Fundamentals of Programming:

Programming for Data Science with Python

1. Identifiers

Overview

- Identifiers are the names that identify the elements of a program such as classes, methods, variables, constants, etc.
  - Identifiers can be variables or constants in a program.
  - Identifiers or names are case sensitive.
  - They can contain letters, digits (or numbers), and underscore.
  - However, they cannot start with digits or numbers.

IMPORTANT NOTES:

- Reserved words cannot be used as identifiers.
- Here is the list of some reserved words in Python:

> and, assert, break, class, continue, def, del, elid, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while

Run the following code:

In [5]:
```
# x is a name of a variable. x is an identifier.
x = 3

print ("x is a variable. It is an identifier. Its value is: ",x, "\n")
print("Data type of x: ", type (x), '\n')

#stdName is a name of a variable that represents the name of a student.
#stdName is an identifier

stdName = "John Smith"

print ("stdName is a variable. It is an identifier. Its value is: ", stdName, "\n")
print("Data type of stdName: ", type (stdName), '\n')
```

```
x is a variable. It is an identifier. Its value is:  3

Data type of x:  <class 'int'>

stdName is a variable. It is an identifier. Its value is:  John Smith

Data type of stdName:  <class 'str'>
```

1.1 Variables: Data - Data Containers - Data Types

Varaibles are data containers whose values are likely to be changed along the course of executing a program.

Variable/Identifier x:

- x is an identifier. It is a ***variable.***
- its value is 3: 3 is ***data***
- The variable (identifier) x is the ***data container*** that contains the piece of data "3".
- ***Data type*** of x is "int"

Run the following code:

In [6]:
```
# x is a name of a variable. x is an identifier.
x = 3
print ("x is a variable. It is an identifier. Its value is: ", x, "\n")
print("Data type of x: ", type(x), '\n')
```

```
x is a variable. It is an identifier. Its value is:  3

Data type of x:  <class 'int'>
```

Variable/Identifier stdName:

- ***stdName*** is an ***identifier***. It is a ***variable***.
- Its ***value*** is "john Smith": "JohN Smith" is ***data***.
- The ***variable (identifier)*** stdName is the ***data container*** that contains the piece of data "John Smith".
- ***Data*** type of stdName is "str" (or String)

IMPORTANT NOTES:

- In Python, identifiers are unlimited in length. Case is significant.
- However, the user is strongly discouraged from using a very long name to label variables. Besides, the names should be meaningful.

Run the following code:

In [7]:
```
#stdName is a name of a variable that represents the name of a student.
#stdName is an identifier
std = "John Smith"
print ("stdName is a variable. It is an identifier. Its value is: ", std, "\n")
print("Data type of stdName: ", type (std), '\n')
```

```
stdName is a variable. It is an identifier. Its value is:  John Smith

Data type of stdName:  <class 'str'>
```

1.2. Constants

Constants are data containers that contain permanent values, i.e., these values cannot be changed, along the course of executing a program.

2. Assignments

Overview

Binding a variable in Python means setting a name to hold a ***reference*** to some object.

- Assignments creates references, not copies.

  Names in Python do not have an intrinsic type. ***Objects have types***.

- Python determines the ***type of the reference*** automatically based on the ***data object*** assigned to it.

  A name is created when it appears the first time on the left side of an assignment expression:

  > x = 3

  A ***reference is deleted*** via garbage collection when NO variables or identifiers are referring to it.

2.1. Reference Sematic in Python

Overview: The process of assigning a value

What happens when we type: x = 3?

- First, an integer 3 is created and stored in memory
- Then a name is created
- Next, a reference to the memory location storing the 3 is assigned to the name x

2.2 Assignment manipulates references

- y = x: An assignment of x to y
- y = x: The assignemnt does not make a copy of the value of x.
- y = x: The assignment makes y refer to the same object as x does.

> Let's say you assign x = 3
>
> And then you assign y = x

When you assign y = x, you now have y = 3 since x = 3 (Run the code below.)

Run the following code:

In [10]:
```
x=3

y=x

y
```

Out[10]: 3

3. Operators and Operations

3.1 Numeric Operations

Operator	Operation	Example	Result
+	Addition	33 + 3	36
-	Subtraction	33 - 3	30
*	Multiplication	33 * 3	99
/	Division	11 / 3	3.666
%	Remainder	33 % 3	0
**	Exponent	33 ** 3	35937
//	Floor	11 // 3	3

3.2 Augmented Operators

Operator	Operation	Example	Result
+=	Addition assignment	i += 8	i = i + 8
-=	Subtraction assignment	i -= 8	i = i - 8
*=	Multiplication assignment	i *= 8	i = i * 8
/=	Division assignment	i /= 8	i = i / 8
%=	Remainder assignment	i %= 8	i = i % 8
**=	Exponent assignment	i **= 8	i = i ** 8
//=	Floor assignment	i //= 8	i = i // 8

3.3 Relational Operators (a.k.a. Comparison Operators)

- Relational operators, a.k.a. comparison operators, are used for comparisons.
- A comparison, e.g. a>b, is called a conditional expression or boolean expression.
- The result of a boolean expression is either True or False.

