement_result}")
print(f"Check with formula: {-(number + 1)}")
negative_number = -10
print(f"Original number: {negative_number}")
print(f"Complement (~) result: {~negative_number}")
```

```
Original number: 5
Complement (~) result: -6
Check with formula: -6
Original number: -10
Complement (~) result: 9
```

In [102…
```python
bin(number)
```

Out[102…
```
'0b101'
```

In [103…
```
20
bin(20)
```

Out[103… '0b10100'

In [104…
```
50
bin(50)
```

Out[104… '0b110010'

In [105…
```
# Octal
number1 = 5
number2 = 20
number3 = 50
number4 = 64
number5 = 0o55
print(f"Decimal 5:   {oct(number1)}")
print(f"Decimal 20:  {oct(number2)}")
print(f"Decimal 50:  {oct(number3)}")
print(f"Decimal 64:  {oct(number4)}")
print(f"Decimal num5: {number5}")
```

```
Decimal 5:   0o5
Decimal 20:  0o24
Decimal 50:  0o62
Decimal 64:  0o100
Decimal num5: 45
```

In [ ]: