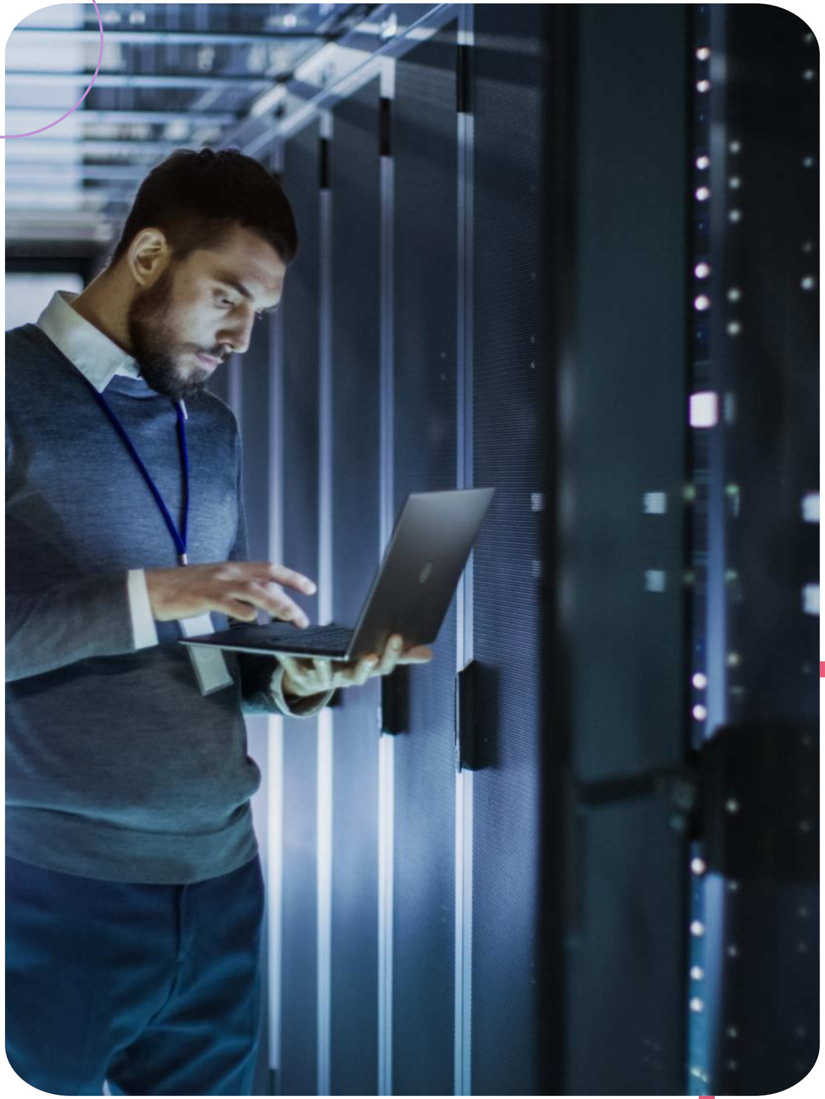
Diploma in

Python Programming

Basic features of Python programming



Objectives

- Examine three basic Python data types and their use cases
- Introduce variables and literals and show how they are applied
- Explain Python statements and comments
- Discuss the importance of indentations in Python

Important facts!

- Everything in Python programming is an object
- Every value in Python has a data type.
- Data types are called ‘instances of classes’
- An ‘instance of a class’ is then referred to as an ‘object’