Operator	Meaning	Example	Ex. Result
(<)	Less than	radius (<) 0	False
(=<)	Less than or equal to	radius (<=) 0	True
(>)	Greater than	radius (>) 0	True
(>=)	Greater than or equal to	radius (>=) 0	False
(==)	Equal to	radius (==) 0	True

Operator	Meaning	Example	Ex. Result
(!=)	Not equal to	radius (!=) 0	False

3.4 Logical Operators

Operator	Meaning	Example
not	Opposite	not (radius < 0)
and	And	(a > b) or (c < d)
or	Or	(a > b) or (c < d)

3.4.1 not Operators

X	not X
True	False
False	True

3.4.2 and Operators

a and b ***is true*** only when ***both a and b are true***.

a	b	a and b
True	True	True
True	False	False
False	True	False
False	False	False

3.4.2 or Operators

a or b ***is false*** only when ***both a and b are false***.

a	b	a or b
True	True	True
True	False	True
False	True	True
False	False	False

3.5 Identity Operators

In Python, ***identity operators*** are used to check if the ***operands are identical***, i.e., they refer to the same object.

Operator	Result	Example
is	True if the operands are identical, i.e., they refer to the same object;	x is y
is not	True if the operands are not identical, i.e., they do not refer to the same object	x is not y

Run the following 3 code blocks:

In [5]:
```
x = 5
y = 5

isTrue = "x is y"
isFalse = "x is not y"

if (x is y):
    print (isTrue)
else:
        print (isFalse)
```

```
x is y
```

In [6]:
```
x = 5
y = 8

isTrue = "x is y"
isFalse = "x is not y"

if (x is y):
    print (isTrue)

else:
    print (isFalse)
```

```
x is not y
```

In [7]:
```
x = 5
y = 5

isTrue = "x is y"
isFalse = "x is not y"

if (x is not y):
    print (isTrue)
else:
        print (isFalse)
```

```
x is not y
```

3.6 Membership Operators

In Python, many data structures have their internal structure of a sequence, e.g., String, List, Tuple, etc.

For example:

- aString = "Hello World"
- aList = [1,2,3,4,5]
- aTuple = (1, 2, 3, 4, 5)

Membership operators - in, not in - are used to test ***if a value is found*** in a sequence or not.

Operator	Result	Example
in	True if the value/variable is found in the sequence;	x in y
not in	True if the value/variable is not found in the sequence	x not in y

****Run the following 2 code blocks:****

In [8]:
```
x = 'Hello World'
if ('H' in x):
    print ("H in x")
else:
        print ("H not in x")
```

```
H in x
```

In [3]:
```
# You will get an error but think why you have an error.
aList = [1, 2, 3, 4, 5]
if (8 not in aList):
    print ("8 not in aList")
else:
        print ("8 in aList")
```

```
8 not in aList
```

5. Pseudo-Code

5.1 Scenario: A Problem

It is assumed that a software developer is asked to write a Python program that can calculate and print the diameter and the circumference of a circle. The user enters the data of the radius and its measurement unit (in, ft, cm, or m) from the console.

5.2 How to Solve the problem

- First, we need to write pseudo-code (or create a flow-chart) of steps that can solve the problem. Formally, we design an algorithm that offers a solution to the problem.
- Then, based on the steps of the pseudo-code/flowchart (or algorithm), we write the code of the program.

These two step are the foundation of software engineering process.

5.3 Pseudo-Code

With pseudo-code, the developer writes down the steps to solve the problem in plain English.

1. Start
2. Read the input of the radius from the console
3. Read the measurement unit of the radius (in, ft, cm, m)
4. Calcualte the diameter of the circle
   - diameter = 2*radius
5. Calcualte the circumference of the circle
   - Circumference = diameter*PI (3.14159)
6. Print out the diameter
7. Print out the circumference
8. End

6. Comments

In Python, comments begin with a hash mark (#), a whitespace character and continue to the end of the line.

Run the following 4 code blocks:

In [4]:
```
# This is a comment
x = 3
x
```

Out[4]: 3

In [11]:
```
# Print "Hello World!" to console
print ("Hello, World")
```

```
Hello, World
```

In [12]:
```
# Inline comment
x = 3 # This is an inline comment
x
```

Out[12]: 3

In [13]:
```
# This is a comment of multiple lines, also known as a block comment
# Add another line of comment
# Another line
# And another line
x = 3
x
```

Out[13]: 3

In [ ]: