| Method | Description |
|---|---|
| capitalize() | Converts the first character to upper case |
| casefold() | Converts string into lower case |
| center() | Returns a centered string |
| count() | Returns the number of times a specified value occurs in a string |
| encode() | Returns an encoded version of the string |
| endswith() | Returns true if the string ends with the specified value |
| expandtabs() | Sets the tab size of the string |
| find() | Searches the string for a specified value and returns the position of where it was found |
| format() | Formats specified values in a string |
| format_map() | Formats specified values in a string |
| index() | Searches the string for a specified value and returns the position of where it was found |
| isalnum() | Returns True if all characters in the string are alphanumeric |
| isalpha() | Returns True if all characters in the string are in the alphabet |
| isascii() | Returns True if all characters in the string are ascii characters |
| isdecimal() | Returns True if all characters in the string are decimals |
| isdigit() | Returns True if all characters in the string are digits |
| isidentifier() | Returns True if the string is an identifier |
| islower() | Returns True if all characters in the string are lower case |
| isnumeric() | Returns True if all characters in the string are numeric |
| isprintable() | Returns True if all characters in the string are printable |
| isspace() | Returns True if all characters in the string are whitespaces |
| istitle() | Returns True if the string follows the rules of a title |
| isupper() | Returns True if all characters in the string are upper case |
| join() | Converts the elements of an iterable into a string |
| ljust() | Returns a left justified version of the string |
| lower() | Converts a string into lower case |
| lstrip() | Returns a left trim version of the string |
| maketrans() | Returns a translation table to be used in translations |
| partition() | Returns a tuple where the string is parted into three parts |
| replace() | Returns a string where a specified value is replaced with a specified value |
| rfind() | Searches the string for a specified value and returns the last position of where it was found |
| rindex() | Searches the string for a specified value and returns the last position of where it was found |
| rjust() | Returns a right justified version of the string |
| rpartition() | Returns a tuple where the string is parted into three parts |
| rsplit() | Splits the string at the specified separator, and returns a list |
| rstrip() | Returns a right trim version of the string |
| split() | Splits the string at the specified separator, and returns a list |
| splitlines() | Splits the string at line breaks and returns a list |
| startswith() | Returns true if the string starts with the specified value |
| strip() | Returns a trimmed version of the string |
| swapcase() | Swaps cases, lower case becomes upper case and vice versa |
| title() | Converts the first character of each word to upper case |
| translate() | Returns a translated string |
| upper() | Converts a string into upper case |
| zfill() | Fills the string with a specified number of 0 values at the beginning |

| Type | Notation | Example |
|---|---|---|
| Addition | + | 5 + 6 = 11 |
| Subtraction | - | 5 – 6 = -1 |
| Multiplication | * | 5 * 6 = 30 |
| Division (float) | / | 5 / 6 = 8.3334 |
| Modulus (Remainder after division) | % | For x % y, if x < y, = x<br>5 % 6 = 5<br>6 % 5 = 1<br><br>Find even numbers:<br>i % 2 == 0 |
| Exponential | ** | 5 ** 6 = 15625 |
| Floor Division | // | For x // y, if x < y, = 0<br>5 // 6 = 0<br>6 // 5 = 1 (always round down to nearest int) |

- Three ways to use range() function:
  - range(stop) takes one argument:

```
for i in range(5):
    print(i)
```

  - range(start, stop) takes two arguments:

```
for i in range(3, 10):
    print(i)
```

  - range(start, stop, step) takes three arguments:

```
for i in range(3, 10, 4):
    print(i)
```

```
for i in range(5, 0, -1):
    print(i)
```

```
def reverseStringI(s):
    output = ''
    for c in s:
        output = c + output
    return output

def reverseStringI(s):
    output = ''
    l = len(s)
    for i in range(l):
        output += s[l-i-1]
    return output

def reverseStringR(s):
    if not s:
        return ''
    return reverseStringR(s[1:])+s[0]
```

```
def fibonaci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonaci(n-1) + fibonaci(n-2)
```

```
def factorial(x):
    if x == 1: # Base Case
        return 1
    else:
        return (x * factorial(x-1))
```

```
x = factorial(3)

def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

3*2 = 6 is returned

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

2*1 = 2 is returned

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

1 is returned

```
def is_odd(n):
    if n == 0:
        return False
    else:
        return is_even(n-1)

def is_even(n):
    if n == 0:
        return True
    else:
        return is_odd(n-1)
```

**Iteration**

```
def sum(n):
    res = 0
    while n > 0:
        res = res + n%10
        n = n//10
    return res
```