Basic data types

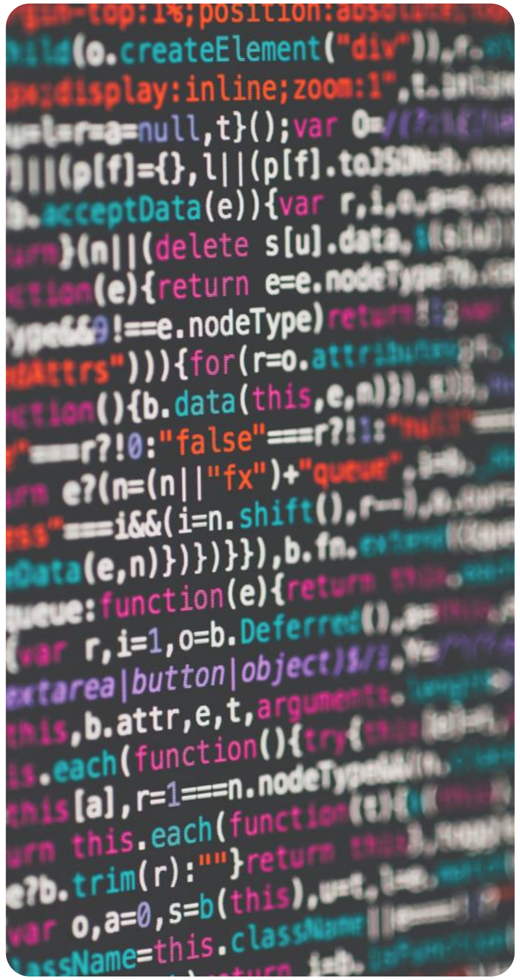
Three basic data types

- Number data types
- String data types
- Boolean data types

01

02

03



Number data types

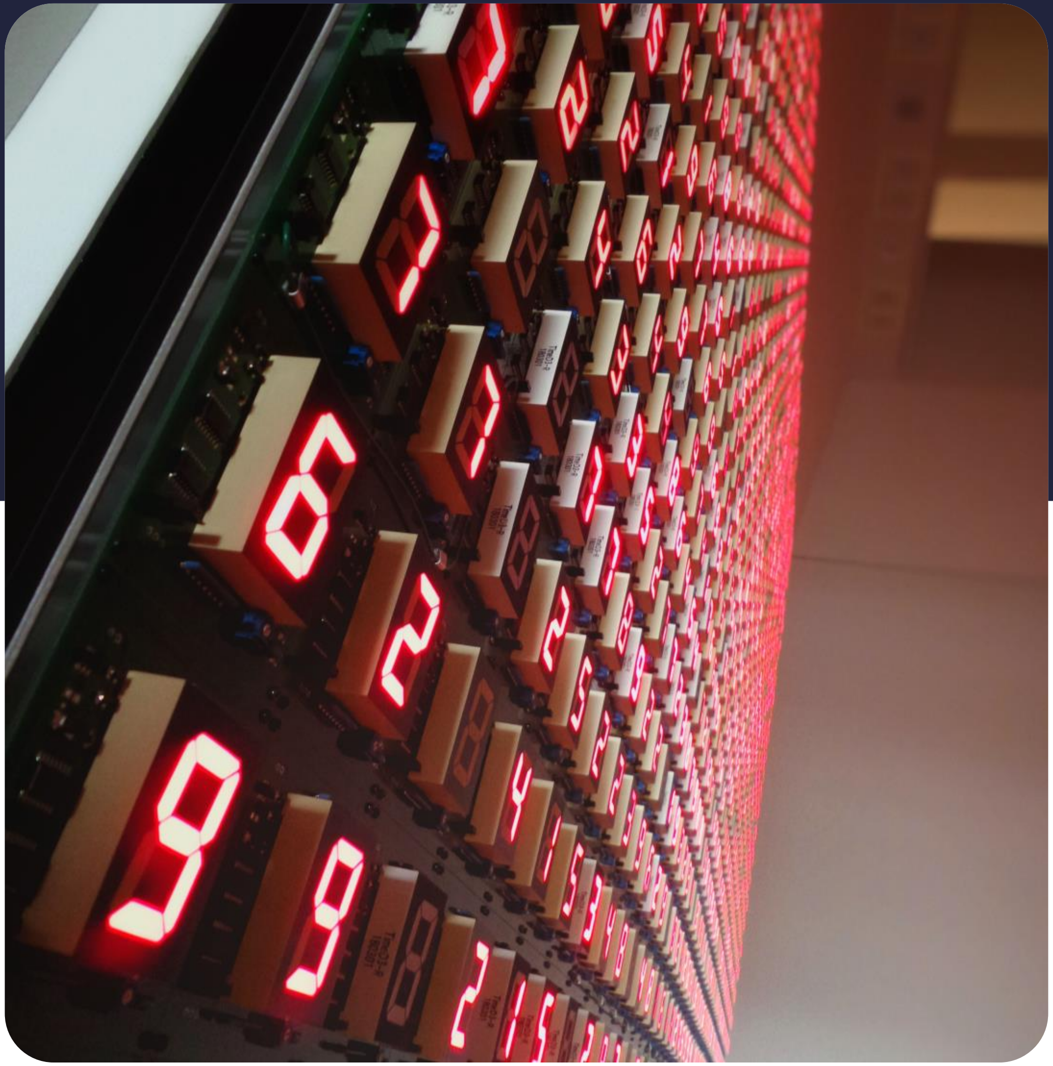
- Integers
- Floating-point numbers
- Complex numbers

DID YOU KNOW?

Code process

- Input
- Process
- Output

Input	(1)
print(2 + 3)	(2)
Output	(3)
5	(4)



Integers

- Positive and negative whole numbers (including 0)
- No limit to length of an integer value
- Can be used for standard addition, subtraction, division and multiplication operations in Python

Integers

Addition

$$\text{Input} \tag{1}$$
$$\text{print}(2 + 3) \tag{2}$$
$$\text{Output} \tag{3}$$
$$5 \tag{4}$$

Subtraction

$$\text{Input} \tag{5}$$
$$\text{print}(5 - 2) \tag{6}$$
$$\text{Output} \tag{7}$$
$$3 \tag{8}$$

Division

$$\text{Input} \tag{13}$$
$$\text{print}(10/2) \tag{14}$$
$$\text{Output} \tag{15}$$
$$5 \tag{16}$$

Multiplication

$$\text{Input} \tag{9}$$
$$\text{print}(3 * 2) \tag{10}$$
$$\text{Output} \tag{11}$$
$$6 \tag{12}$$

Integers

Type() Function

Input (17)

print(type(4)) (18)

Output (19)

$< \text{class}^0\text{int}^0 >$ (20)

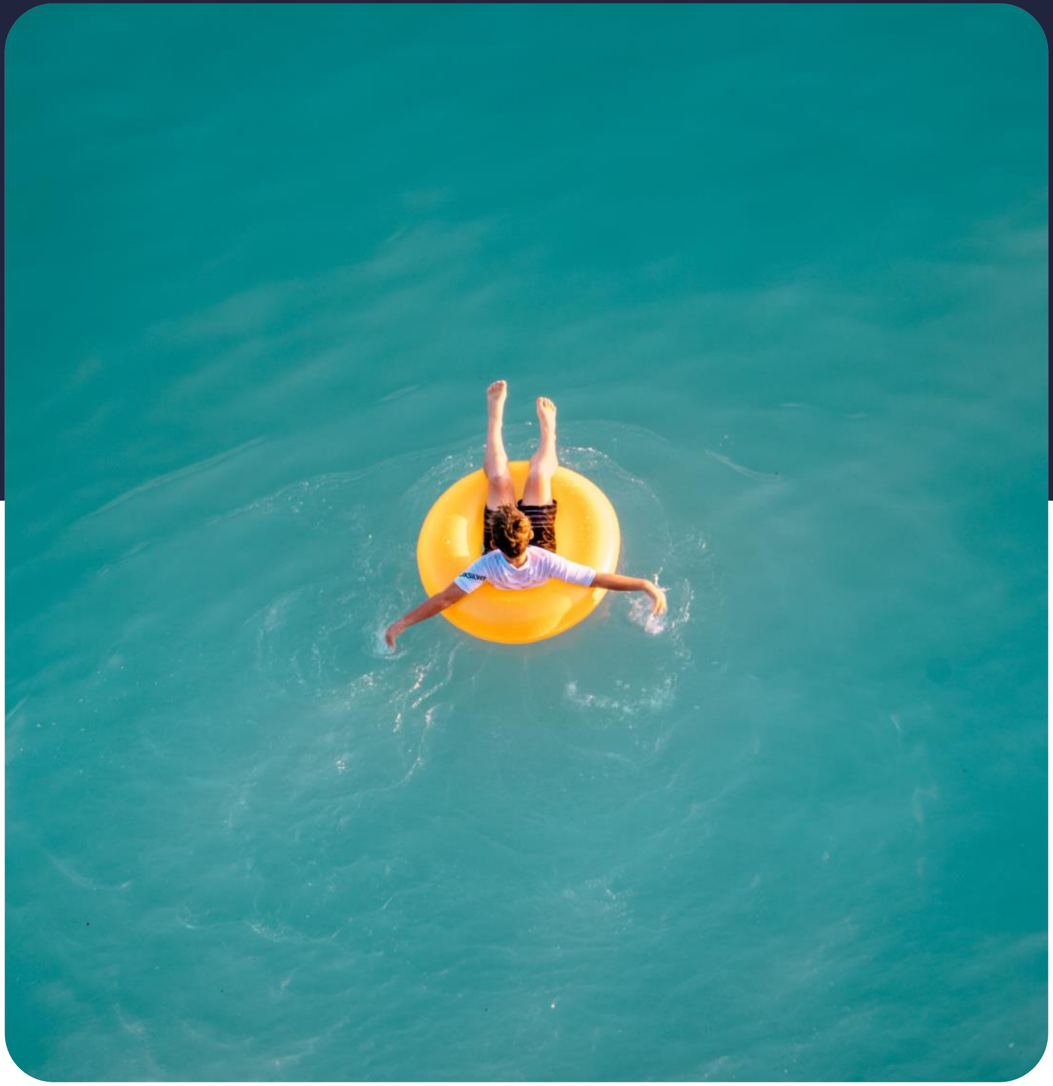
Class integer

Input (21)

print(isinstance(2,int)) (22)

