사랑하는 자들아 우리가 서로 사랑하자 사랑은 하나님께 속한 것이니 사랑하는 자마다 하나님으로부터 나서 하나님을 알고 사랑하지 아니하는 자는 하나님을 알지 못하나니 이는 하나님은 사랑이심이라 (요일4:7-8)

Python for Machine Learning
Lecture Notes by idebtor@gmail.com, Handong Global University

NOTE: The following materials have been compiled and adapted from the numerous sources including my own. Please help me to keep this tutorial up-to-date by reporting any issues or questions. Send any comments or criticisms to `idebtor@gmail.com` Your assistances and comments will be appreciated.

__Revisit Iteration__

Iteration means executing the same block of code over and over, potentially many times. For example, the following code contains an iterative work of accumulating a series of addtions. It keeps on adding to sum from 1 to n, inclusively.

In [ ]:
```python
# add a series of numbers from 1 to n.


print('Sum from 1 to {} is {}'.format(n, sum))
```

What would you do if you want to do the same kind of job but with a different number `n`? For example, what do we do if want to add numbers upto 20, 50, 200, or 5000.

Expected Output:

```
Sum from 1 to 10 is 55
Sum from 1 to 50 is 1275
Sum from 1 to 1000 is 500500
```

To satisfy this kind of needs, we introduce the function.

- **DRY** (In-house Programming Principle I)

Function

학습 목표

- 함수에 대해 이해한다.
- 함수를 적절히 사용하여 문제를 해결할 수 있다.

학습 내용

1. 함수 생성하기
2. arguments 이해하기
3. Return Values 이해하기

4. Default 파라미터 이해하기
5. 익명함수(Lambda) 사용하기

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.

Creating a function

Syntax of Function

```
def function_name(parameters):
    """docstring"""
    statement(s)
```

Example:

Define a function called `greet_hello()` that prints `Hello from a function`

```
In [ ]:
```

Calling a function

To call a function, use the function name followed by parenthesis:

Example:

Invoke `greet_hello()` defined above:

```
In [ ]:
```

Passing information as arguments

- Information can be passed into functions as arguments.
- Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.
- The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Example:

Define `greet_hello()` that takes two objects, `fname` , and `lname` , as arguments. Then it prints two arguments with `Hello` in front.

```
In [ ]:
```

In [ ]:
```
greet_hello('Joe', 'Blow')
greet_hello('Joe', 'Adam')
greet_hello('King', 'David')
```

Example:

Define a function called `sum_n_print()` that adds upto an argument `n` and prints its result as shown below:

Sample Run:

```
sum_n_print(10)
sum_n_print(50)
sum_n_print(1000)
```

Expected Output:

```
Sum from 1 to 10 is 55
Sum from 1 to 50 is 1275
Sum from 1 to 1000 is 500500
```

- **NSE** (In-house Programming Principle II - No Side Effect)

In [ ]:
```
def sum_n_print(n):
    None
```

In [ ]:
```
sum_n_print(10)
sum_n_print(50)
sum_n_print(1000)
```

Arbitrary arguments

- If you do not know how many arguments that will be passed into your function, add an `*` before the parameter name in the function definition.
- This way the function will receive **a tuple of arguments**, and can access the items accordingly:

Example:

Define `greet_hello()` such that it takes any number of arguments and prints `Hello` with those names.

Sample Run:

```
greet_hello('Joe', 'Blow', 'David', 'Stones')
```

Expected Output:

```
Hello Joe
Hello Blow
Hello David
Hello Stones
```

In [ ]:
```
def greet_hello(None):
    None
```

In [ ]:
```
greet_hello('Joe', 'Blow', 'David', 'Stones')
```

Keyword arguments

- You can also send arguments with the key = value syntax.
- This way the order of the arguments does not matter.