```
def binom_coeff(n, k):
    if k < 0 or k > n:
        return 0
    elif k == 0 or k == n:
        return 1
    if k > n - k:
        k = n - k

    result = 1
    for i in range(k):
        result *= n - i
        result //= i + 1
    return result
```

```
def binom_coeff_recur(n, k):
    if k < 0 or k > n:
        return 0
    elif k == 0 or k == n:
        return 1

    return binom_coeff_recur(n - 1, k - 1) \
        + binom_coeff_recur(n - 1, k)
```

```
def monte_carlo_pi(n):
    inside_circle_count = 0

    for _ in range(n):
        x = random.uniform(-1, 1)
        y = random.uniform(-1, 1)

        if x**2 + y**2 <= 1**2:  # Pythagoras Theorem, x^2 + y^2 = r2
            inside_circle_count += 1

    pi = (inside_circle_count / n) * 4
    return pi
```

SetA = {1,2,3,4}
SetB = {3,4,5,6}

| | | |
|---|---|---|
| SetA \| SetB *OR* | Union (combine both sets but only with element occurring once) | {1,2,3,4,5,6} |
| SetA & SetB *AND* | Intersection (common element) | {3,4} |
| SetA – SetB | A – B,<br>Remove any element that is in A that is from B | {1,2} |
| SetA ^ SetB *XOR* | (A\|B)-A&B<br>The order of function is Union and intersect first comes before minus | {1,2,5,6}<br>get the different ones |

```
def calc_poly(const_seq, var_poly):
    result = 0
    for i in range(len(const_seq)):
        result += const_seq[i] * (var_poly ** i)
    return round(result, 2)
```

```
def is_anagram(s1, s2):
    s1 = s1.lower()
    s2 = s2.lower()

    if len(s1) != len(s2):
        return False

    s1_count = {}
    s2_count = {}

    for i in s1:
        s1_count[i] = s1_count.get(i, 0) + 1

    for i in s2:
        s2_count[i] = s2_count.get(i, 0) + 1

    return s1_count == s2_count
```

```
def checkPrime(n):
    for i in range(2,n):
        if divisible(n,i):
            return False
    return True
```

| Function | Description |
|---|---|
| pop() | Remove the item with the specified key. |
| update() | Add or change dictionary items. |
| clear() | Remove all the items from the dictionary. |
| keys() | Returns all the dictionary's keys. |
| values() | Returns all the dictionary's values. |
| get() | Returns the value of the specified key. |
| popitem() | Returns the last inserted key and value as a tuple. |
| copy() | Returns a copy of the dictionary. |

```
{ key : value, key : value }
country_capitals = {
  "United States": "Washington D.C.",
  "Italy": "Rome",
  "England": "London"
}

print(country_capitals["United States"])
 # Washington D.C.

country_capitals["Germany"] = "Berlin
 # add an item with "Germany" as key
and "Berlin" as its value

del country_capitals["United States"]
 # delete item having "United States" key
```

In a dictionary in Python, The values can be any data type , but the key cannot be list, dict, and set.

```
dd = {1: {2: 3}, 2: {3: 1}, 3: {1: 3}}
print(dd[2][3])   # 1
print(dd[dd[2][3]])   # {2: 3}
# The max function is applied to the keys of the dictionary
# {2: 3}, and the maximum key is 2, so it prints 2.
print(max(dd[dd[2][3]]))
```

```
print([1, 2, [3, 4], 5, 6][[1, 2, 4][2]:[1, 2, 3, 4, 5][3]])  # []
# Steps
# [1, 2, [3, 4], 5, 6][4:4]

# lambda no return statement
(lambda x,y,z:return x-y+z)(3,2,1) # SyntaxError: 'return' outside function
```

```
# 17//2 + 17//2//2 = 8 + 4 = 12
print((lambda x, y: y(y(x))+y(x))(17, lambda x: x//2))

print((lambda x: x((lambda x: x(lambda x: x))(x(x))))
      (lambda x: x)(lambda x: x+x)(3))  # 3+3 = 6 the rest is just x = x
```

**Lambda**
A pseudo temporary function called

| | |
|---|---|
| Def add1(x):<br>  Return x+1 | func = lambda x: x+1 |
| Def make_power_func(n):<br>  Return lambda x:x**n | Square = make_power_func(2)<br>Print(square(3)) = 9<br>Cube = make_power_func(3)<br>Print(cube(2)) = 8 |

**map(function, sequence)** → apply function to every element in sequence and RETURNS the element with function applied

**map(function, sequence)** → apply function to every element in sequence and RETURNS the element with function applied

