Chapter 2: Data Types and Operators

Contents

Syllabus Learning Outcomes

2.2 Programming Elements and Constructs

Use programming language elements and constructs to write recursive and non-recursive programs to solve a variety of problems.

- 2.2.1 Understand the different types: integer, real, char, string and Boolean and initialise arrays (1- dimensional and 2-dimensional).
- 2.2.2 Use common library functions for input/output, strings and mathematical operations.

3.1 Data Representation

Understand that values can be represented in different number bases: denary, binary and hexadecimal.

- 3.1.1 Represent data in binary and hexadecimal forms.
- 3.1.2 Write programs to perform the conversion of positive integers between different number bases: denary, binary and hexadecimal forms; and display results.

3.2 Character Encoding

Understand the use of ASCII code and Unicode to represent characters.

- 3.2.1 Give examples of where or how Unicode is used.
- 3.2.2 Use ASCII code in programs.

1 Data Types

Multiple data types are developed for convenient data manipulation and management. In the table below are some of the more common data types:

Data Types	Description	Example
Integer	A signed whole number	1, 1234, -15
Float (real)	A signed number with a decimal point	1.0, 2.345, - 54.321
Boolean	The logical values TRUE and FALSE	TRUE, FALSE
String	A sequence of zero or more characters	'hello', 'action 123', 'A-level', ''

The data type is set when you assign a value to a variable. The `type()` function can be used to check the data type of a variable.

```
1  # Example on integer and float (real numbers)
2  num1 = 123
3  num2 = 12.3
4
5  print(type(num1), type(num2))
```

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

```
1  # Example on char and string
2  check = True
3  string1 = 'hello!'
4
5  print(type(check), type(string))
```

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

1.1 Type conversion

We can convert one type of number into another. Operations like addition, subtraction can convert integer to float implicitly (automatically), if one of the operands is float.

```
1  print(1 + 2.0)
```

```
3.0
```

We can also use built-in functions like `int()`, `float()` and `str()` to convert between types explicitly.

```
# Example of converting float to int and vice versa
num1 = 2.3
num2 =-2.8
num3 = 5

print(type(num1), type(num2), type(num3))

print(type(int(num1)), type(int(num2)), type(float(num3)))

# Example of converting int to str

s = str(num2)
print(type(s))

```

```
<class 'float'> <class 'float'> <class 'int'>
<class 'int'> <class 'int'> <class 'float'>
<class 'str'>
```

1.2 Input

We can ask user for input using Python built-in `input()` function.

- When `input()` function is called, the program will stop and wait for user to key in data.
- It is optional to add prompt as function parameter.
- It always returns a string value.

```
# Example of using input()

age = input("Enter your age: ")
print(type(age))
```

```
Enter your age: 5
<class 'str'>
```

Notice that even when the value entered by user is an integer, the returned value will still be of the `str` type. We need to convert the type of the returned value using the built-in functions for further manipulation of the returned value if required.

```
# Example of using input()

age = input("Enter your age: ")
print(type(int(age)))
```

```
Enter your age: 5
<class 'int'>
```

1.3 Output

Python provides a built-in function `print()` to display data to the console or a file object.

The `print()` function

- can accept multiple values, and uses sep value to separate them in output. By default, the sep value is a space,
- automatically appends an end string at the end. By default, the end value is a new line (\n), and
- outputs data to a file object defined in file, which has a default value sys.stdout (console).

```
1  # Examples of different output using print()
2  print('hello', 'world')
3  print()
4  print('have', 'a', 'good', 'day', sep='-')
```

```
hello world

have-a-good-day
```

```
1  # output is printed in a new line
2  print("Hello")
3  print("World!")
4
5  print()
6
7  # append a space at the end instead of a new line
8  print("Hello", end = " ")
9  print("World!")
```

```
Hello
World!

Hello World!
```

2 Numbers

2.1 Integer in different bases

The numbers we deal with every day are of the decimal (base 10) number system. However, computer programmers (generally embedded programmers) need to work with binary (base 2), hexadecimal (base 16) and octal (base 8) number systems.

In Python, we can represent these numbers by appropriately placing a prefix before that number. The following table lists these prefixes.