Example:

Define a function that uses three keyword arguments as shown below:

Sample Run:

```
greet_hello(child = 'David', father = 'Joe', mother = 'Jane')
```

Expected Output:

```
The youngest person is David
```

In [ ]:
```
def greet_hello(None):
    print('The youngest person is', None)
```

In [ ]:
```
greet_hello(child = 'David', father = 'Joe', mother = 'Jane')
```

Arbitrary keyword arguments

- If you do not know how many keyword arguments that will be passed into your function, add two asterisk: `**` before the parameter name in the function definition.
- This way the function will receive a dictionary of arguments, and can access the items accordingly:

Example:

Sample run:

```
greet_hello(father = 'Joe', mother = 'Jane', child = 'David')
```

Expected Output:

```
The youngest person is David
The lovely person is Jane
The respectful person is Joe
```

In [ ]:
```python
def greet_hello(None):
    print('The youngest person is', None)
    print('The lovely person is', None)
    print('The respectful person is', None)
```

In [ ]:
```python
greet_hello(father = 'Joe', mother = 'Jane', child = 'David')
```

Default Parameter Value

- The following example shows how to use a default parameter value.
- If we call the function without argument, it uses the default value:

Example:

Define `greet_hello()` such that it take a default parameter value `Korea` .

Sample Run:

```
greet_hello('England')
greet_hello()
greet_hello('Heaven')
```

Expected Output:

```
Are you from England?
Are you from Korea?
Are you from Heaven?
```

In [ ]:
```python
def greet_hello(None):
    print('Are you from ' + country + '?')
```

In [ ]:
```python
greet_hello('England')
greet_hello()
greet_hello('Heaven')
```

Return Values

To let a function return a value, use the return statement:

Example:

Define a function `my_favorites` which takes a list and returns the first and last elements in it. Add a code to be able to handle an empty list gracefully.

Sample Runs:

```
first, last = my_favorites(["apple", "banana", "Romans", "Genesis"])
print(first, last)
```

```
first, last = my_favorites([])
print(first, last)
first, last = my_favorites()
print(first, last)
```

Expected Output:

```
apple Genesis
None None
None None
```

In [22]:
```
def my_favorites(None):
    pass
```

In [26]:
```
first, last = my_favorites(["apple", "banana", "Romans", "Genesis"])
print(first, last)
first, last = my_favorites([])
print(first, last)
first, last = my_favorites()
print(first, last)
```

```
apple Genesis
None None
None None
```

More Examples

Example 1: sum_to_n(n)

- Generalize this capability of summing up to `n`. In other words, write a function, `sum_to_n(n)`, such that it returns 55 when `sum_to_n(10)` is invoked.
- Complete the following code cells to sum the integers from `1` to `n` inclusively.
- Avoid any `print()` statement unless the function name include something `print`, `output` or `show` etc since we honor the NSE principle.

```
Sum from 1 to 10 is 55
Sum from 1 to 100 is 5050
Sum from 1 to 1000 is 500500
Sum from 1 to 10000 is 50005000
```

- NSE (In-house Programming Principle II)

In [ ]:
```
# function to sum upto n inclusively.
def sum_to_n(n):
    None
```

In [ ]:
```
# For testing
for n in None:
    print('Sum from 1 to {} is {}'.format(n, sum_to_n(n)))
```

Pay attention that maximum value in `range()` is `n + 1` to make `i` equal to `N` on the last step.

To iterate over a decreasing sequence, we can use an extended form of `range()` with three arguments - `range(start, end, step)`. When omitted, the `step` is implicitly equal to `1`. However, can be any non-zero value. The loop always includes `start` and excludes `end` during iteration:

Example 2: count_alpha()

Write a function `count_alpha()` which takes a string and returns two values or counts of lowercase and uppercase alphabets in the string, respectively.

Use two `string` class methods, `islower()` and `isupper()` to distinguish the cases.