```
tup = (1, -2, 3)
map1 = map(abs, tup)
map1 #map object
type(map1) #class map
map1list = list(map1)  # [1,2, 3]
map1tuple = tuple(map1)  #(), this is empty as it is already taken out
of map1 for map1list. You can only convert to list/tuple once
```

**filter(predicate, sequence)** → Applies a function(predicate) that returns True or False to sequence's elements. Keep only the item if f(x) returns True, remove otherwise

```
l = [1, 2, 3, 'a', (1,2), ('b', 3)]
filter(lambda x:type(x) == int, l)  #filter object
list(filter(lambda x:type(x)==int, l) # [1, 2, 3]
```

**ASCII**

ord('A') -> 65  ord('a') -> 97
chr(65) -> 'A'  chr(97) -> 'a'

```
lst = [1, 2, 3]
# append(): Adds an element at the end of the list.
lst.append(4)  # [1, 2, 3, 4]
# extend(): Adds elements of a list to the end of the current list.
lst.extend([5, 6])  # [1, 2, 3, 4, 5, 6]
# insert(): Adds an element at a specified position.
lst.insert(1, 'a')  # [1, 'a', 2, 3, 4, 5, 6]
# remove(): Removes the first occurrence of the element with the specified value.
lst.remove('a')  # [1, 2, 3, 4, 5, 6]
# pop(): Removes the element at the specified position, or the last item.
lst.pop()  # [1, 2, 3, 4, 5]
# clear(): Removes all the elements from the list.
lst.clear()  # []
# reverse(): Reverses the order of the list.
lst.reverse()  # [5, 4, 3, 2, 1]
len(lst)  # Output: 5
min(lst)  # Output: 1
max(lst)  # Output: 5
print(1 in lst)  # Output: True
print(6 not in lst)  # Output: True

tup = (1, 2, 3, 2, 2, 3)
# count(): Returns the number of times a specified value occurs in a tuple.
tup.count(2)  # Output: 3
# index(): Searches the tuple for a specified value and returns the position.
tup.index(3)  # Output: 2

lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# slicing [start:stop:step] (stop is exclusive)
# (start defaults to 0) (step defaults to 1)
# if start > stop or start > len(), empty list is returned
lst[1:4]  # Output: [2, 3, 4]
lst[3:]  # Output: [4, 5, 6, 7, 8, 9, 10]
lst[:3]  # Output: [1, 2, 3]
lst[::2]  # Output: [1, 3, 5, 7, 9]
lst[1:5:2]  # Output: [2, 4]
lst[0:6:-1]  # Output: []
lst[::-1]  # Output: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
s1 = '1234567890'
print(s1[8:2:-12])  # 9
print(s1[-2:2:-2])  # 975
print(s1[8:-7:-2])  # 975
print(s1[8:3:-2])  # 975
print(s1[-2:3:-2])  # 975
print(s1[8:4:-2])  # 97
```

**Higher Order Functions:**

```
e.g., 1 :
def f():
    print("hello")
def do_twice(x):
    x()
    x()

do_twice(f)
>> "hello"
>>"hello"
```

```
DeepMapping:
def deepMap(func, seq):
    if seq == []:
        return seq
    elif type(seq) != list:
        return func(seq)
    else:
        return [deepMap(func, seq[0])] + deepMap(func, seq[1:])
```

```python
def f1(x):
    return 1+f3(x)
def f3(x):
    return 1+f1(x)
f1(4) # RecursionError: maximum recursion depth exceeded in comparison

def f1(x):
    return '1'+f2(x)
def f2(x):
    return f3(x)+'2'
def f3(x):
    return '3'+f4(x)
def f4(x):
    return '4'+x
f1(0) # Error because '4' (str) + 0 (int) is not possible
```

```python
def chking(seq):
    d = list(seq)
    b = len(seq)
    for a in range(b-1):
        for i in d:
            if i == d[a:b]:
                # will never get executed because i is an element of d,
                #and it's not possible for an element to be equal to a sublist d[a:b].
                return False
            elif i >= max(d[a+1:b]):
                return True
            else:
                return False
    return True
```

```python
def f(x):
    return lambda y: x(y+1)
def g(x):
    return f(g)(x+1)
print(g(1))
```

Ans:

g(1)

f(g)(1+1)

(lambda y: g(y+1)) (2)

g(2+1)

g(3)

and the number keep increasing….

Infinite loop

```python
x = {'a', 'bc', 'de'}
y = {'b', 'de', 'a', 'b'}
# {'bc', 'b'} symmetric difference means the
# elements that are in one set but not the other
print(x ^ y)
```

