Basic operators: `+, -, *, /, //` (integer division), % (modulus), `**` (exponentiation)

Print statement: **`print`**`(thing1, thing2,… [end = "\n"])`

Print a blank line **`print`**`()`

Conditional statements - **`if`**, **`elif`** and **`else`**:

```
if <condition>:
  statement
  statement
  statement
```

```
if <condition>:
  statement
  statement
else:
  statement
  statement
```

```
if <condition>:
  statement
elif <condition>:
  statement
else:
  statement
```

Make sure your `<condition>` resolves to a Boolean (**`True`** or **`False`**) value. Don't forget the colon at the end of **`if`**, **`elif`** and **`else`** and remember to indent accordingly.

Relational operators: `>, <, ==, !=, >=, <=`

Loops – **`for`** loop and **`while`** loop:

`for` loop:

```
for iterating_var in sequence:
  statement
  statement
  …
```

```
for i in range(5):
  print(i, end = " ")

# 0 1 2 3 4
```

```
for value in [1,3,4,6,8]:
  print(value, end = " ")

# 1 3 4 6 8
```

```
for char in "hello":
  print(char, end = " ")

# h e l l o
```

`range()` function: `range([start], stop, [step]`

Generate a sequence from `start` up to but not including `stop`, with a difference of `step` between consecutive items. `start` and `step` are optional parameters, if omitted, they default to 0 and 1, respectively.

```
>> range(7)
0, 1, 2, 3, 4, 5, 6
```

```
>> range(1, 7)
1, 2, 3, 4, 5, 6
```

```
>> range(1, 7, 2)
1, 3, 5
```

```
>> range(7, 3, -1)
7, 6, 5, 4
```

`while` loop:

```
while <condition>:
  statement
  statement
```

Strings:

Operation	**Syntax**	**Example** **aStr = "Hello!"**	**Result**
Indexing: Access a character in a string[1]	`aStr[index]`	`aStr[1]`	`'e'`
Slicing: Extract part of a string[2]	`aStr[start: stop]`	`aStr[1:5]`	`'ello'`
Concatenation	`aStr + anotherStr`	`aStr + ' Yay!'`	`'Hello! Yay!'`
Repetition	`aStr * i`	`aStr * 3`	`'Hello!Hello!Hello!'`
Membership: check if a substring is in a string	`subStr in aStr`	`'llo' in aStr`	**`True`**
Length: return the number of characters in a string	`len(aStr)`	`len(aStr)`	`6`

[1] Indexing: 0-based index: 0, 1, 2, 3, …

`aStr[-1]` returns the last character of the string, `aStr[-2]` returns the 2nd last character of the string,…

[2] Slicing: If `start` is omitted, it defaults to 0 (slice starts at the 1st character). If `stop` is omitted, it defaults to -1 (slice ends at the last character)

Other string operations:

Operation	**Explanation**
`aStr.upper()`	Return `aStr` in all upper-case characters
`aStr.lower()`	Return `aStr` in all lower-case characters
`aStr.strip()`	Return `aStr` with leading and trailing whitespaces removed
`aStr.count(subStr)`	Return number of occurrences of `subStr` in `aStr`
`aStr.split()`	Return a list of substrings of `aStr` separated by whitespace.
`aStr.find(subStr)`	Return the starting index of the first occurrence of `subStr` in `aStr`
`aStr.replace(old, new)`	Return a new string with all occurences of `old` in `aStr` replaced by `new`

Lists

Indexing, Slicing, Concatenation, Repetition, Membership & Length: see *String*

Operation	**Explanation**
`aList = list()` or `aList = []`	Initialize a new, empty list
`aList.append(item)`	Add `item` to the end of `aList`
`aList.extend(otherList)`	Extend `aList` by adding all items in `otherList` to the end of `aList`
`aList.insert(pos, item)`	Insert `item` to `aList` at index `pos`
`aList.pop()`	Remove and return the last item in `aList`
`aList.pop(pos)`	Remove and return the item at index `pos` from `aList`
`aList.remove(item)`	Remove the first occurrence of `item` in `aList`
`aList.index(item)`	Return the index of the first occurrence of `item` in `aList`
`aList.count(item)`	Return the number of occurrences of `item` in `aLis`t
`aList.sort()`	Sort `aList` (default = ascending)
`aList.reverse()`	Reverse the order of `aList`

Dictionaries

Operation	**Explanation**
`aDict = dict()` or `aDict = {}`	Initialize a new, empty dictionary
`len(aDict)`	Return the number of key-value pairs in `aDict`
`aDict[aKey] = aValue`	Add a new key-value pair to a dictionary. If key `aKey` exists, this will modify the value of `aDict[aKey]` instead
`aKey` **`in`** `aDict`	Returns **`True`** if `aKey` is in `aDict`, otherwise **`False`**
`del aDict[aKey]`	Delete the key-value pair in `aDict` whose key is `aKey`

Sets

Operation	**Explanation**
`aSet = set()`	Initialize a new, empty set
`aSet = set(sequence)`	Create a new set from a sequence
`len(aSet)`	Return the number of elements in `aSet`
`ele` **`in`** `aSet`	Return **`True`** if `ele` is in `aSet`, otherwise **`False`**
`aSet.add(ele)`	Add element `ele` to `aSet`
`aSet.remove(ele)`	Remove element `ele` from `aSet`
`aSet.union(anotherSet)` `aSet | anotherSet`	Return a new set with elements from both `aSet` and `anotherSet`
`aSet.intersection(anotherSet)` `aSet & anotherSet`	Return a new set with elements common to `aSet` and `anotherSet`
`aSet.difference(anotherSet)` `aSet – anotherSet`	Return a new set with elements in `aSet` but not in `anotherSet`
`aSet.symmetric_difference(anotherSet)` `aSet ^ anotherSet`	Return a new set with elements from either `aSet` or `anotherSet`, but not both

Files

Operation	**Explanation**
`open(filename, [mode])` `f = open('input.txt')`	Open a file to be used. The modes are `'r'` (read-only, default), `'w'` (write-only), `'a'` (append), `'r+'` (both reading and writing)
`f.close()`	Close the file after reading / writing
`for` `line` **`in`** `f:` `  statement`	Loop through a file, line-by-line
`f.read()`	Return the whole file as a single string
`f.readline()`	Return the next line from the file
`f.readlines()`	Return a list containing all the lines (string) of the file
`f.write(aString)`	Write `aString` to the file