Sample Run:

```
lower, upper = count_alpha("Hello World!")
print(lower)
print(upper)
```

Output:

```
8
2
```

In [ ]:
```
def count_alpha(str):
    None
    return lower, upper
```

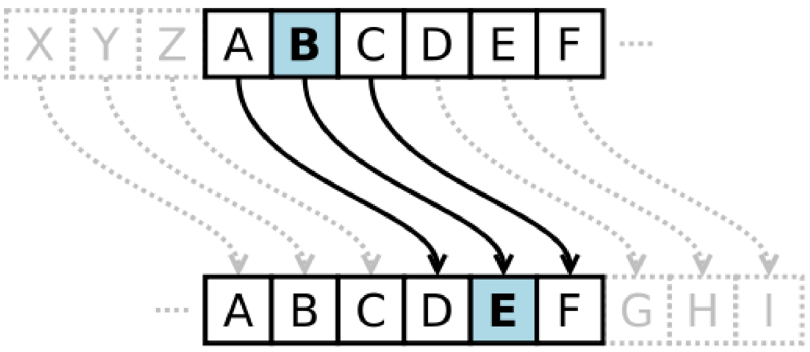
In [ ]:
```
lower, upper = count_alpha("Hello World!")
print(lower)
print(upper)
```

Example 3: Caesar Cypher

카이사르 암호(Caesar cipher) 또는 시저 암호는 암호학에서 다루는 간단한 치환암호의 일종이다. 실제로 로마의 황제 카이사르는 이 카이사르 암호를 사용하기도 했다. 카이사르 암호는 암호화하고자 하는 내용을 알파벳별로 일정한 거리만큼 밀어서 다른 알파벳으로 치환하는 방식이다. 예를 들어 3글자씩 밀어내는 카이사르 암호로 'COME TO ROME'을 암호화하면 'FRPH WR URPH'가 된다. 여기서 밀어내는 글자 수는 암호를 보내는 사람과 함께 정해 더 어려운 암호를 만들 수 있다. 이런 카이사르 암호는 순환암호라고 한다. 카이사르는 'RUSQHUVKBVEHQIIQIYDQJEH' 라는 암호를 받았다. 해독하면 BECAREFULFORASSASINATOR, 암살자를 조심하라는 뜻이된다. 카이사르는 이 암호를 해독하지 못해 암살자에게 암살당하고 말았다..

카이사르 암호는 약 기원전 100년경에 만들어져 로마의 장군인 카이사르가 동맹군들과 소통하기 위해 만든 암호이다

카이사르 암호는 단순하고 간단하여 일반인도 쉽게 사용할 수 있지만, 철자의 빈도와 자주 사용되는 단어와 형태를 이용하면 쉽게 풀 수 있다는 단점이 있다.



Step 1:

Write a function, `next_alpha(ch, n)` which takes an ascii character `ch` and an int `n`. Then the function returns a character at some fixed positions `n` from the current letter `ch` in the alphabet.

Sample Run:

```
for ch in "Hello World! zzz":
    print(next_alpha(ch, 1), end=' ')
```

Output:

```
I f m m p   X p s m e !   a a a
```

Hint: Use two constant strings `ascii_lowercase` and `ascii_uppercase` which are defined in `string` module.

Note: This coding is YET to be continued in the following homework. Some bugs are left intentionally.

In [ ]:

```
import string
lower = string.ascii_lowercase
upper = string.ascii_uppercase

print(lower)
print(upper)
```

In [ ]:

```
def next_alpha(ch, n):
    if len(ch) != 1:
        return ch
    if ch.islower():
        None
    elif ch.isupper():
        None
    return ch
```

In [ ]:

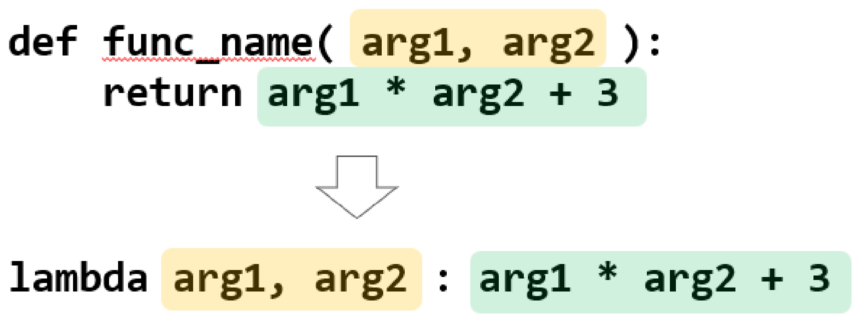
```
for ch in "Hello World!":
    print(next_alpha(ch, 1), end=' ')
```