Output (23)

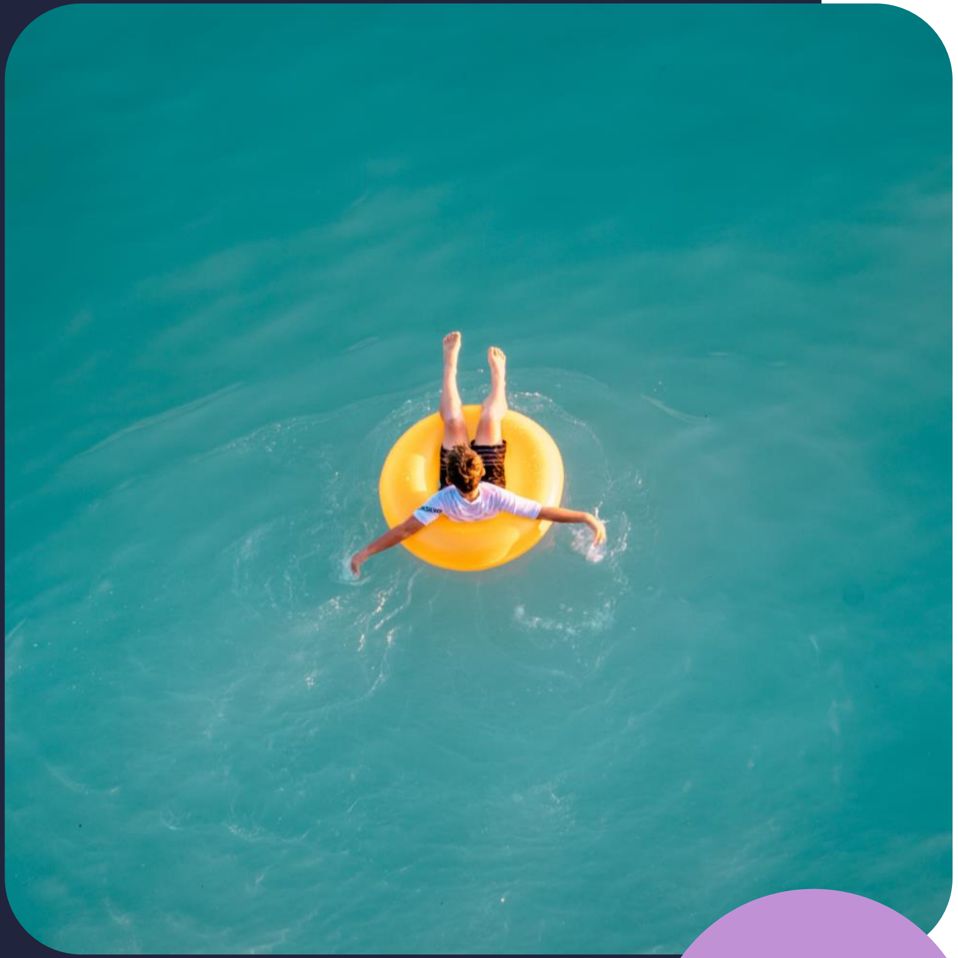
True (24)



Floating-point numbers

- Numbers are values with decimal points
- For example, 5.1

Floating-point numbers



Type[]

Input (25)

print(type(5.0)) (26)

Output (27)

< class⁰float⁰ > (28)

Input (29)

print(type(3.2)) (30)

Output (31)

< class0float0 > (32)

Isinstance[]

Input (33)

print(isinstance (34)
(5.0,float))

Output (35)

True (36)

Input (37)

print(isinstance (38)
(5.0,int))

Output (39)

False (34)

Complex numbers

‹real part› + ‹imaginary part›j

Input (41)

print(type(2 + 3j)) (42)

Output (43)

$< class^{0}complex^{0} >$ (44)

Strings data types

- Sequences of character data
- Immutable
- Represented by single or double quotes
- Triple quotes for multi-line strings

Examples of a string

Single quote

Input (45)

print(^{0}Hello I am a string^0) (46)

Output (47)

Hello I am a string (48)

Examples of a string

Confirmation

Input (49)

Print(type(^{0}Hello I am a string^{0})) (50)

Output (51)

$<$ class ^{0}string^{0} $>$ (52)

Examples of a string

Double quote

Input (53)

Print(type(^{00}Hello I am a string^{00})) (54)

Output (55)

$<$ class ^{0}string^{0} $>$ (56)

Examples of multi-line strings

Triple quote

Input (57)

print(type($(^{000}$Hello I (58)

am a (59)

string^{000})) (60)

Output (61)

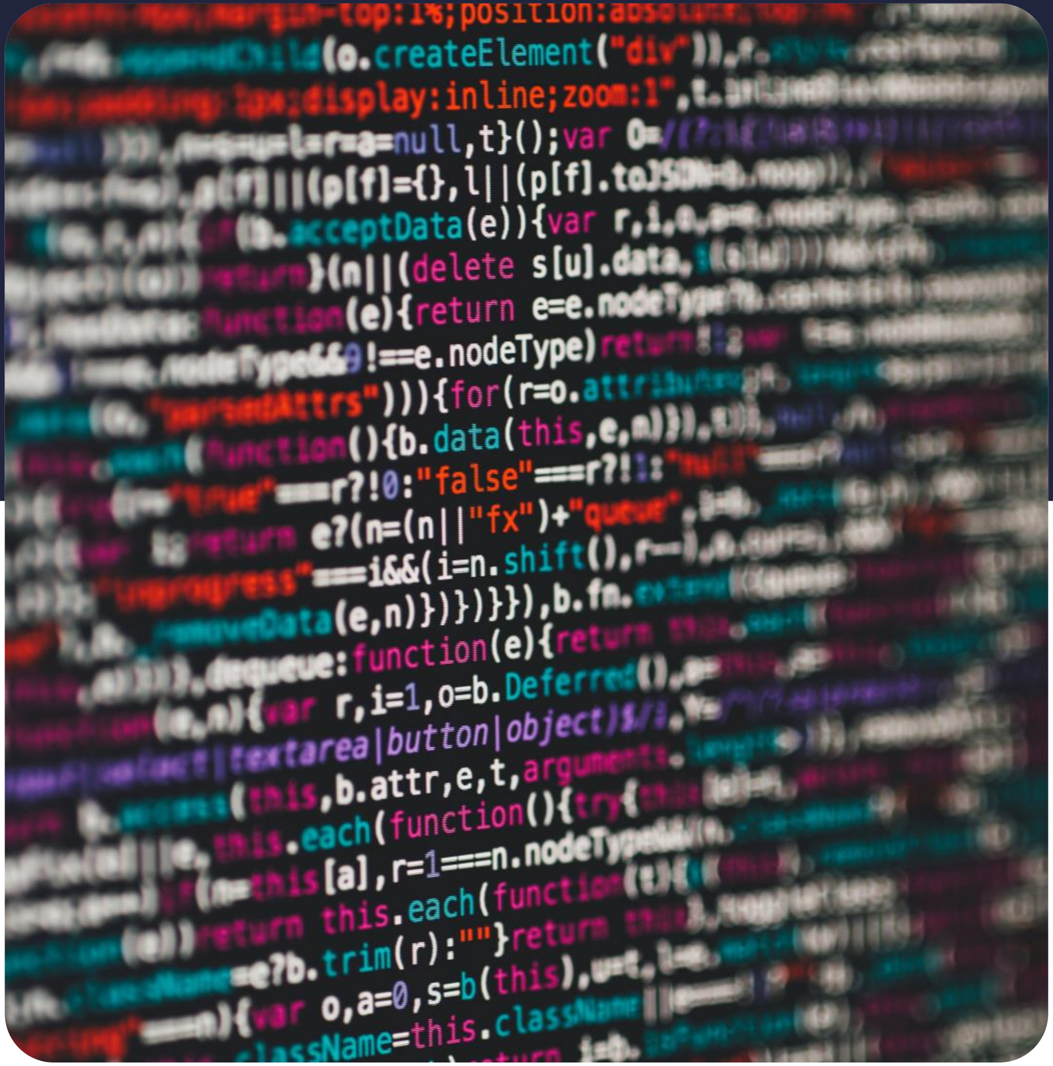
$<$ class^{0}string^{0} $>$ (62)

