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 5 % 6 = 5 6 % 5 = 1 Find even numbers: i % 2 == 0
Exponential	**	5 ** 6 = 15625
Floor Division	//	For x // y, if x < y, = 0 5 // 6 = 0 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:

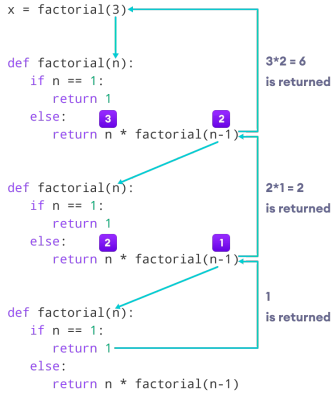
```
for i in range(3, 10):
    print(i)
```

- `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))
```




```
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)
```

```
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]
```

ASCII

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

```
# 1920 S1
def sum_digit_square_I(n):
    sum = 0
    while n > 0:
        digit = n % 10
        sum += digit**2
        n //= 10
    return sum

def sum_digit_square_R(n):
    if n == 0:
        return 0
    else:
        return (n % 10)**2 + sum_digit_square_R(n // 10)

def is_happy_number(n):
    prev = []
    while n != 1:
        if n in prev:
            return False
        else:
            prev.append(n)
            n = sum_digit_square_I(n)
    return True

def all_happy_number(n, m):
    result = []
    for i in range(n, m+1):
        if is_happy_number(i):
            result.append(i)
    return result

def is_unique_2(seq):
    for index, i in enumerate(seq):
        # enumerate('minions') gives
        # (0, 'm')
        # (1, 'i')
        # (2, 'n')
        # (3, 'i')
        # (4, 'o')
        # (5, 'n')
        # (6, 's')
        new_seq = seq[index+1:]
        for j in new_seq:
            # index:  0 | 1st:  m | 2nd:  i
            # index:  0 | 1st:  m | 2nd:  n
            # index:  0 | 1st:  m | 2nd:  i
            # index:  0 | 1st:  m | 2nd:  o
            # index:  0 | 1st:  m | 2nd:  n
            # index:  0 | 1st:  m | 2nd:  s
            # index:  1 | 1st:  i | 2nd:  n
            # index:  1 | 1st:  i | 2nd:  i
            if i == j:
                return False
    return True
```

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

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

```
def item_price(item):
    if 'B' in item:
        return burger_price(item)
    elif 'D' in item or 'F' in item:
        size = item[1]
        base_price = 3 if 'D' in item else 4
        if size == 'S':
            return base_price
        elif size == 'M':
            return base_price + 1
        elif size == 'L':
            return base_price + 2
```

```
def matchResistors(R, n):
    resistor_list = sorted(list(R))
    left = 0
    right = len(resistor_list) - 1
    result_pairs = []

    while left < right:
        if resistor_list[left] + resistor_list[right] == n:
            result_pairs.append((resistor_list[left], resistor_list[right]))
            left += 1
            right -= 1
        elif resistor_list[left] + resistor_list[right] < n:
            left += 1
        else:
            right -= 1

    return result_pairs
```

Iteration

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

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

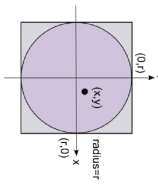
```
def is_anagram(s1, s2):
    s1 = s1.lower()
    s2 = s2.lower()

    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 calc_poly(var, coeffs):
    result = 0
    for i in range(len(coeffs)):
        result += coeffs[i] * var**i
    return result
```

```
def total_price_with_set_discount(order):
    burger_count = sum(1 for item in order if 'B' in item)
    drink_count = sum(1 for item in order if 'D' in item)
    fries_count = sum(1 for item in order if 'F' in item)
    meal_count = min(burger_count, drink_count, fries_count)
    discount = 10 * meal_count
    return total_price(order) - discount
```

The algorithm to approximate the value of π using the method described earlier is as follows.

1. Let the radius of the circle be r and the length of the square be $2r$. (Is it ok to just simplify $r = 1$? Can I use any value for r?)
2. Generate two random numbers x, y, such that $-r \leq x, y \leq r$. The position (x, y) is our dart position. *Hint: Use the function in the package random that starts with the letter 'u'.*
3. Calculate the distance d from the dart position to the center of the circle.
4. If $d \leq r$. This implies that the point (x, y) is within the circle. Otherwise, (x, y) falls outside of the circle, but in the square. Remember, you cannot use the function sqrt(). How do you get around this?
5. Repeat the procedure n times and keep track of the number of points that fall inside the circle, k.
6. After you have completed n trials, compute $p = \frac{k}{n}$.
7. For large n, the value of π can be approximated by $4p$.



