Basics of Python

Python Keywords

- Python keywords are special reserved words that have specific meanings and purposes and can't be used for anything but those specific purposes. These keywords are always available—you'll never have to import them into your code.
- Keywords are the reserved words in Python.
- We cannot use a keyword as a variable name, function name or 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. The list of all the keywords is given below.

List of Keywords

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

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
- 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.
- An identifier cannot start with a digit. 1variable is invalid, but variable1 is a valid name.
- Keywords cannot be used as identifiers.
- We cannot use special symbols like !, @, #, $, % etc. in our identifier.
- An identifier can be of any length.
- 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

Python Variables

- A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data that can be changed later in the program. For example-
- number = 10
- Another Example-
- website = "abc.com"
- print(website)
- # assigning a new variable to website
- website = "xyz.com"
- print(website)
- **Output**
- abc.com
- xyz.com

Constants

- A constant is a type of variable whose value cannot be changed. It is helpful to think of constants as containers that hold information which cannot be changed later. You can think of constants as a bag to store some books which cannot be replaced once placed inside the bag. Example-
- Create a constant.py:
- PI = 3.14
- GRAVITY = 9.8
- Create a main.py:
- import constant
- print(constant.PI)
- print(constant.GRAVITY)
- **Output**
- 3.14
- 9.8

Rules and Naming Convention for Variables and constants

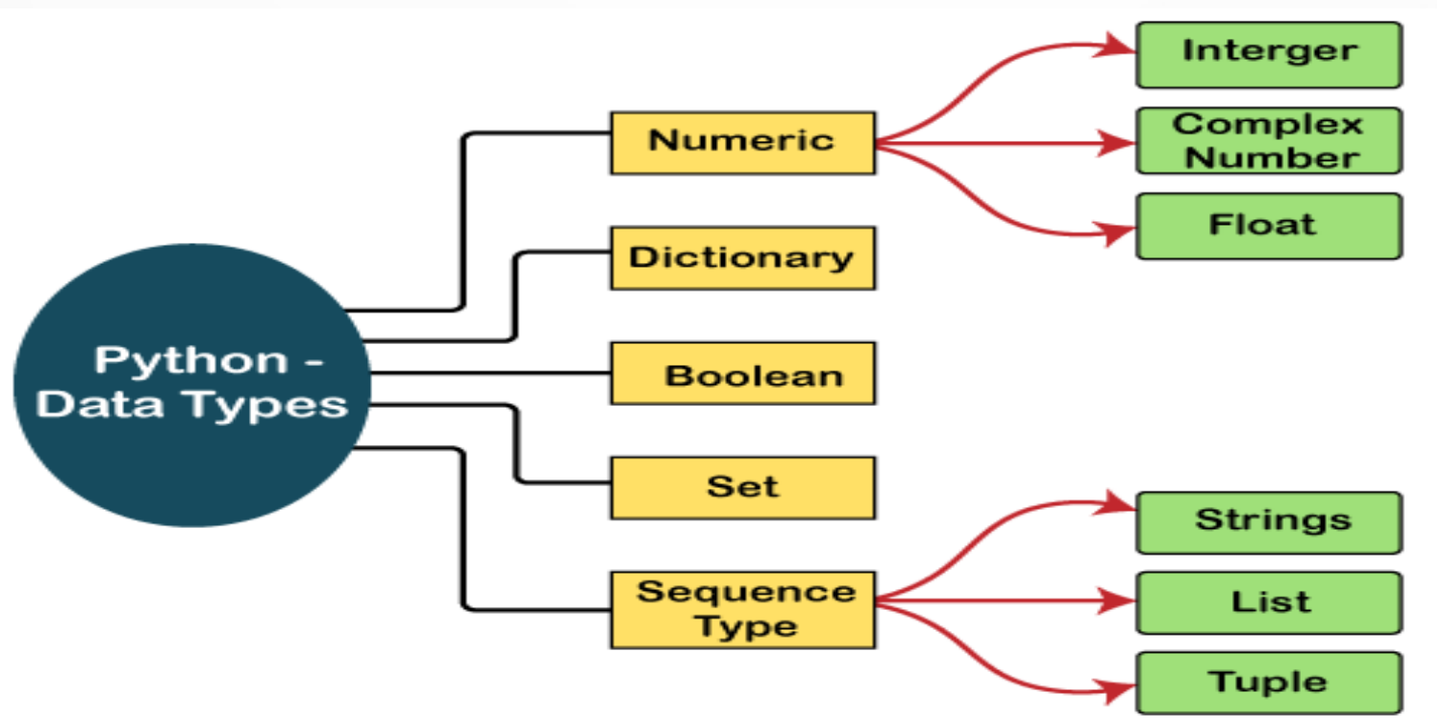
- Constant and variable names should have a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_). For example:
- first_name LAST_NAME homeAddress PinCode
- Create a name that makes sense. For example, marks makes more sense than m.
- If you want to create a variable name having two words, use underscore to separate them. For example:
- my_name current_salary
- Use capital letters possible to declare a constant. For example:
- PI MAX
- Never use special symbols like !, @, #, $, %, etc.
- Don't start a variable name with a digit.