```python
[5, [3], [2, 3]][[2, 1][0]][:[1, 2][1]]  # [2, 3]
# Steps:
# [[2, 1][0]]  # [2]
# [:[1, 2][1]]  # [:2]
# [5, [3], [2, 3]][2][:2]
# [2, 3][:2]  # [2, 3]


(lambda a, b: lambda x: b(x(a)))('a', lambda a: a*2)(lambda a: a[1:])  # ''
# Steps:
# (lambda a, b: lambda x: b(x(a)))('a', lambda a: a*2)
# lambda x: (lambda a: a*2)(x('a'))
# (lambda a: a*2)('a'[1:])
# (lambda a: a*2)('')
```

```python
lst1 = ['bc', 'de', 'ya', 'ab', 'bq', 'bd']
lst2 = []
for x in lst1:
    lst2.append(tuple(x))
# d = dict(lst2) # dict object is not callable
# [('b', 'c'), ('d', 'e'), ('y', 'a'), ('a', 'b'), ('b', 'q'), ('b', 'd')]
print(lst2)
```

```python
def foo(x):
    return lambda x: x+x

print(foo(5)(2))  # 4
```

```python
f = lambda f, g: (f, (g))
g = lambda g: g(g, g)
print(g(f)[0](4, 2))
```

Ans:

g(f)[0](4, 2)

(lambda g: g(g, g)) (f)[0](4, 2)

f(f, f) [0](4, 2)

(lambda f, g: (f, (g))) (f, f) [0](4, 2)

(f, (f))[0](4, 2)

f(4, 2)

lambda f, g: (f, (g)) (4,2)

(4,(2))

(4,2)

```python
x = {'a', 'bc', 'de', 'a'}
y = {'b', 'de', 'a', 'a', 'b'}
print(x ^ y)  # {'b', 'bc'} one have other dont
print(y-x ^ y)  # {'de', 'a'} remove from y
print(x | y-x ^ y)  # {'de', 'a', 'bc'} OR against x
```

```python
print(list((1)) + list([2]) + list('3'))
# TypeError: 'int' object is not iterable list((1))
```

```python
def foo(L):
    for i in range(len(L)-1):
        for j in range(len(L)-i):
            if L[j] > L[j+1]:  # +1 will eventually go out of range
                L[j], L[j+1] = L[j+1], L[j]
```

```python
dd = {'a': 0, 'b': 1, 'c': 2, 'd': 0, 'e': 1, 'f': 2}

def valKeys(dict, valLst):
    kys = []
    for val in valLst:
        kys.extend([k for (k, val) in dict.items()])
    return kys

print(valKeys(dd, [0, 2])) # ['a', 'b', 'c', 'd', 'e', 'f', 'a', 'b', 'c', 'd', 'e', 'f']
```

```python
x = ['a', 'b', 'c', 'd']

def foo(l, f):
    if not l:
        return l
    return foo(f(l[1:]), f)+[f(l[0])]

print(foo(x, lambda x: x[::-1]))  # ['c', 'b', 'd', 'a']
```

```python
dict = {1: 'Z', 2: 'MAX', 3: 'IS', 4: 'DEF', 5: 'UN', 6: 'A'}
print(dict[max(dict)] == min(dict.values()), dict[3] == dict.pop(
    3), dict.pop(min(dict.keys())) == max(dict.values()))
# 1. max(dict) returns the maximum key in the dictionary, which is 6.
# 2. min(dict.values()) returns the minimum value in the dictionary values, which is 'A'.
# 3. dict[max(dict)] == min(dict.values()):
#    This is equivalent to dict[6] == 'A', which is True.
# 4. dict[3] == dict.pop(3):
#    This is equivalent to 'IS' == dict.pop(3), which removes the key 3 and
#    returns its value 'IS'. So, this expression evaluates to True.
# 5. dict.pop(min(dict.keys())) == max(dict.values()):
#    min(dict.keys()) returns the minimum key, which is 1.
#    dict.pop(1) removes the key 1 and returns its value 'Z'.
#    So, this expression is equivalent to 'Z' == max(dict.values()), which is False.
```

```python
print(6-------6)  # 0

print(6-True+False**0)  # 6 - 1 + 0^0 = 6 - 1 + 1 = 6
```

```python
print(8/4*2)  # 4.0
```

```python
print('1234567'[2:5][1:2][1:])  # ""
'1234567'[2:5]  # 345
'1234567'[2:5][1:2]  # 4
'1234567'[2:5][1:2][1:]  # ""
```

```python
def circle(f, g):
    return lambda x: g(f(x))

def g(x): return (x, x)

h = circle(g, g)
print(h(3)) # ((3, 3), (3, 3))
```

```python
def rec(stg, acc):
    if stg:
        return rec(stg[1:], acc+'a')+'b'
    else:
        return acc