In [ ]:

```
for ch in "Hello World! zzz":
    print(next_alpha(ch, 1), end=' ')
```

Lambda function 익명함수

한번 사용하고 다시 사용하지 않는 간단한 함수는 **익명함수** (**lambda함수**)로 만들 수 있습니다. 람다함수는 함수이름이 없으며, `return` 을 사용하지 않고 자동으로 결과를 반환할 수 있습니다. 아래 그림을 참조하십시오.



```
def func_name( arg1, arg2 ):
    return arg1 * arg2 + 3

lambda arg1, arg2 : arg1 * arg2 + 3
```

예를 들면, x를 인자로 받아 5를 더해서 반환하는 이름없는 lambda함수는 다음과 같이 정의할 수 있습니다.

```
lambda x: x + 5
```

람다함수는 정의하면서 동시에 사용할 수 있으며, 아래와 같이 람다 함수에 3을 입력한 것이고 8이 출력됩니다.

```
(lambda x: x + 5)(3)
8
```

람다함수도 객체이기 때문에 정의와 동시에 변수에 저장할 수 있습니다. 다시 사용할 수 있습니다.

```
func = lambda x: x + 5
func(3)
8
```

정리하면, 람다 함수는 다음과 같은 특징이 있습니다.

- 람다는 단일 표현식만 허용한다.
- `if` , `while` 같은 문장은 허용하지 않는다.
- `sort()` 와 같은 함수와 함께 사용하는 것이 일반적인 용도이다.

Example:

주어진 리스트를 다음과 같은 조건으로 정렬하십시오.

1. 알파벳 순으로 정렬하십시오.
2. 문자열의 길이에 따라 정렬하십시오. 힌트: lambda함수와 len()을 사용하십시오.

다음 sort()함수를 참조하거나 help(str)를 실행하여 참조하십시오.

```
|  sort(self, /, *, key=None, reverse=False)
|      Sort the list in ascending order and return None.
```

```
 |
 |      The sort is in-place (i.e. the list itself is modified) and
stable (i.e. the
 |      order of two equal elements is maintained).
 |
 |      If a key function is given, apply it once to each list item
and sort them,
 |      ascending or descending, according to their function values.
 |
 |      The reverse flag can be set to sort in descending order.
```

Sample Run: by alphabetical order

```
items = ['faith', 'abba', 'sin', 'zzzz', 'foo']
['abba', 'faith', 'foo', 'sin', 'zzzz']
```

In [ ]:
```
items = ['faith', 'abba', 'sin', 'zzzz', 'foo']
items.sort()
items
```

Sample Run: by length

```
items = ['faith', 'abba', 'sin', 'zzzz', 'foo']
['foo', 'sin', 'abba', 'zzzz', 'faith']
```

In [ ]:
```
items = ['faith', 'abba', 'sin', 'zzzz', 'foo']
```

Example:

- 문자열에 포함된 a 의 갯수에 따라 정렬하십시오. 힌트: lambda함수와 str클래스의 count() 메소드를 사용하십시오.

In [ ]:
```
items = ['faith', 'abba', 'sin', 'zzzz', 'foo']
items.sort(key = len)
items
```

Sample Run: by user-defined lambda function

```
items = ['faith', 'abba', 'sinai', 'aaaa', 'foo']
items.sort(key = None)
print(items)
```

Expected Output:

```
items = ['faith', 'abba', 'sinai', 'aaaa', 'foo']
['foo', 'faith', 'sinai', 'abba', 'aaaa']
```

In [2]:
```
asfun = lambda x: x.count('a')
items = ['faith', 'abba', 'sinai', 'aaaa', 'foo']
items.sort(key = asfun)
print(items)
```

```
['foo', 'faith', 'sinai', 'abba', 'aaaa']
```