Examples of multi-line strings

Double triple quote

Input (63)
print(type("""Hello I (64)
am a (65)
string""")) (66)
Output (67)
< class⁰string⁰ > (68)

Boolean data types



- Objects have TRUE or FALSE value

Examples of a Boolean data type

$$Input \tag{69}$$

$$print(type(True)) \tag{70}$$

$$Output \tag{71}$$

$$< class\ {}^{0}bool^{0} > \tag{72}$$

and

$$Input \tag{73}$$

$$print(type(False)) \tag{74}$$

$$\text{Output} \tag{75}$$

$$< class\ {}^{0}bool^{0} > \tag{76}$$

DID YOU KNOW?

You can convert between data types using different type conversion functions such as *int()*, *float()* and *str()*.

Examples of a Boolean data type

Data type

Input (77)

print(float(5)) (78)

Output (79)

5.0 (80)

Truncate

Input (81)

print(int(10.6)) (82)

Output (83)

10 (84)

Examples of a Boolean data type

Compatible values

Input (85)

print(float(^{0}3.4^0)) (86)

Output (87)

3.4 (88)

String data type

Input (89)

print(str(34)) (90)

Output (91)

^{0}34^0 (92)

Python variables and literals

Pic needs to be of containers

A variable is a named location with memory that stores data.

Examples of a Boolean data type

Containers

Input	(93)
num = 5	(94)
print(num)	(95)
Output	(96)
5	(97)

Examples of a Boolean data type

Value assignment

Input	(98)
num = 5	(99)
num = 10	(100)
print(num)	(101)
Output	(102)
10	(103)

Variables

Variables can contain any type of data types

surname = ‘Smith’ (104)

Examples of a Boolean data type

Containers

```
Input                    (105)
surname = ‘Smith’        (106)
surname = ‘Denga’        (107)
print(surname)           (108)
Output                   (109)
Denga                    (110)
```

Examples of changing a string data type to a number or Boolean data type

Example 1

Input
```
surname = 'Smith'
surname = 10
print(surname)
```
Output
```
10
```

Example 2

Input
```
surname = 10
surname = 'Smith'
print(surname)
```
Output
```
Smith
```

Example 3

Input
```
surname = 10
surname = True
print(surname)
```
Output
```
True
```

DID YOU KNOW?

You can assign the same value to multiple variables

Example

```
Input                                (129)
age,name,salary = 77                 (130)
print(age)                           (131)
print(num )                          (132)
print(salary)                        (133)
Output                               (134)
77                                   (135)
77                                   (136)
77                                   (137)
```

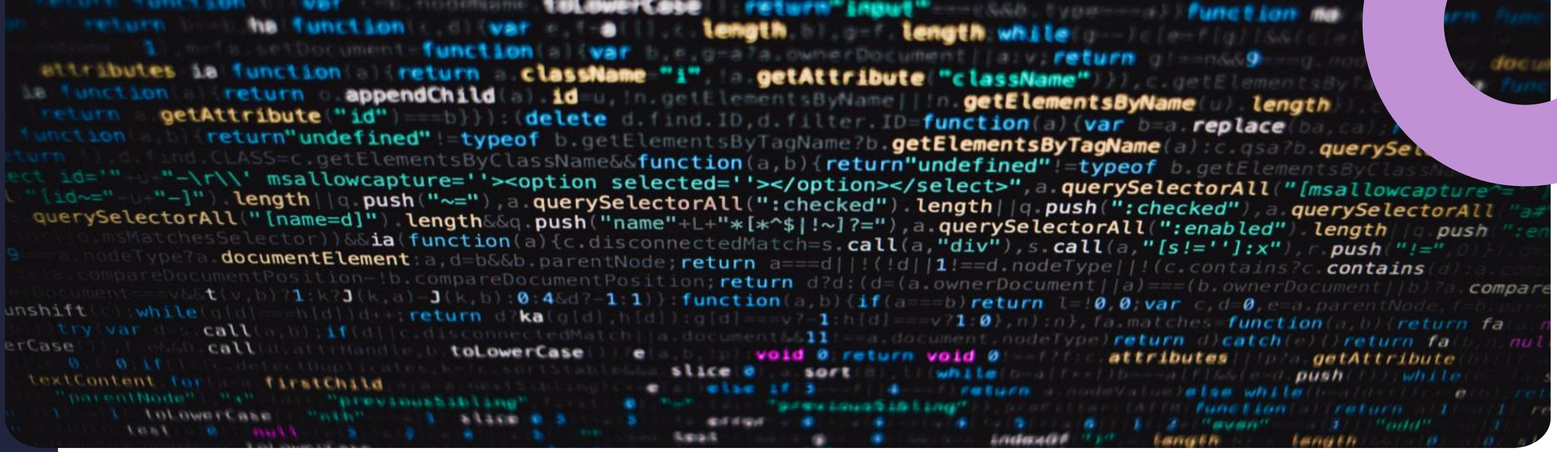
Examples of assigning the same value to multiple variables

Input

```
age,name,salary = 77,“Smith”,1000
print(age)
print(num)
print(salary)
```