```python
def rotate(bouquet, step):
    bouquet = list(bouquet)
    for _ in range(step):
        tmp = bouquet[0]
        bouquet.pop(0)
        bouquet.append(tmp)

    return tuple(bouquet)


def flower_I(bouquet, k):
    bouquet = list(bouquet)
    result = ""

    for _ in range(len(bouquet)):
        bouquet = list(rotate(bouquet, k-1))
        selected_flower = bouquet.pop(0)
        result += selected_flower

    return result


def flower_R(bouquet, k):
    bouquet = list(bouquet)

    if len(bouquet) == 1:
        return bouquet[0]
    else:
        bouquet = list(rotate(bouquet, k-1))
        selected_flower = bouquet.pop(0)
        remaining_flowers = flower_R(bouquet, k)
    return selected_flower + remaining_flowers
def pink_rose(bouquet):
    if len(bouquet) == 0 or "P" not in bouquet:
        return -1
    i = 0
    while True:
        seq = flower_I(bouquet, i)
        if seq[-1] == "P":
            return i
        i += 1
```

```python
def encode_I(word):
    letter_to_number = {'A': '00', 'B': '01', 'C': '02', 'D': '03', 'E':
                        'N': '13', 'O': '14', 'P': '15', 'Q': '16', 'R':
    # letter_to_number = {chr(i+65): str(i).zfill(2) for i in range(26)}
    encoded_word = ''

    for char in word:
        encoded_word += letter_to_number[char]
    return encoded_word


def encode_R(word):
    encoded_word = ''
    letter_to_number = {'A': '00', 'B': '01', 'C': '02',
                        'N': '13', 'O': '14', 'P': '15',
    if word == '':
        return ''
    else:
        encoded_word += letter_to_number[word[0]]
        return encoded_word + encode_R(word[1:])


def decode(msg, offset):
    result = ''
    number_to_letter = {'00': 'A', '01': 'B', '02': 'C', '0
                        '12': 'M', '13': 'N', '14': 'O', '1
    for i in range(0, len(msg), 2):
        if msg[i:i+2] == '99':
            result += ' '
        else:
            number = str(int(msg[i:i+2]) - offset).zfill(2)
            while int(number) < 0:
                number = str(int(number) + 26).zfill(2)
            result += number_to_letter[number]

    return result


def decode_with_love(msg):
    i = 0
    while True:
        decoded_msg = decode(msg, i)
        i += 1
        if "LOVE" in decoded_msg:
            return decoded_msg


def dna_transcription_I(dna):
    dna_to_rna = {'A': 'U', 'G': 'C', 'T': 'A', 'C': 'G'}
    rna = ''
    for dna_base in dna:
        rna += dna_to_rna[dna_base]
    return rna


def dna_transcription_R(dna):
    if dna == '':
        return ''
    dna_to_rna = {'A': 'U', 'G': 'C', 'T': 'A', 'C': 'G'}
    rna = ''
    dna_base = dna[0]
    rna += dna_to_rna[dna_base]

    return rna + dna_transcription_R(dna[1:])
```

```python
def make_bouquet(shop, number):
    for i in range(len(shop)):
        for j in range(i, len(shop)):
            for k in range(j, len(shop)):
                if (shop[i][1] + shop[j][1] + shop[k][1] == number and
                        (shop[i][0] == 'P' or shop[j][0] == 'P' or shop[k][0] == 'P')):
                    return True
    return False


def minimum_cost(shop, number):
    min_cost = -1
    for i in range(len(shop)):
        for j in range(i, len(shop)):
            for k in range(j, len(shop)):
                if (shop[i][1] + shop[j][1] + shop[k][1] == number and
                        (shop[i][0] == 'P' or shop[j][0] == 'P' or shop[k][0] == 'P')):
                    cost = shop[i][2] + shop[j][2] + shop[k][2]
                    if min_cost == -1 or cost < min_cost:
                        min_cost = cost
    return min_cost

        def paint_area(S, C, D):
            n = 10000
            m = 0

            pink_centers = [(0, S/2 - D/2), (S/2 - D/2, 0),
                            (0, -S/2 + D/2), (-S/2 + D/2, 0)]
            area_pink = (D / 2) ** 2
            area_white = (C / 2) ** 2

            for _ in range(n):
                x = uniform(-S/2, S/2)
                y = uniform(-S/2, S/2)

                is_in_pink = any((x - x_2)**2 + (y - y_2)**2 <=
                                 area_pink for x_2, y_2 in pink_centers)

                is_in_white = x**2 + y**2 <= area_white

                if is_in_pink and not is_in_white:
                    m += 1

            pink_area = (m/n) * (S**2)

            return pink_area
```

Recursion

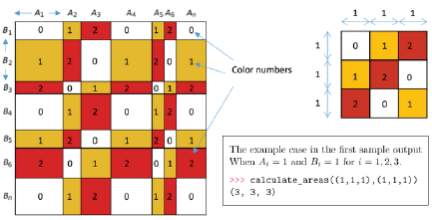
```python
def sum(n):
  if n == 0:
    return 0
  else:
    return n%10 + sum(n//10)
```

Iteration