Exercise

Teen Age

- Write a function, `is_teen(age)` , which returns if `age` is a teen age.
- Write a code to test `is_teen()` while calling it from `age = 9` to `age = 21` .
- Print the result as shown below:

```
The age 9 is not a teen
The age 10 is not a teen
The age 11 is not a teen
The age 12 is not a teen
The age 13 is a teen
The age 14 is a teen
The age 15 is a teen
The age 16 is a teen
The age 17 is a teen
The age 18 is a teen
The age 19 is a teen
The age 20 is not a teen
The age 21 is not a teen
```

Step 1:

Code is_teean(age) function first

In [ ]:
```
# Replace `None` with the logic expression you developed above.
def is_teen(age):
    return None
```

Step 2:

Write a test code to produce the output shown above.

Method 1: Using a list of ages

In [ ]:
```
# test is_teen()
for age in None
    None
```

Method 2: Using `range()`

In [ ]:
```
for age in None
    None
```

Leap Year

A leap year is exactly divisible by 4 except for century years (years ending with 00). The century year is a leap year only if it is perfectly divisible by 400.

For example,

```
2017 is not a leap year
1900 is a not leap year
2012 is a leap year
2000 is a leap year
```

To test a leap year interactively, you may code it as shwon below:

In [ ]:
```
# Python program to check if the input year is a leap year or not
year = int(input("Enter a year: "))
if (year % 4) == 0:
    if (year % 100) == 0:
        if (year % 400) == 0:
            print("{0} is a leap year".format(year))
        else:
            print("{0} is not a leap year".format(year))
    else:
        print("{0} is a leap year".format(year))
else:
    print("{0} is not a leap year".format(year))
```

Step 1:

Coding `is_leap_year()` function: Now, complete the following `is_leap_year()` function that returns `True` if `year` is a leap year, `False` otherwise.

In [ ]:
```
def leap_year(year):
    return None
```

Step 2:

Coding to test `is_leap_year()` to have the following output shown below:

```
100 No, it is not.
200 No, it is not.
1990 No, it is not.
2000 Yes, it is a leap year
2002 No, it is not.
2004 Yes, it is a leap year
2010 No, it is not.
2100 No, it is not.
```

In [ ]:
```
# test leap_year()
for year in None:
    None
```

Acronym

An **acronym** is a word formed by taking the first letters of the words in a phrase and then making a word from them. For example, RAM is an acronym for random access memory. Write a function `acronym()` that takes a phrase (i.e., a string) as input and then returns the acronym for that phrase. Note: The acronym should be all uppercase, even if the words in the phrase are not capitalized.

Sample Run:

```
print(acronym('Graphics processing unit'))
print(acronym('handong global university'))
print(acronym('United Nations'))
print(acronym('Random Access Memory'))
GPU
HGU
UN
RAM
```

Solution: In this problem we iterate over the words of the phrase `s` and accumulate the first letter in every word. So we need to break the phrase into a list of words using the string `split()` method and then iterate over the words in this list. We will add the first letter of every word to the accumulator string `ans`.

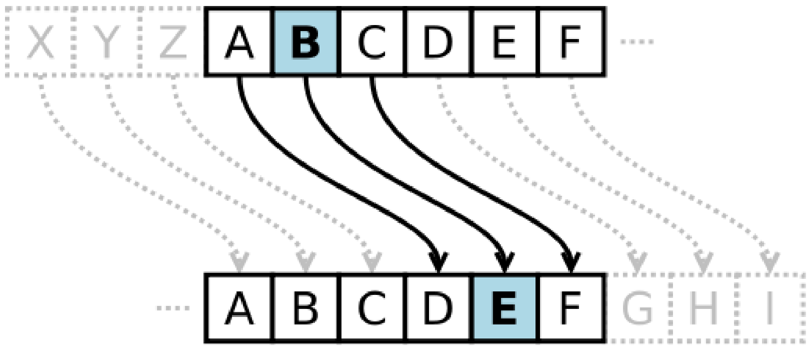
In [ ]:

```
def acronym(s):
    pass

if __name__ == '__main__':
    print(acronym('Graphics processing unit'))
    print(acronym('handong global university'))
    print(acronym('United Nations'))
    print(acronym('Random Access Memory'))
```

Caesar Cypher

Write a function, `next_alpha(ch, n)` which takes an ascii character `ch` and an int `n`. Then the function returns a character at some fixed positions `n` from the current letter `ch` in the alphabet.