Output

```
77
Smith
1000
```

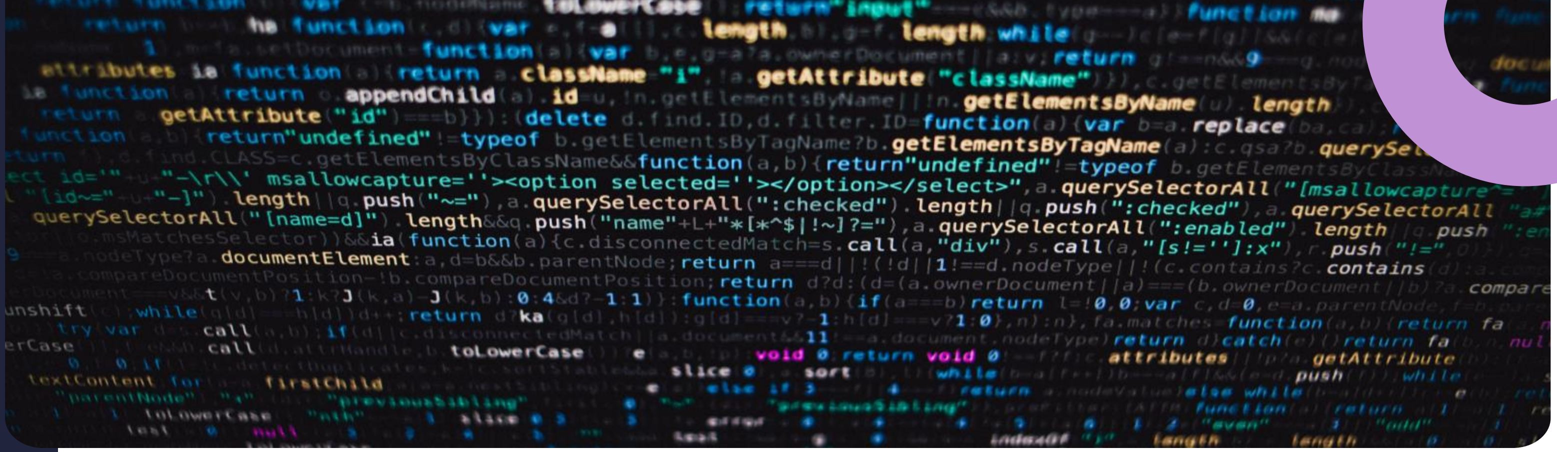


6 rules to remember when naming your variables

01

Variable names should have a combination of letters in uppercase (A to Z), or lowercase (a to z), or digits (0 to 9) AND you use an underscore (_) when you are combining two different words.

lucky_number	(148)
MySurname	(149)
LuckyNumber	(150)
mySurname	(151)
luckyName	(152)



6 rules to remember when naming your variables

02

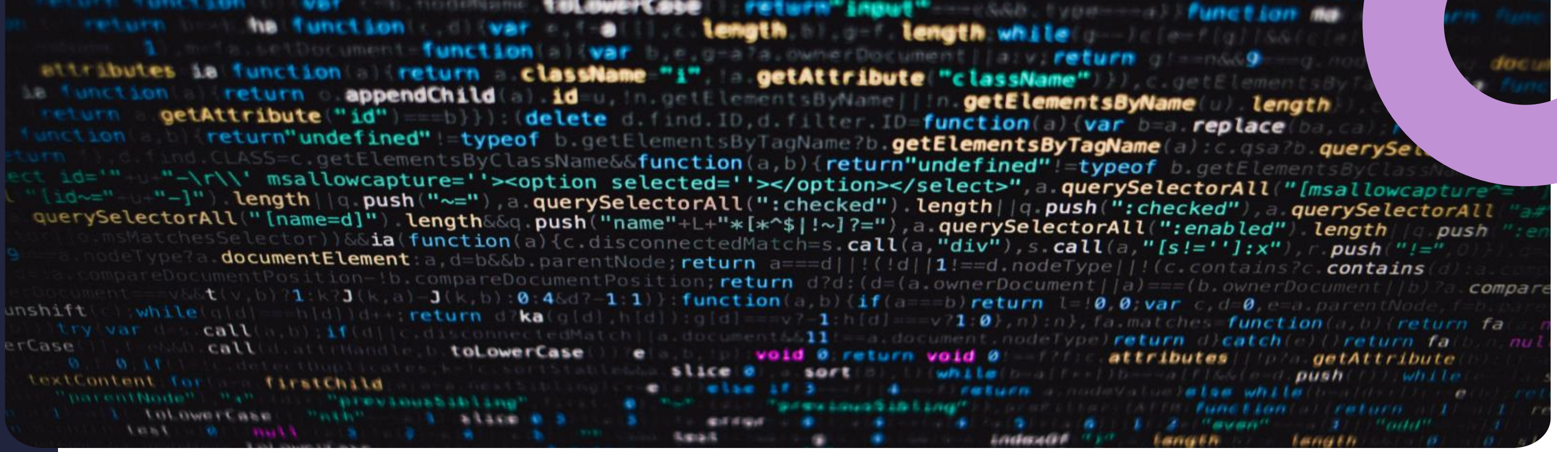
Variable names should make sense. For example, if you want to create a variable that will store your age, creating your variable as.

age = 77 (153)

and not

a = 77 (154)

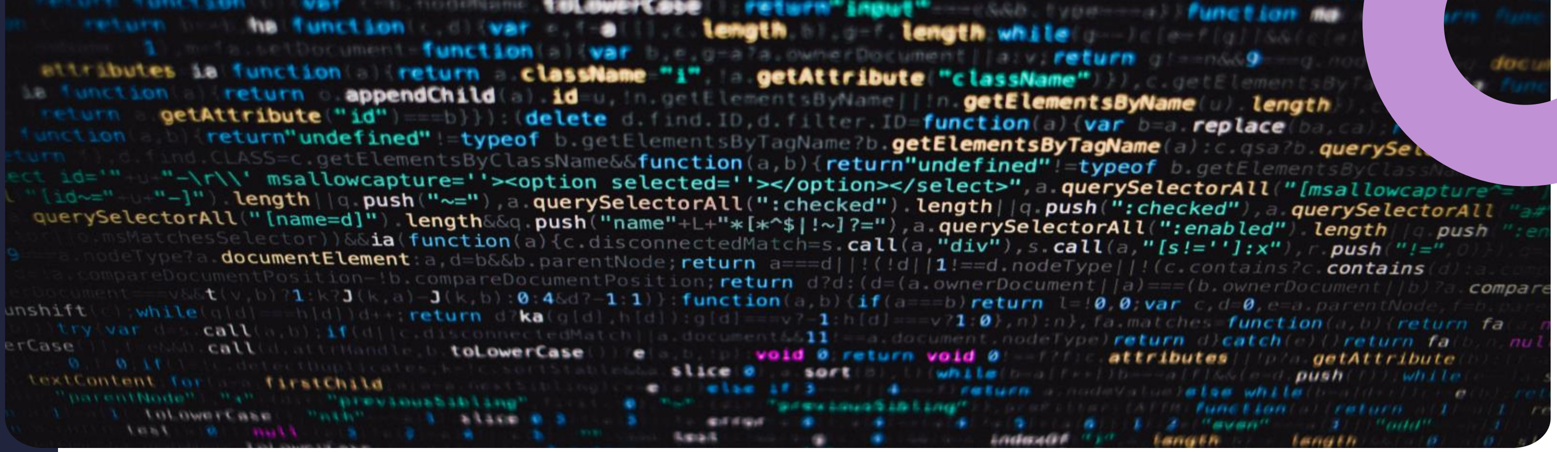
6 rules to remember when naming your variables



