Strings

- ➔ data type consists of a set of values and a set of operations that can be performed on those values.
- ➔ Literal is the way a value of a data type looks to a programmer.

Type of Data	Python Type Name	Example Literals
Integers	`int`	`-1, 0, 1, 2`
Real numbers	`float`	`-0.55, .3333, 3.14, 6.0`
Character strings	`str`	`"Hi", "", 'A', "66"`

String Literals

- In Python, a string literal is a sequence of characters enclosed in single or double quotation marks.

```
>>> 'Hello there!'
'Hello there!'
>>> "Hello there!"
'Hello there!'
>>> ''
''
>>> """
''
```

empty string

String Literals

```
>>> print("""This very long sentence extends
all the way to the next line.""")
This very long sentence extends
all the way to the next line.
```

Note that the first line in the output ends exactly where the first line ends in the code.

When you evaluate a string in the Python shell without the **print** function, you can see the literal for the **newline character**, **\n**, embedded in the result, as follows:

```
>>> """This very long sentence extends
all the way to the next line."""
'This very long sentence extends\nall the way to the next line.'
```

Escape Sequences

Escape sequences are the way Python expresses special characters, such as the tab, the newline etc.

Escape Sequence	Meaning
\b	Backspace
\n	Newline
\t	Horizontal tab
\\	The \ character
\'	Single quotation mark
\"	Double quotation mark

String Concatenation

- You can join two or more strings to form a new string using the concatenation operator + .

```
>>> "Hi " + "there, " + "Ken!"
```

- The * operator allows you to build a string by repeating another string a given number of times. The left operand is a string, and the right operand is an integer.

“Hai”*5

Variables and the Assignment Statement

➔ variable are names associated with a value.

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

➔

Variables and the Assignment Statement

- Assignment statement : Variables receive their initial values and can be reset to new values with an assignment statement .
  - <variable name> = <expression>
- The Python interpreter first evaluates the expression on the right side of the assignment symbol and then binds the variable name on the left side to this value.
- Value assigned to the variable name for the first time, is called defining or initializing the variable.
- Subsequent uses of the variable name in expressions are known as variable references .

Variables and the Assignment Statement

```
>>> firstName = "Ken"
>>> secondName = "Lambert"
>>> fullName = firstName + " " + secondName
```

Program Comments and Docstrings

- ➔ comment is a piece of program text that the computer ignores but that provides useful documentation to programmers.
- ➔ Docstring: a brief statement about the program's purpose at the beginning of the program file.

```
"""
Program: circle.py
Author: Ken Lambert
Last date modified: 10/10/17

The purpose of this program is to compute the area of a
circle. The input is an integer or floating-point number
representing the radius of the circle. The output is a
floating-point number labeled as the area of the circle.
"""
```

Program Comments and Docstrings

➔ Program Comments :

- end-of-line comments can document a program.
- These comments begin with the # symbol and extend to the end of a line.

```
>>> RATE = 0.85 # Conversion rate for Canadian to US dollars
```

Numeric Data Types and Character Sets

- Integers:
  - include 0, the positive whole numbers, and the negative whole numbers.
  - Python integer is much larger and is limited only by the memory of the computer.
- Floating-Point Numbers:
  - to represent real numbers.
  - Values of the most common implementation of Python's float type range from approximately 210 308 to 10 308 and have 16 digits of precision.
- Character Sets:
  - Character set is the set of valid characters that a language can recognize.
  - the characters in a string each map to an integer value. This mapping is defined in character sets, is the ASCII set and the Unicode set .
  - Python's ord and chr functions convert characters to their numeric ASCII codes and back again.

```
>>> ord('a')
97
>>> ord('A')
65
>>> chr(65)
'A'
>>> chr(66)
'B'
```

Expressions

- **Arithmetic Expressions:** An arithmetic expression consists of operands and operators combined in a manner that follows rules of algebra.

Operator	Meaning	Syntax
-	Negation	-a
**	Exponentiation	a ** b
*	Multiplication	a * b
/	Division	a / b
//	Quotient	a // b
%	Remainder or modulus	a % b
+	Addition	a + b
-	Subtraction	a - b

Mixed-Mode Arithmetic and Type Conversions

- Performing calculations involving both integers and floating-point numbers is called mixed-n

```
>>> 3.14 * 3 ** 2
28.26
```

-
- In a binary operation on operands of different numeric types, the less general type (int) is temporarily and automatically converted to the more general type (float) before the operation is performed.
- in the example expression, the value 9 is converted to 9.0 before the multiplication.

type conversion function

A type conversion function is a function with the same name as the data type to which it converts.

```
>>> int(6.75)
6
```

Conversion Function	Example Use	Value Returned
int(<a number or a string>)	int(3.77)	3
	int("33")	33
float(<a number or a string>)	float(22)	22.0
str(<any value>)	str(99)	'99'

Functions and Modules

- ➔ Modules: Python includes many useful functions, which are organized in libraries of code called modules.
- ➔ function : is a chunk of code that can be called by name to perform a task.
- ➔ arguments : specific data values, passed to the function to perform the desired tasks.
- ➔ parameters : Names that refer to arguments.
- ➔ returning a value : The process of sending a result back to another part of a program.