```python
def fact(n):
  res = 1
  while n > 0:
    res = res * n
    n = n-1
  return res
```

Recursion

```python
def fact(n):
  if n == 0:
    return 1
  else:
    return n * fact(n-1)
```




```python
def calculate_areas(w_list, h_list):
    # Initialize the variables for each color
    white, yellow, red = 0, 0, 0
    # Create a dictionary to store the heights for each color
    color_dict = {i: 0 for i in range(3)}

    # Loop through the height list and add the heights to the corresponding color in the dictionary
    for i in range(len(h_list)):
        color_dict[i % 3] += h_list[i]

    for i in range(len(w_list)):
        # Calculate the offset to ensure the colors are distributed evenly
        offset = i % 3
        # Calculate the area for each color
        white += w_list[i] * color_dict[(i + offset) % 3]
        yellow += w_list[i] * color_dict[(i + 1 + offset) % 3]
        red += w_list[i] * color_dict[(i + 2 + offset) % 3]

    # Return the calculated areas
    return (white, yellow, red)
```

```python
def rna_segment(rna):
    new_rna = ''
    for i in range(len(rna)):
        base = rna[i:i+2]
        if base == 'UU' or base == 'UG':
            new_rna = rna[i:]
            break
    for i in range(0, len(new_rna), 2):
        base = new_rna[i:i+2]
        if base == 'AC' or base == 'AA':
            new_rna = new_rna[:i+2]
            break
    return new_rna
```

- `Acidic` : If there are more acidic amino acids than basic amino acids.
- `Basic` : If there are more basic amino acids than acidic amino acids.
- `Polar` : If neither `Acidic` nor `Basic` and there are more polar amino acids than non-polar amino acids.
- `Neutral` : If it does not fall into either one of the above categories.

Amino	Property	Amino	Property	Amino	Property	Amino	Property
F	*acidic*	S	*polar*	Y	*polar*	Q	*acidic*
L	*non-polar*	P	*acidic*	O	*non-polar*	C	*basic*
M	*basic*	T	*polar*	A	*basic*	R	*basic*

```python
def poly_property(poly):
    poly_property_dict = {'F': 'acidic', 'L': 'non-polar', 'M': 'basic'
                          'T': 'polar', 'Y': 'polar', 'O': 'non-polar',
    result = {'acidic': 0, 'basic': 0, 'polar': 0, 'non-polar': 0}
    for i in range(len(poly)):
        if poly[i] in poly_property_dict:
            result[poly_property_dict[poly[i]]] = result.get(
                poly_property_dict[poly[i]], 0) + 1
    if result['acidic'] > result['basic']:
        return 'Acidic'
    elif result['acidic'] < result['basic']:
        return 'Basic'
    elif result['polar'] > result['non-polar']:
        return 'Polar'
    else:
        return 'Neutral'
```

```python
def auspicious_number(n, bad):
    # Calculate the number of choices for the first digit.
    # The first digit cannot be a 'bad' number.
    # If the number of 'bad' digits is less than 9, add the difference to the total number of choices (9).
    # For example, if 'bad' is [1, 4], there are 7 choices for the first digit (1, 2, 3, 5, 6, 7, 8, 9).
    first_digit_choices = 9 - len(bad)

    # Calculate the number of choices for the remaining digits.
    # The remaining digits can be any number from 0 to 9, excluding the 'bad' numbers.
    # If the number of 'bad' digits is less than 10, subtract it from the total number of choices (10).
    # For example, if 'bad' is [1, 4], there are 8 choices for the remaining digits (0, 2, 3, 5, 6, 7, 8, 9).
    remaining_digits_choices = 10 - len(bad)

    # If 0 is a 'bad' number, add 1 to the number of choices for the first digit.
    # This is because 0 can be used as the first digit if it's not a 'bad' number.
    if 0 in bad:
        first_digit_choices += 1

    # Calculate the total number of possible auspicious numbers.
    # The total is the number of choices for the first digit times the number of choices for the remaining digits.
    # The number of choices for the remaining digits is raised to the power of (n - 1) because we're choosing n - 1 digits.
    total = first_digit_choices * pow(remaining_digits_choices, n - 1)

    return total
```

CS1010E PE1 Cheat Sheet

Brians Tjipto

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