03

Variable names with two words must include an underscore to separate the words. Let's say you want to create a variable name for your lucky number, you can name your variable like this.

```
lucky_number = 10                    (155)
```



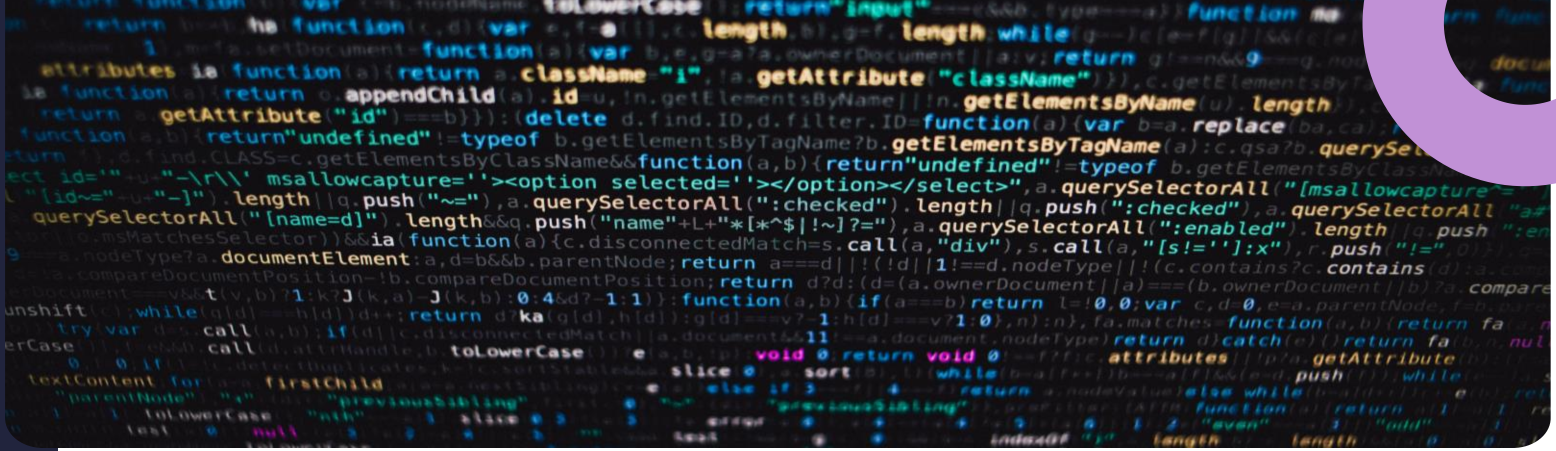
6 rules to remember when naming your variables

04

Variable names in Python are case sensitive. For example, the two variable names, Lucky–number and lucky–number will be treated differently in Python even though they are pronounced the same.

GRAV ITY = 9.8 (156)

SPEED_OF _LIGHT = 299792458 (157)

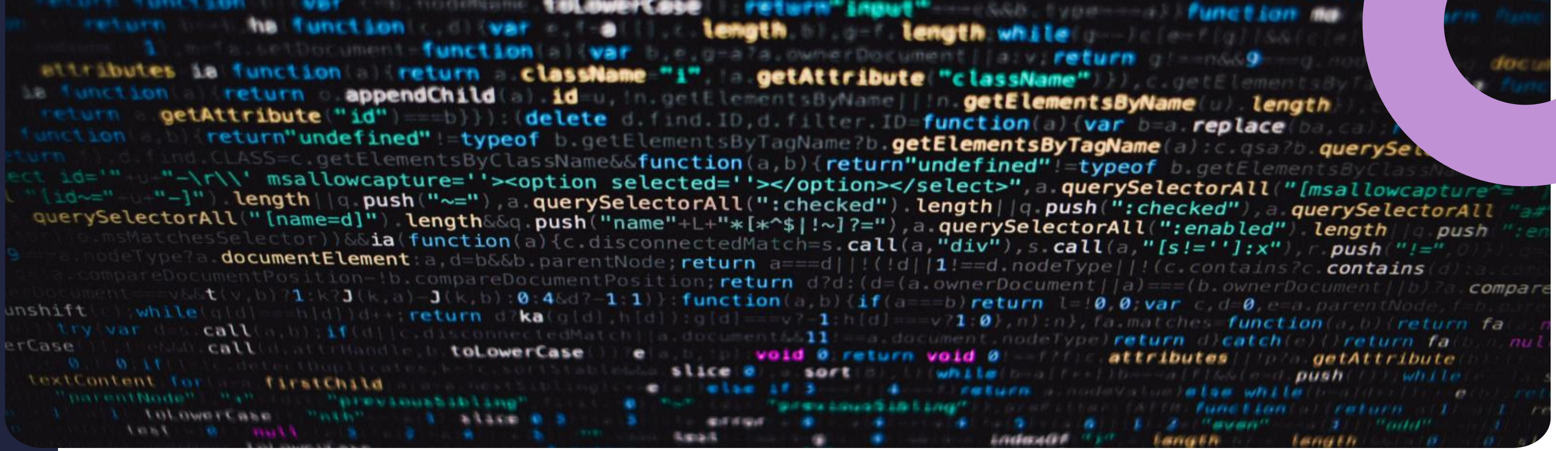


6 rules to remember when naming your variables

05

Variable names should never start with a digit.