Number System	Prefix
Binary	'0b' or '0B'
Octal	'0o' or '0O'
Hexadecimal	'0x' or '0X'

```
# Example of binary numbers
print(0b1101011)

# Example of addition of hexadecimal and binary numbers
print(0xFB + 0b10)

# Example of octal numbers
print(0o15)
```

```
107
253
13
```

We can use `bin()` and `hex()` to convert a decimal number to a binary and hexadecimal respectively. We can also use `int()` to convert other number systems to decimals.

```
x = 100

# convert decimal to binary
print(bin(x))

# convert decimal to hexadecimal
print(hex(x))

y = '100'
# convert binary to decimal
print(int(y,2))

# convert hexadecimal to decimal
print(int(y,16))
```

```
0b1100100
0x64
4
256
```

2.2 Arithmetic Operators

Arithmetic operators perform common mathematical operations on values.

Operator	Name	Expression	Example	Answer
+	Addition	x + y	2+3	5
-	Subtraction	x - y	7-3	4
*	Multiplication	x * y	2*3	6
/	Division	x / y	20/8	2.5
//	Integer division	x // y	20//8	2
%	Modulus	x % y	20%8	4
**	Exponentiation	x ** y	4**2	16

```
# Examples of using arithmetic operators
x = 8
y = 4

# Addition
a = x+y
print(a)
print(type(a))

# Subtraction
print(x-y)

# Multiplication
print(x-y)

# Division
b=x/y
print(b)
print(type(b))

# Floor Division (returns quotient without remainder)
print(x//y)

# Modulus (Remainder)
print(x%y)
```

```
26
27 # Exponentiation (x to the power of y)
28 print(x**y)
29
```

```
12
<class 'int'>
4
4
2.0
<class 'float'>
2
0
4096
```

Sometimes we want to do more than just usual arithmetic operation. Python has a built-in function that gives us access to more functions. (Refer to annex on how to import a module)

```
1 # Example of importing math module
2
3 import math
4
5 print(math.pi)
```

```
3.141592653589793
```

3 Boolean

A Boolean variable can represent either `True` or `False`.

3.1 Boolean operators

Comparison operators are used to compare two values. It returns a boolean value.

Operator	Name	Example
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y
==	Equal	x == y
!=	Not equal	x != y

```
# Example on boolean types
print(type(True), type(False))

print(5 > 4)
print(5 < 4)
print(5 == 4)
print(5 != 4)
```

```
<class 'bool'> <class 'bool'>
True
False
False
True
```

An object can also be evaluated to be True or False using built-in `bool()` function.

```
print(bool(2))
print(bool(-2))
print(bool(0.2))
print(bool(0))
print(bool('Hi'))
print(bool(''))
```

```
True
True
True
False
True
False
```

3.2 Logical Operators

Sometimes we need to make decisions based on multiple conditions. Logical operators are used to combine conditional statements.

- Operands shall be conditions which can result in a boolean value.
- The outcome of such an operation is either true or false too.

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if **at least** one of the statements is true	x < 5 or x < 10
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Given that x = 6, predict the outcome in the "Example" column from the table above.

4 Strings

Characters in programming are not limited to the 26 English characters. It can also be space character, question mark, Chinese character, etc.

A string is a sequence of characters. It can be a word, a sentence or a paragraph.

4.1 ASCII code and Unicode

Today's programs need to be able to handle a wide variety of characters. Applications are often internationalized to display messages and output in a variety of user-selectable languages; the same program might need to output an error message in English, French, Japanese, Hebrew, or Russian. Web content can be written in any of these languages and can also include a variety of emoji symbols.

Two standard character sets used in displaying text are ASCII code and Unicode.

ASCII stands for American Standard Code for Information Interchange The ASCII character set is a 7-bit set of codes that allows 128 different characters. That is enough for every upper-case letter, lower-case letter, digit and punctuation mark on most keyboards. ASCII is only used for the English language.

This table shows some examples of letters represented using the ASCII character set:

Character	Denary value	Binary value	Hex
N	78	1001110	4E
O	79	1001111	4F
P	80	1010000	50
Q	81	1010001	51
R	82	1010010	52

Extended ASCII code is an 8-bit character set that represents 256 different characters, making it possible to use characters such as é or ©. Extended ASCII is useful for European languages.