Functions and Modules

argument

- round(6.5) , the value returned is 7 .
- abs(4 – 5) : If argument is an expression first evaluates the expression 4 – 5 and then passes the result, –1 , to abs .

argument

- optional arguments
- required arguments
- round(7.563) : returns 8.
- round(7.563, 2) : returns 7.56 .

Functions and Modules : math Module

- The math module includes several functions that perform basic mathematical operations.
- programmer must explicitly import other functions from the modules where they are defined.
  - import math
  - math.sqrt(2)
  - from math import pi, sqrt
  - >>> print(pi, sqrt(2))
- help is available if needed: help(math.cos)

Loops and Selection Statements

- Loops : repeat an action
- Each repetition of the action is known as a pass or an iteration .
- two types of loops
  - definite iteration :those that repeat an action a predefined number of times
  - indefinite iteration : those that perform the action until the program determines that it needs to stop .

Definite Iteration: The for Loop

Loop header :The integer expression in loop header denotes the number of iterations that the loop performs.

```
for <variable> in range(<an integer expression>):
    <statement-1>
    .
    .
    <statement-n>
```

The colon (:) ends the loop header.

loop body comprises the statements in the remaining lines of code, below the header. These statements are executed in sequence on each pass through the loop.

loop to compute an exponentiation for a nonnegative exponent.

```
number = 2
exponent = 3
product = 1
for eachPass in range(exponent):
 product = product * number
print(product)
```

Output : 8

for Loop : Count-Controlled Loops

- for loop counts from 0 to the value of the header's integer expression minus 1.
- On each pass through the loop, the header's variable is bound to the current value of this count.
-

```
>>> for count in range(4):
        print(count, end = " ")
0 1 2 3
```

Factorial using Count-Controlled Loops

```
>>> product = 1
>>> for count in range(1, 5):
        product = product * count
>>> product
```

```
for <variable> in range(<lower bound>, <upper bound + 1>):
    <loop body>
```

Augmented Assignment

The assignment symbol can be combined with th arithmetic and concatenation operators. <variable> <operator>= <expression>

```
a = 17
s = "hi"
a += 3          # Equivalent to a = a + 3
a -= 3          # Equivalent to a = a - 3
a *= 3          # Equivalent to a = a * 3
a /= 3          # Equivalent to a = a / 3
a %= 3          # Equivalent to a = a % 3
s += " there"   # Equivalent to s = s + " there"
```

for Loop : Traversing the Contents of a Data Sequence

```
for <variable> in <sequence>:
    <do something with variable>
```

```
>>> for number in [6, 4, 8]:
        print(number, end = " ")
6 4 8
>>> for character in "Hi there!":
        print(character, end = " ")
H i  t h e r e !
```

for Loop : Specifying the Steps in the Range

- A variant of Python range function expects a third argument that allows you to nicely skip some numbers. The third argument specifies a step value
-
```
>>> list(range(1, 6, 1))
[1, 2, 3, 4, 5]
>>> list(range(1, 6, 2))
[1, 3, 5]
>>> list(range(1, 6, 3))
[1, 4]
```

for Loop : Specifying the Steps in the Range

sum of the even numbers between 1 and 10.

```
>>> theSum = 0
>>> for count in range(2, 11, 2):
        theSum += count
>>> theSum
30
```

for Loop : Count Down

```
>>> for count in range(10, 0, -1):
        print(count, end = " ")
10 9 8 7 6 5 4 3 2 1
```

Selection: if and if-else Statements

- selection statements , or control statements, allows a computer to make choices.
- Comparison operator in python