94_year
96 year
4people



6 rules to remember when naming your variables

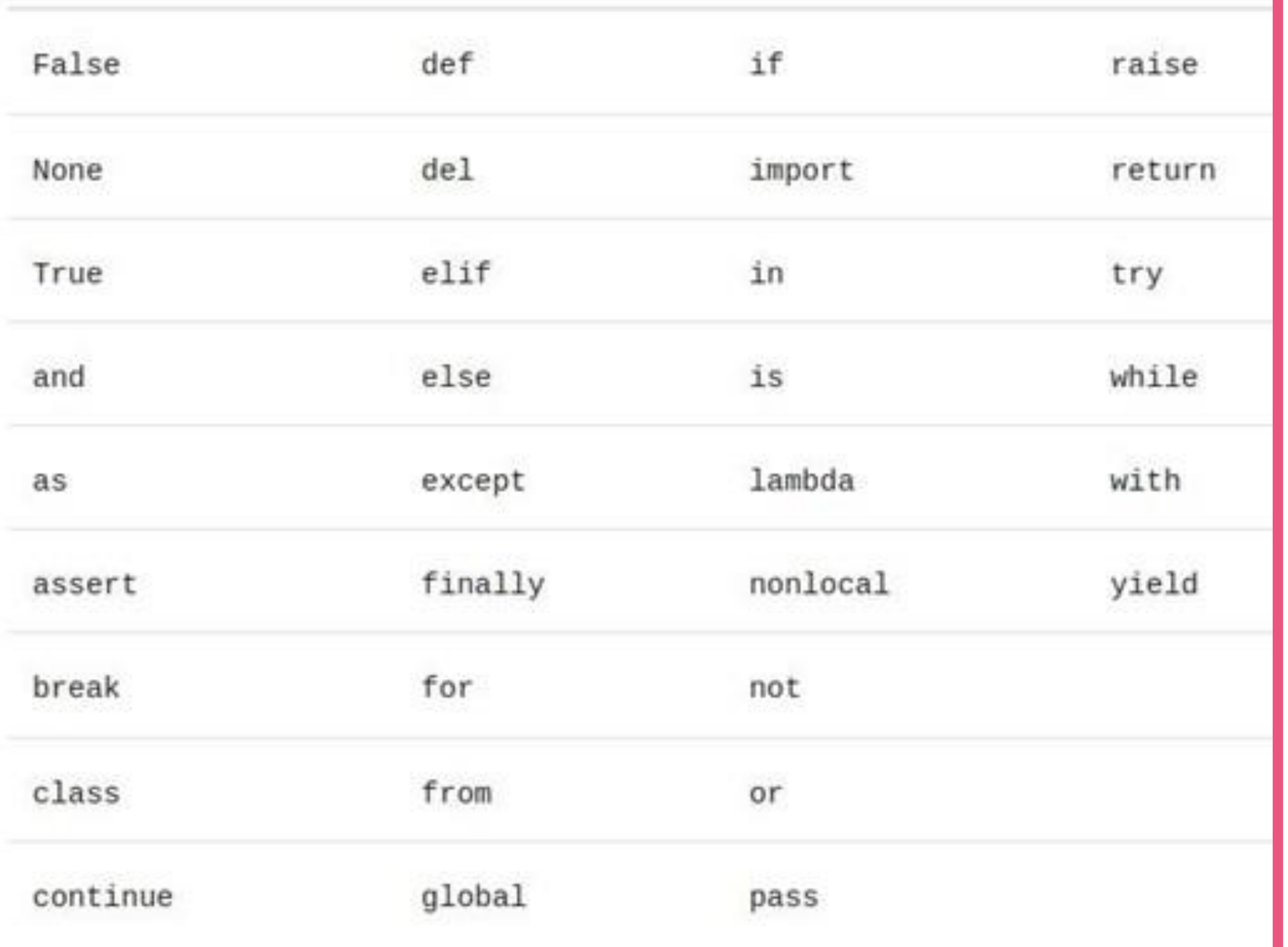
06

Variable names should never include characters such as @,!,#,%,&.,etc.

lucky_#
@lucky_number
@lucky_#

Important tip

Python special functionalities

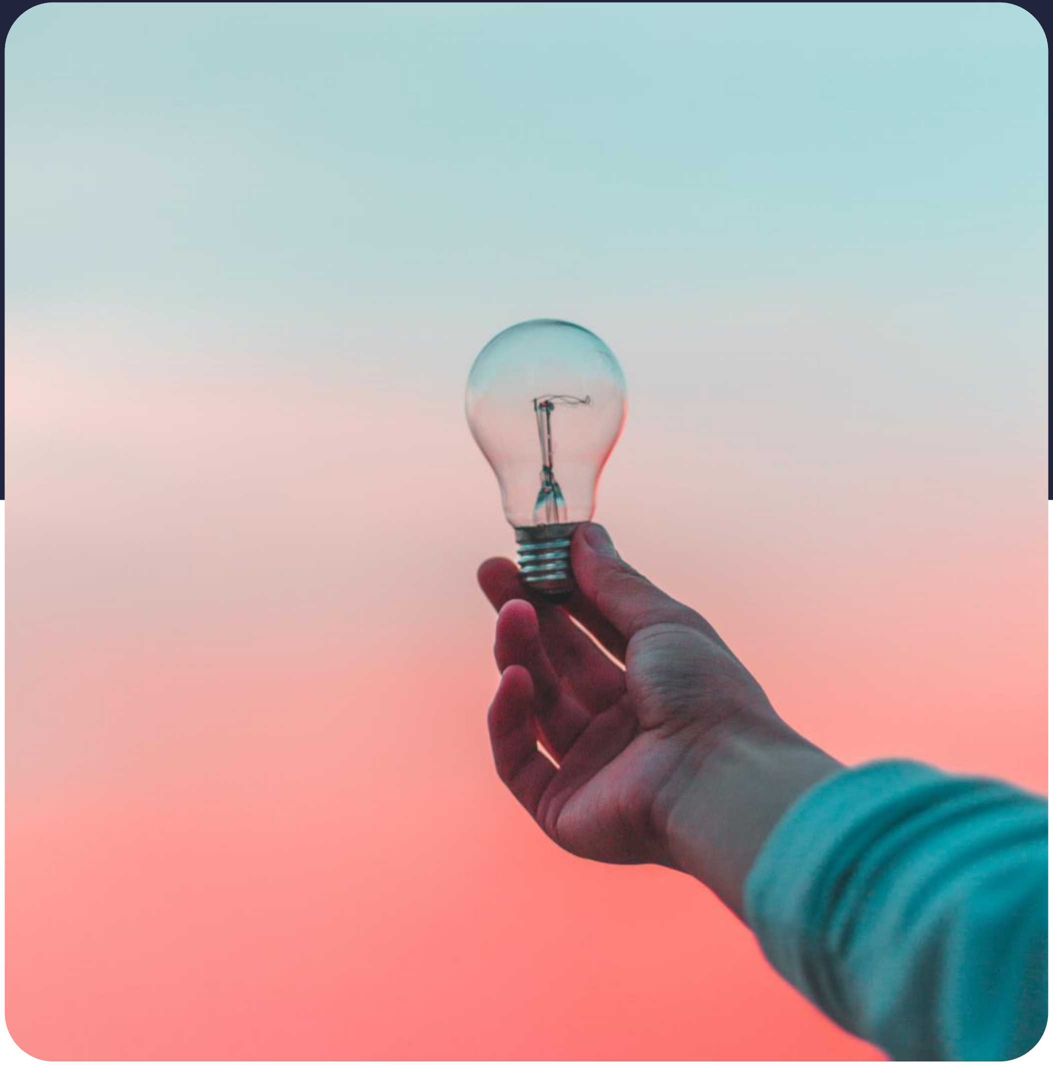


False	def	if	raise
None	del	import	return
True	elif	in	try
and	else	is	while
as	except	lambda	with
assert	finally	nonlocal	yield
break	for	not	
class	from	or	
continue	global	pass	

Object identity

```
Input                                   (158)
num                                     (159)
print(id(num))                          (160)
Output                                  (161)
94668063081792                          (162)
```

Please note that the value of this identity number will differ from program to program.