Unicode uses between 8 and 32 bits per character (UTF-8, UTF-16 and UTF-32), so it can represent characters from languages from all around the world.

```
1  # Example of using unicode
2  print('\u7434')
```

琴

It is commonly used across the internet. As it is larger than ASCII, it might take up more storage space when saving documents. Global companies, like Facebook and Google, would not use the ASCII character set because their users communicate in many different languages.

We can get the ASCII code of a character by using the build-in function `ord()`. We can also retrieve the character from an ASCII code by using `chr()`.

```
# Geting ASCII code of character
ascii_code = ord("A")

# Retrieve character from ASCII code
character = chr(ascii_code)

print(ascii_code)
print(character)
```

```
65
A
```

4.2 Create Strings

String is one of the most commonly used data type in Python. Strings in Python are objects.

There are 3 ways to declare strings. Strings can be declared in either single-quote, double-quotes or triple double-quotes.

```
x = 'hi'
y = "hello"
z = """world"""
```

```
# Using single quotes
m = 'I\'m here'

# Using single and double quotes
n = "You're there"

print(m)
print(n)
```

```
I'm here
You're there
```

Triple double-quotes are commonly used to initialize multi-line strings.

```
z = """How
are
you?"""
print(z)
```

```
How
are
you?
```

Triple quotes are also used to create docstring, which we will go into more detail when we learn functions.

4.3 Escape character

In Python strings, the backslash "\" is a special character, also called the "escape" character. It is used in representing certain whitespace characters: "\t" is a tab, "\n" is a newline.

```
print('apple\torange') # tab between apple and orange
print('apple\norange') # print on a new line after apple
```

```
apple   orange
apple
orange
```

Conversely, prefixing a special character with "\" turns it into an ordinary character. This is called "escaping". For example, "\'" is the single quote character.

```
# Both print statements give the same output
print('It\'s raining')
print("It's raining")
```

```
It's raining
It's raining
```

Likewise, "" can be escaped: "\"hello\"" is a string begins and ends with the literal double quote character. Finally, "\" can be used to escape itself: "\\" is the literal backslash character.

```
print("\"hello\"")
print('"\\" is the backslash')
```

```
"hello"
"\" is the backslash
```

The escape character can also simply be used to break a long line of code.

```
if 5 > 2 and 2 < 8 and \
3 < 5:
    print("This is \
correct!")
```

```
This is correct!
```

4.4 Indexing

Each character in a string can be accessed by its index (location).

In Python, index value starts from 0. Python also supports negative indexes. Index of '-1' represents the last character of the String.

```
x = "spiderman"
print(x[0]) # refers to first character of string
print(x[1]) # refers to second character of string
print(x[-1]) # refers to last character of string
print(x[-2]) # refers to second last character of string
```

```
s
p
n
a
```

The `len()` function is one useful method that returns the number of items in an object. When the object is a string, the `len()` function returns the number of characters in the string.

```
x = "spiderman"

print(len(x)) # prints the number of characters in the string x
```

```
9
```

The function `len()` can also be used in indexing.

```
# different ways to print out the last characteer in a string
print(x[len(x)-1]) # refers to last character of string, x[len(x)-1] same as x[9-1]
print(x[-1])
print(x[8])
```

```
n
n
n
```

What will be the output for `print(x[20])`?

4.5 Slicing

Slicing is to retrieve a range of characters in a String.

The x[a:b] returns all characters in x from index a to b-1.

- Note: Character at index 'b' is not included in output.

The x[:b] returns all characters before index b.

- Start-index will be set to 0 if it is ommited.
- Character at index 'b' is not included.

The x[a:] returns all characters from index a.

- End-index will be length of x if it is omitted.
- Character at index 'a' is included.

```
# Examples on slicing
x = "spiderman"

print(x[0:3]) # print characters from index 0 to 2
print(x[:3]) # print characters from index 0 to 2
print(x[3:]) # print characters from index 3 to last character
print(x[3:len(x)]) # print characters from index 3 to last character
print(x[2:5]) # print charactes from index 2 to 4
```

```
spi
spi
derman
derman
ide
```

What will be the output for `print(x[:20]), print(x[1:-1])` and `print(x[::-1])`?

4.6 Immutable

An immutable object cannot be changed after it is created.

String objects are immutable!

Do you think I can change the letter 'r' in the String "spiderman" to capital letter by doing `x[0] = 'R'`? Test it out.