# First call: stg='1234', acc=''
# Calls rec('234', 'a') + 'b'
#   Second call: stg='234', acc='a'
#   Calls rec('34', 'aa') + 'b'
#       Third call: stg='34', acc='aa'
#       Calls rec('4', 'aaa') + 'b'
#           Fourth call: stg='4', acc='aaa'
#           Calls rec('', 'aaaa') + 'b'
#               Fifth call: stg='', acc='aaaa'
#               Returns 'aaaa'
#           Returns 'aaaab'
#       Returns 'aaaabb'
#   Returns 'aaaabbb'
# Returns 'aaaabbbb'


print(rec('1234', ''))
```

```python
tuple('xyz')+tuple((3))  # 3 is not iterable
```

| Method | Description |
|---|---|
| count() | Returns the number of times a specified value occurs in a tuple |
| index() | Searches the tuple for a specified value and returns the position of where it was found |

```python
def evenDigits(N):
    if N == 0:
        return 0
    if N % 2 == 0:
        return evenDigits(N//10)*10+N % 10
    else:
        return evenDigits(N//10)


print(evenDigits(123456))  # 246
print(evenDigits(12332401))  # 2240
```

| Method | Description |
|---|---|
| append() | Adds an element at the end of the list |
| clear() | Removes all the elements from the list |
| copy() | Returns a copy of the list |
| count() | Returns the number of elements with the specified value |
| extend() | Add the elements of a list (or any iterable), to the end of the current list |
| index() | Returns the index of the first element with the specified value |
| insert() | Adds an element at the specified position |
| pop() | Removes the element at the specified position |
| remove() | Removes the first item with the specified value |
| reverse() | Reverses the order of the list |
| sort() | Sorts the list |

```python
x = [1, 2, 3]


def foo(l, x):
    if not l:
        return l
    return foo(l[1:], x) + [x(l[0])]


print(foo(x, lambda x: 4-x))  # [1, 2, 3] it appends to the back of the list
```

```python
def foo(y):
    return lambda x: x(x(y))
def bar(x):
    return lambda y: x(y)
print((bar)(bar)(foo)(2)(lambda x:x+1))
```

Ans:

(bar)(bar)(foo)(2)(lambda x:x+1)

(lambda y: bar(y) (foo)(2)(lambda x:x+1)

bar(foo) (2)(lambda x:x+1)

lambda y:foo(y) (2)(lambda x:x+1)

foo(2) (lambda x:x+1)

lambda x: x(x(2)) (lambda x:x+1)

Lets simplify it as f = (lambda x:x+1)

lambda x: x(x(2)) (f)

f(f(2))  4

f(3)  4

```python
def foo(x):
    return lambda a: a(x)(a)
def bar(x):
    return lambda a: a(x+1)
print(foo(5)(bar)(lambda x:x))
```

Ans:

foo(5)(bar)(lambda x:x)

(lambda a: a(5)(a)) (bar)(lambda x:x)

bar(5)(bar) (lambda x:x)

(lambda a: a(5+1)) (bar) (lambda x:x)

bar(6) (lambda x:x)

lambda a: a(6+1) (lambda x:x)

7

```python
def f(x, y):
    return lambda z: (x)(y)(z)
def g(x):
    return lambda y: y(x)
print(f(g, 0)(g)(42))
```

Ans:

f(g, 0)(g)(42)

(lambda z: (g)(0)(z)) (g)(42)

(g)(0)(g) (42)

(lambda y: y(0)) (g) (42)

g(0) (42)

(lambda y: y(0)) (42)

(42)(0)

Error! 42 is not callable

| Mode | Description |
|---|---|
| r | Open a file for reading. (default) |
| w | Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists. |
| x | Open a file for exclusive creation. If the file already exists, the operation fails. |
| a | Open a file for appending at the end of the file without truncating it. Creates a new file if it does not exist. |
| t | Open in text mode. (default) |
| b | Open in binary mode. |
| + | Open a file for updating (reading and writing) |

- open() - takes two parameters; filename, and mode.
- write() - Writes the specified string to the file
- writelines() - writes a list of strings to the file
- close() - closes the file
- read () - returns the file content
- readable() - returns whether the file stream can be read of not
- readline() - returns one line from the file
- readlines() - returns a list of lines from the file

```
f = open("demofile3.txt", "w")
f.write("CS1010E CHEAT SHEET")
f.close()
```

# CS1010E Mid Term Cheat Sheet

Brians Tjipto

found: https://github.com/brianstm/NUS