What is a Literal?

- Numeric literals
- String literals
- Boolean literals

Numeric Literals

- Integer
- Float
- Complex

Integer literal

- **Binary literal**
- **Decimal literal**
- **Octal literal**
- **Hexadecimal literal**

```
Input
binary_literal = 0b1010
decimal_literal = 100
octal_literal = 0o310
hexa_decimal = 0 × 12c
print(binary_literal)
print(decimal_literal)
print(octal_literal)
print(hexa_decimal)
Output
10
100
200
300
```

Float literal

Input

```
float_literal = 2.5
print(float_literal)
```

Output

```
2.5
```

Complex literal

```
Input                              (183)
complex_literal = 2 + 3.4j         (184)
print(complex_literal)             (185)
Output                             (186)
2 + 3.4j                           (187)
```

String Literals

Input (188)

print(^{0}Hello I am a string^{0}) (189)

Output (190)

I   am   a   string (191)

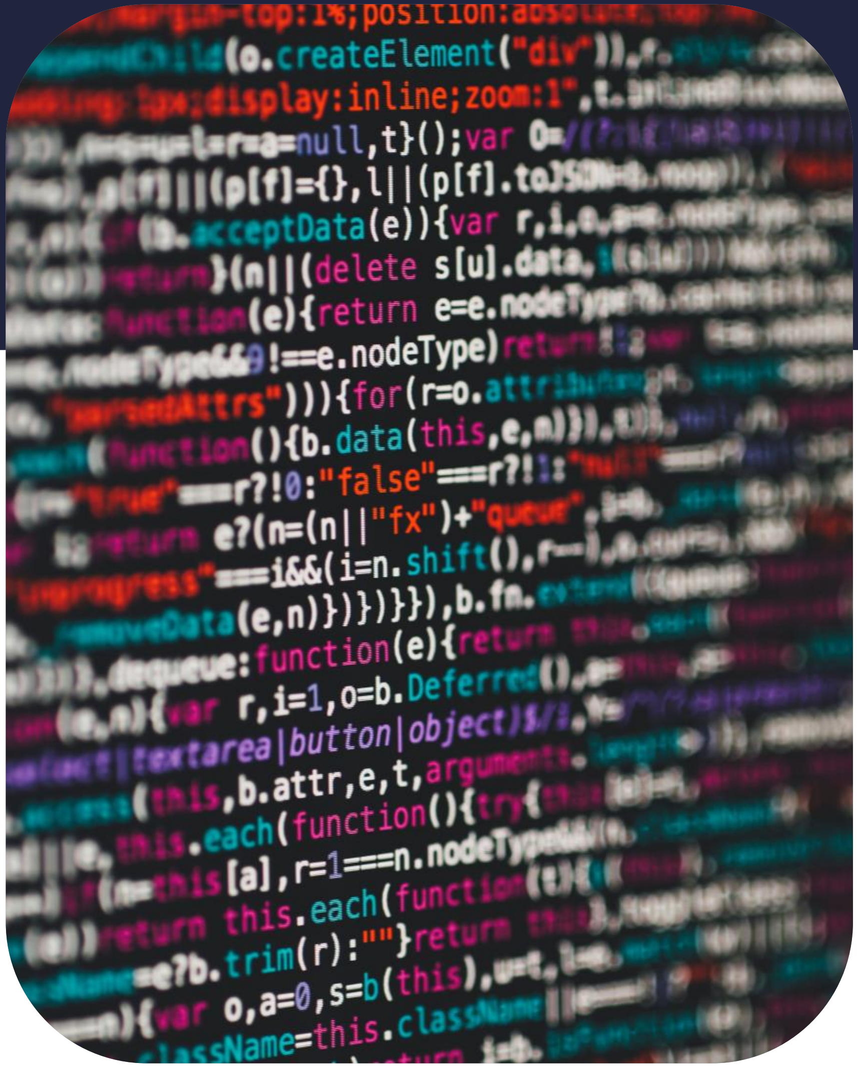
Boolean Literals

Input

```
boolean_literal = True
print(boolean_literal)
```

Output

```
True
```



Python Statement

Coming up In Lesson 3

Python Statements

A Python statement is an instruction that a Python interpreter can execute.

surname = ‘Smith⁰ (197)

Multi-line Statements

Input

```
num = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8
print(num)
```

Output

```
36
```

Multi-line Statements

```
Input                                   (203)
num = 1 + 2 + 3 + \                     (204)
          4 + 5 + 6 + \                  (205)
          7 + 8                          (206)
print(num)                              (207)
Output                                  (208)
36                                      (209)
```

Multi-line Statements

```
Input                                    (210)
num = 1 + 2 + 3 +                        (211)
            4 + 5 + 6 +                  (212)
            7 + 8                        (213)
print(num)                               (214)
Output                                   (215)
SyntaxError: invalid syntax              (216)
```

Multi-line Statements

Example 1

Parenthesis (), {}, []

```
Input
num = (1 + 2 + 3 +
           4 + 5 + 6 +
           7 + 8)
print(num)
Output
36
```

Multi-line Statements

Example 2

Input

```
num = {1 + 2 + 3 +
            4 + 5 + 6 +
            7 + 8}
print(num)
```

Output

```
36
```

Multi-line Statements

Example 3

Input

```
num = [1 + 2 + 3 +
           4 + 5 + 6 +
           7 + 8]
print(num)
```

Output

```
36
```

Python Indentation

Python uses indentation to define and write a code block.

Multi-line Statements

Input (238)

if True : (239)

Num=10 (240)

print(num) (241)

Output (242)

10 (243)

Multi-line Statements

```
Input                                (244)
if True:                             (245)
   num=10                            (246)
             print(num)              (247)
Output                               (248)
IndentationError                     (249)
```

Multi-line Statements

Input (250)

```
if True;   num=10;   print(num)
```
(251)

Output (252)

```
10
```
(253)

Multi-line Statements

Input

```
if True:
   num=10
   print(num)
```

Output

```
10
```

IndentationError

A common error when indentation is not consistent.

Python comments

Writing a comment within a code

```
Input
#This is just a comment
#This is just a comment
   num=10
   print(num)
Output
10
```

Multiple line comments

Sometimes we have comments that extend over multiple lines.

Writing a comment within a code block

Example 1

```
Input                              (267)
#First comment                     (268)
#Second comment                    (269)
#Third comment                     (270)
  num=‘Hello’                      (271)
  print(num)                       (272)
Output                             (273)
Hello                              (274)
```

Writing a comment within a code

Example 2

```
Input                          (275)
000First comment               (276)
Second comment                 (277)
Third comment000               (278)
   num="Hello00                (279)
   print(num)                  (280)
Output                         (281)
Hello                          (282)
```

Writing a comment within a code

Example 3

Input (283)

^{000000}First comment (284)

Second comment (285)

Third comment^{000000} (286)

    num=“Hello^{00} (287)

    print(num) (288)

Output (289)

Hello (290)