Literals

- Numeric Literals - age=19
- String literals - name=”ram”
- Boolean literals – True, False
- Special literals - None

Python Data Types

- Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.



Python Numbers

- Integers, floating point numbers and complex numbers fall under Python numbers category. They are defined as int, float and complex classes in Python. We can use the type() function to know which class a variable or a value belongs to. Similarly, the isinstance() function is used to check if an object belongs to a particular class. Example-
- a = 5
- print(a, "is of type", type(a))
- a = 2.0
- print(a, "is of type", type(a))
- a = 1+2j
- print(a, "is complex number?", isinstance(1+2j,complex))
- **Output**
- 5 is of type <class 'int'>
- 2.0 is of type <class 'float'>
- (1+2j) is complex number? True
- Integers can be of any length, it is only limited by the memory available.
- A floating-point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. 1 is an integer, 1.0 is a floating-point number.
- Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part.

Python List

- List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type.
- Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ].
- a = [1, 2.2, 'python']
- We can use the slicing operator [ ] to extract an item or a range of items from a list. The index starts from 0 in Python.
- a = [5,10,15,20,25,30,35,40]
- print("a[2] = ", a[2])
- print("a[0:3] = ", a[0:3])
- print("a[5:] = ", a[5:])
- **Output**
- a[2] = 15
- a[0:3] = [5, 10, 15]
- a[5:] = [30, 35, 40]

Python Tuple

- Tuple is an ordered sequence of items same as a list. The only difference is that tuples are immutable. Tuples once created cannot be modified. Tuples are used to write-protect data and are usually faster than lists as they cannot change dynamically. It is defined within parentheses () where items are separated by commas. We can use the slicing operator [] to extract items but we cannot change its value.
- t = (5,'program', 1+3j)
- print("t[1] = ", t[1])
- print("t[0:3] = ", t[0:3])
- # Generates error Tuples are immutable
- t[0] = 10
- **Output**
- t[1] = program
- t[0:3] = (5, 'program', (1+3j))
- Traceback (most recent call last):
- File "test.py", line 11, in <module>
- t[0] = 10
- TypeError: 'tuple' object does not support item assignment

Python Strings

- String is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """.
- s = "This is a string"
- print(s)
- s = '''A multiline
- string'''
- print(s)
- **Output**
- This is a string
- A multiline
- string
- Just like a list and tuple, the slicing operator [ ] can be used with strings. Strings, however, are immutable.
- s = 'Hello world!'
- print("s[4] = ", s[4])
- print("s[6:11] = ", s[6:11])
- # Generates error Strings are immutable in Python
- s[5] ='d'
- **Output**
- s[4] = o
- s[6:11] = world
- Traceback (most recent call last):
- File "<string>", line 11, in <module>
- TypeError: 'str' object does not support item assignment

Python Set

- Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.
- a = {5,2,3,1,4}
- # printing set variable
- print("a = ", a)
- # data type of variable a
- print(type(a))
- **Output**
- a = {1, 2, 3, 4, 5}
- <class 'set'>
- We can perform set operations like union, intersection on two sets. Sets have unique values. They eliminate duplicates.
- a = {1,2,2,3,3,3}
- print(a)
- **Output**
- {1, 2, 3}
- Since, set are unordered collection, indexing has no meaning. Hence, the slicing operator [] does not work.

Python Dictionary

- Dictionary is an unordered collection of key-value pairs. It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value. In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type
- >>> d = {1:'value','key':2}
- >>> type(d)
- <class 'dict'>
- We use key to retrieve the respective value. But not the other way around.
- d = {1:'value','key':2}
- print(type(d))
- print("d[1] = ", d[1]);
- print("d['key'] = ", d['key']);
- # Generates error
- print("d[2] = ", d[2]);
- **Output**
- <class 'dict'>
- d[1] = value
- d['key'] = 2
- Traceback (most recent call last):
- File "<string>", line 9, in <module>
- KeyError: 2

Python Operators

- Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.
- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Arithmetic operators

Operator	Meaning	Example
+	Add two operands or unary plus	x + y+ 2
-	Subtract right operand from the left or unary minus	x - y- 2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

Assignment operators

Operator	Example	Equivalent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

Comparison operators

Operator	Meaning	Example
>	Greater than - True if left operand is greater than the right	x > y
<	Less than - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

Logical operators

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Identity & membership operators

- Identity Operators

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

- Membership Operators

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Bitwise operators

Operator	Meaning	Example
&	Bitwise AND	x & y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x >> 2 = 2 (0000 0010)
<<	Bitwise left shift	x << 2 = 40 (0010 1000)

Thank You