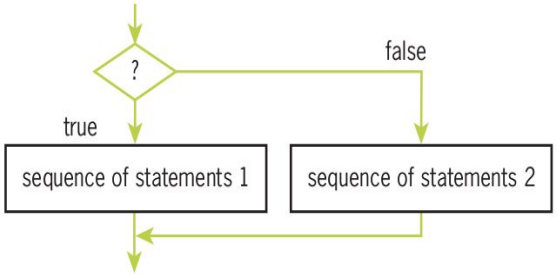
Comparison Operator	Meaning
==	Equals
!=	Not equals
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal

```
>>> 4 == 4
True
>>> 4 != 4
False
>>> 4 < 5
True
>>> 4 >= 3
True
>>> "A" < "B"
True
```

if-else Statements

➔ It is also called a two-way selection statement.

```
if <condition>:
    <sequence of statements-1>
else:
    <sequence of statements-2>
```




```
import math
area = float(input("Enter the area: "))
if area > 0:
    radius = math's(area / math.pi)
    print("The radius is", radius)
else:
    print("Error: the area must be a positive number")
```

compound Boolean expression

➔ The two conditions can be combined in a Boolean expression that uses the logical operator or / and

```
number = int(input("Enter the numeric grade: "))
if number > 100 or number < 0:
    print("Error: grade must be between 100 and 0")
else:
    # The code to compute and print the result goes here
```

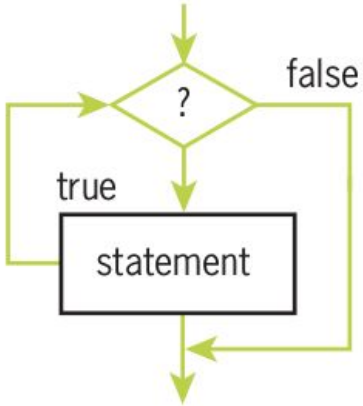
```
number = int(input("Enter the numeric grade: "))
if number >= 0 and number <= 100:
    # The code to compute and print the result goes here
else:
    print("Error: grade must be between 100 and 0")
```

Conditional Iteration: The while Loop

- the process continues to repeat as long as a condition remains true.
- The input loop accepts these values until the user enters a special value [sentinel] that terminates the input.
- This type of process is called conditional iteration.

```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
while <condition>:
    <sequence of statements>
```



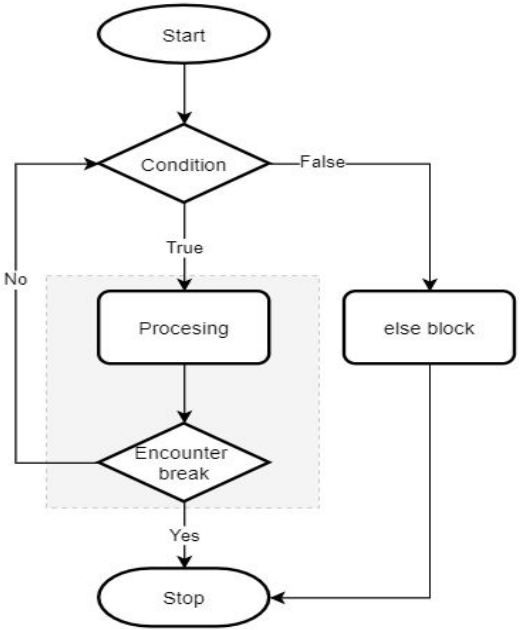
while True Loop and the break Statement

```
n = 5
while n > 0:
    n = n-1
    if n == 2:
        break
    print(n)
print('Loop is finished')
```

Python For-Else and While-Else

- The condition is checked at the beginning of each iteration.
- The code block inside the while statement will execute as long as the condition is True.
- When the condition becomes False and the loop runs normally, the else clause will execute.
- if the loop is terminated prematurely by either a break or return statement, the else clause won't execute at all.

```
while condition:
    # code block to run
else:
    # else clause code block
```



Python For-Else and While-Else

```
my=["a", "b", "Diamond", "c"]
print(my)
search="D"
for item in my:
 if(item==search):
  print("Found!")
  break
else:
 print("Not found.")
```

Output

Not found

Switch- case

- ➔ In Python, there is no case statement by default.
- ➔ The concept can be implemented using if …else block.

Lazy Evaluation [pending topic in module I]

- lazy evaluation means an object is evaluated when it's needed.
- Lazy evaluation is a concept that many programming languages utilize to help optimize performance at runtime.
- Eg:

Sample programming questions

- Write a Python program to reverse a given number
- Write a Python program to find the sum of digits of a given number
- find the sum of cubes of all positive even numbers within a given range.
- Find

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

Radian value of x= pi/180

Read number of terms and x from the user

- Check the given string is palindrome without using string functions.
- Generate fibonacci series within a given range
- Find the circumference of a circle for a given radius.
- Python Program to check Armstrong Number

```
abcd... = pow(a,n) + pow(b,n) + pow(c,n) + pow(d,n) + ....
```

- Python Program to Solve Quadratic Equation
-
-