4.7 String operators

Operator	Name	Expression
+	Concatenation	x + y
*	Repetition	x * y
in	Exists	x in y
not in	Not Exists	x not in y

```
# Concatenation
s1 = "spider"
s2 = "man"
print(s1 + s2)

# Repetition
s = "Ha"
print(s * 4)

# Returns True if a string is present in another string, otherwise returns False
print("spider" in s1)

# Returns True if a string is NOT present in another string, otherwise returns False
print("bat" in s1)
```

```
spiderman
HaHaHaHa
True
False
```

4.8 Built-in String methods

String object comes with a number of built-in methods to manipulate strings. All string methods returns new values. They do not change the original string.

Method	Description
count()	Counts how many times str occurs in string or in a substring of string
find()	Determine if str occurs in string or in a substring of string, returns index if found and -1 otherwise.
Index()	Same as find(), but raises an exception if str not found.
lower()	Converts all letters in string to lowercase.
upper()	Converts all letters in string to uppercase.
title()	Converts the first character of each word to upper case
strip()	Remove white space characters from the beginning or the end by default. One or more characters can be specified to be stripped.
lstrip()	Removes all leading whitespace in string.
rstrip()	Removes all trailing whitespace of string.
isalnum()	Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
isalpha()	Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.
isdigit()	Returns true if string contains only digits and false otherwise.
islower()	Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
isnumeric()	Returns true if a unicode string contains only numeric characters and false otherwise.
isupper()	Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.
isspace()	Returns true if string contains only whitespace characters and false otherwise.
startswith()	Returns True if the string starts with the specified value
endswith()	Returns true if the string ends with the specified value

String formatting

The format() method formats the specified value(s) and insert them inside the string's placeholder.

The placeholder is defined using curly brackets: {}. The format() method returns the formatted string. It is the recommended string formatting tool. It automatically handle data conversion for some data types.

```
# Example on format
print('Hi {}'.format('Peter')) # the placeholder{} will be replaced by 'peter'
```

```
Hi Peter
```

The format() method takes unlimited number of arguments, and are placed into the respective placeholders.

```
name = 'Mark'
age = 18
print('{} is {} years old'.format(name,age))
```

```
Mark is 18 years old
```

Index numbers can be used, for eg. {0}, to be sure the arguments are placed in the correct placeholders. The same index number can be used more than once.

```
print('His name is {1}. {1} is {0} years old'.format(age, name)) #positioning
```

```
His name is Mark. Mark is 18 years old
```

The format() method can also be used to format numbers.

```
print(type('{:0>6}'.format(123333333333))) # format a number into a fixed number of digits with zeros appended to the front
print(type('{:x}'.format(10))) # converts an integer into hexadecimal
print('{0:o}'.format(10)) # convers an integer into an octal number
print('{0:b}'.format(10)) # converts an integer into a binary number
print('{:.3f}'.format(3.1415)) # format a number to be displayed as a number with specific number of decimal places
print('{:.3}'.format(3.1415)) # format a number to be displayed as a number with specific number of significant figures

print('{:10.3}'.format(3.1415)) # format a number to be displayed with 10 column width, right align by default
print('{:<10.3}'.format(3.1415)) # format a number to be displayed with 10 column width and left align
print('{:^10.3}'.format(3.1415)) # format a number to be displayed with 10 column width and center
```

```
<class 'str'>
<class 'str'>
12
1010
3.142
3.14
      3.14
3.14
   3.14
```

Annex

Import a module

A module is a Python program that ends with .py extension and a folder that contains a module becomes a package.

You can consider a module to be the same as a code library whereby it contains functions and variables that you want to include in your application.

There are many Python modules available for use. For eg, random, time, math and many more. Alternatively, we may also create our own module by simply saving the code we want to reuse elsewhere in a file with the .py extension; its filename will be used as the module name.

We can use a module by using the `import` statement and we use '.' to access the variables and/or functions.

```
#import a module using the import keyword
#followed by the module name
import math

#print the pi variable stored in math module
print(math.pi)
```

```
3.141592653589793
```

If we need to use only one specific function/variable in a module, we can import the required function/variable as shown below:

```
from math import pi

print(pi)
```

```
3.141592653589793
```

Sometimes, the module name may be very long and troublesome to type repeatedly each time we use it. In these cases, we can rename our module when importing it by using the `import` and `as` keywords:

```
#rename math as m
import math as m

print(m.pi)
```

```
3.141592653589793
```