< Exam Reference Sheet > - DSA8640

Algebraic expressions

Usage	Explanation
a + b	sum of a and b
a - b	difference of a and b
a * b	product of a and b
a / b	quotient of a and b
a ** b	a to the b power
a // b	quotient from floor division of a and b
a % b	remainder of a / b

Boolean expressions

Usage	Explanation
a < b	a is smaller than b
a <= b	a is smaller than or equal to b
a > b	a is bigger than b
a >= b	a is bigger than or equal to b
a == b	a is equal to b
a != b	a is not equal to b

String operators, functions, and methods

Usage	Explanation
`x in s`	x is a substring of s
`x not in s`	x is not a substring of s
`s + t`	Concatenation of s and t
`s * n, n * s`	Concatenation of n copies of s
`s[i]`	Character at index i of s
`len(s)`	(function) Length of string s

`s[i:j]` : the slice of s starting at index i and ending before index j

`s[i:]` : the slice of s starting at index i

`s[:j]` : the slice of s ending before index j

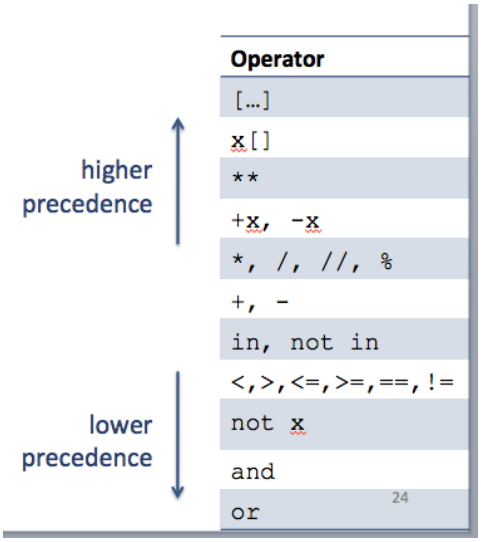
Usage	Explanation
`s.capitalize()`	returns a copy of `s` with first character capitalized
`s.count(target)`	returns the number of occurrences of `target` in `s`
`s.find(target)`	returns the index of the first occurrence of `target` in `s`
`s.lower()`	returns lowercase copy of `s`
`s.upper()`	returns uppercase copy of `s`
`s.split(sep)`	returns list of substrings of `s`, delimited by `sep`

List operators, functions, and methods

Usage	Explanation
`x in lst`	`x` is an item of `lst`
`x not in lst`	`x` is not an item of `lst`
`lst + lstB`	Concatenation of `lst` and `lstB`
`lst*n, n*lst`	Concatenation of `n` copies of `lst`
`lst[i]`	Item at index `i` of `lst`
`len(lst)`	Number of items in `lst`
`min(lst)`	Minimum item in `lst`
`max(lst)`	Maximum item in `lst`
`sum(lst)`	Sum of items in `lst`

Usage	Explanation
`lst.append(item)`	adds `item` to the end of `lst`
`lst.count(item)`	returns the number of times `item` occurs in `lst`
`lst.index(item)`	Returns index of (first occurrence of) `item` in `lst`
`lst.pop()`	Removes and returns the last item in `lst`
`lst.remove(item)`	Removes (the first occurrence of) `item` from `lst`
`lst.reverse()`	Reverses the order of items in `lst`
`lst.sort()`	Sorts the items of `lst` in increasing order

Operator precedence



Import a module

```
import <module>
```

One-way if statement syntax

```
if <condition>:
    <indented code block>
<non-indented statement>
```

Two-way if statement syntax

```
if <condition>:
    <indented code block 1>
else:
    <indented code block 2>
<non-indented statement>
```

Multi-way if statement syntax

```
If <condition>:
    <indented code block 1>
elif <condition2>:
    <indented code block 2>
else:
    <indented code block 3>
<non-indented statement>
```

for loop syntax

```
for <variable> in <sequence>:
    <indented code block >
<non-indented code block>
```

while loop syntax

```
while <condition>:
    <indented code block>
<non-indented statement>
```

function definition syntax

```
def <function name> (<0 or more variables>):
    <indented function body>
```

range() function

- To iterate over the n numbers 0, 1, 2, …, n-1
```
for i in range(n):
```
- To iterate over the range j, j+1, j+2, …, n-1
```
for i in range(j, n):
```
- To iterate over the range with step c: j, j+c, j+2c, j+3c, …, n-1
```
for i in range(j, n, c):
```

File modes

The file mode defines how the file will be accessed

Mode	Description
r	Reading (default)
w	Writing (if file exists, content is wiped)
a	Append (if file exists, writes are appended)
r+	Reading and Writing
t	Text (default)
b	Binary

File methods

Usage	Description
infile.read(n)	Read n characters starting from cursor; if fewer than n characters remain, read until the end of file
infile.read()	Read starting from cursor up to the end of the file
infile.readline()	Read starting from cursor up to, and including, the end of line character
infile.readlines()	Read starting from cursor up to the end of the file and return list of lines
outfile.write(s)	Write string s to file outfile starting from cursor
infile.close(n)	Close file infile

20

`format` method of class `str`

```
print('{}'.format(<variable>))
```

Dictionary methods

Operation	Explanation
d.items()	Returns a view of the (key, value) pairs in d
d.keys()	Returns a view of the keys of d
d.pop(key)	Removes the (key, value) pair with key key from d and returns the value
d.update(d2)	Adds the (key, value) pairs of dictionary d2 to d
d.values()	Returns a view of the values of d

Module random

random.randrange() : takes a pair of integers a and b, and returns some number between a and b-1

random.uniform(): takes two numbers a and b, and returns a float number x such that a <= x <= b

random.shuffle(): shuffles, or permutes, the objects in a sequence

random.choice(): allows us to choose an item from a container uniformly at random

random.sample(): takes an input the container and an integer k, and returns a list of k items in the container

Class definition, in general

```
class <Class Name>:
    <class variable 1> = <value>
    <class variable 2> = <value>
    ...
    def <class method 1>(self, arg11, arg12, ...):
        <implementation of class method 1>

    def <class method 2>(self, arg21, arg22, ...):
        <implementation of class method 2>

    ...
```

A class method is really a function defined in the class namespace; when Python executes

```
instance.method(arg1, arg2, …)
```

it first translates it to

```
class.method(instance, arg1, arg2, …)
```

and actually executes this last statement

Operator	Method
x + y	x.__add__(y)
x – y	x.__sub__(y)
x * y	x.__mul__(y)
x / y	x.__truediv__(y)
x // y	x.__floordiv__(y)
x % y	x.__mod__(y)
x == y	x.__eq__(y)
x != y	x.__ne__(y)
x > y	x.__gt__(y)
x >= y	x.__ge__(y)
x < y	x.__lt__(y)
x <= y	x.__le__(y)
repr(x)	x.__repr__()
str(x)	x.__str__()
len(x)	x.__len__()
<type>(x)	<type>.__init__(x)