Step 1:

Fix the following code such that it works with the following cases: **Hint:** Use two constant strings `ascii_lowercase` and `ascii_uppercase` which are defined in `string` module.

Hint: Perform a modulo operation (%) to prevent the index from exceeding the length of alphabet.

Sample run 1:

```
for ch in "Hello World! zzz":
    print(next_alpha(ch, 1), end=' ')
```

Output:

```
I f m m p   X p s m e !   a a a
```

Sample run 2:

```
for ch in "aaa BBB ccc Hello World! XXX yyy zzz":
    print(next_alpha(ch, 3), end=' ')
```

Output:

```
d d d   E E E   f f f   K h o o r   Z r u o g !   A A A   b b b   c c
c
```

In [ ]:

```
import string
lower = string.ascii_lowercase
upper = string.ascii_uppercase

def next_alpha(ch, n):
    if len(ch) != 1:
        return ch
    if ch.islower():
        idx = lower.find(ch)
        return lower[None]
    elif ch.isupper():
        idx = upper.find(ch)
        return upper[None]
    return ch
```

In [ ]:

```
for ch in "Hello World! zzz":
    print(next_alpha(ch, 1), end=' ')
```

In [ ]:

```
for ch in "aaa BBB ccc Hello World! XXX yyy zzz":
    print(next_alpha(ch, 3), end=' ')
```

Step 2:

Write a Caesar cypher function that encoded by shifting n to the right in alphabet.

Sample run 1:

```
encoded = cypher("aaa Hello World! zzz", 2)
print(encoded)
```

Output:

```
ccc Jgnnq Yqtnf! bbb
```

Sample run 2:

```
encoded = cypher("aaa Hello World! zzz", 3)
print(encoded)
```

Output:

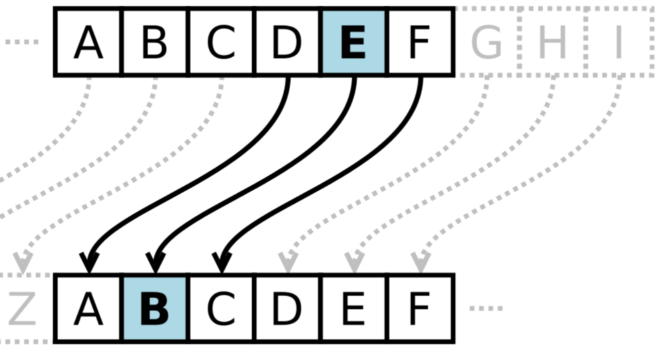
```
ddd Khoor Zruog! ccc
```

In [ ]:
```
def cypher(str, n):
    None
    return None
```

In [ ]:
```
encoded = cypher("aaa Hello World! zzz", 2)
print(encoded)
```

In [ ]:
```
encoded = cypher("aaa Hello World! zzz", 3)
print(encoded)
```

Step 3: Decypher



Modify cypher() to have a default optional argument `coded = False`. The function works as `cypher` function as usual, but it works as `decypher` function if `coded = True`.

Sample Run:

```
hello = "aaa Hello World! zzz"
print(hello)
encoded = cypher(hello, 2)
print(encoded)
decoded = cypher(encoded, 2, coded = True)
print(decoded)
```

Output:

```
aaa Hello World! zzz
ccc Jgnnq Yqtnf! bbb
aaa Hello World! zzz
```

In [ ]:
```
def cypher(None):
    None
    return None
```

In [ ]:
```
hello = "aaa Hello World! zzz"
```

```
print(hello)
encoded = cypher(hello, 2)
print(encoded)
decoded = cypher(encoded, 2, coded = True)
print(decoded)
```

Factorial

Implement function `factorial()` that takes as input a nonnegative integer and returns its factorial. The **factorial** of a nonnengative integer `n`, denoted $n!$, is define in this way:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1 & \text{if } n > 0 \end{cases} \tag{1}$$

So, $0! = 1$`, $3! = 6$, and $5! = 120$.

Sample Run:

```
print(factorial(0))
print(factorial(3))
print(factorial(5))
1
6
120
```

Solution:

We need to multiply (accumulate) integer $1, 2, 3, \ldots, n$. The accumulator `ans` is initialized to 1, the identity for multiplication. Then we iterate over sequence $2, 3, 4, \ldots, n$ and multiply `ans` by each number in the sequence:

In [ ]:
```
def factorial(n):
    pass

if __name__ == '__main__':
    print(factorial(0))
    print(factorial(1))
    print(factorial(3))
    print(factorial(5))
```

Euler's number e

The number e, known as Euler's number, is a mathematical constant approximately equal to 2.71828. Euler's number (e) play's a crucial role in mathematics. It represents the basis of the exponential functions and logarithms.

It is known that the precise value of e is equal to this infite sum:

$$\frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \ldots \tag{1}$$

An infinite sum is impossible to compute. We can get an approximation of e by computing the sum of the first few terms in the infinite sum.

For example, $e_0 = \frac{1}{0!} = 1$ is a very rough approximation for e.

Then next sum, $e_1 = \frac{1}{0!} + \frac{1}{1!} = 2$, is still a rough approximation for e.

Then next sum, $e_2 = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} = 2.5$, looks better.

The next few sums that we are heading in the right direction:

$$e_3 = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} = 2.6666... \tag{2}$$
$$e_4 = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} = 2.7083... \tag{3}$$

Now, because, $e_4 - e_3 = \frac{1}{4!} > \frac{1}{5!} + \frac{1}{6!} + \frac{1}{7!} + \ldots$, we know that e_4 is within $\frac{1}{4!}$ of the actual value for e. This give us a way to compute an approximation of e that is guaranteed to be within a given range of the true value of e.

Write a function `approx_e()` that takes as a input a float value `max_error` and returns a value that approximates constant e to within `max_error` . You will do this by generating the sequence of approximation $e_0, e_1, e_2, \ldots$until the difference between the current approximation and the previous one is no greater than `max_error` .

Check that the approximation that returns from `approx_e()` is really smaller than `max_error` . Use `math.e` for the correct value of e.

Sample Run:

```
max_error = 0.01
me = approx_e(max_error)
print(' approx_e:', me)
print('    error:', math.e - me)
print('max_error:', max_error)

max_error = 0.000000001
me = approx_e(max_error)
print(' approx_e:', me)
print('    error:', math.e - me)
print('max_error:', max_error)
```

Expected Output:

```
 approx_e: 2.7166666666666663
    error: 0.0016151617923787498
max_error: 0.01
 approx_e: 2.7182818284467594
    error: 1.2285727990501982e-11
max_error: 1e-09
```

Solution:

- We start by assigning the first approximation(1) to `prev` and the second (2) to `curr` .
- The `while` loop condition is then `curr - prev > max_error` .
- If the condition is true, then we need to generate new values for `prev` and `curr` .

- The value of `curr` becomes `prev`, and the new `curr` value is then `prev + 1/factoral(???)`. What should ??? be?
- In the first iteration, it should be 2 because the third approximation is the value of the second $\frac{1}{2!}$. In the next iteration, it should be 3, then 4, and so on.

In [ ]:
```python
import math

def approx_e(max_error):
    """returns approximation of e within max_error"""
    prev = 1          # approximation 0
    curr = 2          # approximation 1
    i = 2             # index of next approximation

    pass

    return curr
```

학습 정리

1. 함수 생성하기
2. arguments 이해하기
3. Return Values 이해하기
4. Default 파라미터 이해하기
5. 익명함수(Lambda) 사용하기

참고자료

- Python Functions

"Everything is permissible" - but not everything is beneficial. "Everything is permissible" - but not everything is constructive. 1 Corinthians 10:23