|  |
| --- |
|  |
| \\ | একটা ব্যাকস্ল্যাশ |
| \' | সিঙ্গল কোট (') |
| \" | ডাবল কোট (") |
| \a | বেল |
| \b | ব্যাকস্পেইস |
| \f | ফর্মফিড |
| \n | লাইন ব্রেক |
| \t | ট্যাব |
| \v | ভার্টিক্যাল ট্যাব |

print('\a') # for create bip sound

print('mahmud \bhossain') # back-space

print('\f') #

print('mahmud\_\nhossain') #line break

print('mahmud\thossain') # tab

print('mahmud\vhossain') #virtical tab

# 

## ইনপুট

>>>input('type a name:')

type a name:

type a name:mahmud

## স্ট্রিং অপারেশন

print('hello'+'mahmud')

hellomahmmud

তাই বলে কোন নাম্বারের সাথে স্ট্রিং যোগ করা যাবে না,

1 + '2' + 3 + '4'

Traceback (most recent call last):

## ****রিপিটেশন (Repetition)****

print('hello\_world'\*2)

print('5'\*2)

print(2\*'5')

print(2\*'mahmud')

hello\_worldhello\_world

55

55

Mahmudmahmud

## ****স্ট্রিং ফরম্যাটিং****

নন স্ট্রিং ডাটার সাথে স্ট্রিং টাইপের ডাটাকে যুক্ত করে সুন্দর স্ট্রিং আউটপুট তৈরি করতে format মেথড ব্যবহার করা হয়। format মেথডের মধ্যের প্রত্যেকটি আর্গুমেন্ট দিয়ে এর সামনে থাকা স্ট্রিং এর মধ্যের প্লেস হোল্ডার গুলোকে রিপ্লেস করা হয়। প্লেস হোল্ডার গুলো {} এর সাথে ইনডেক্স বা নাম ব্যবহার করে ডিফাইন করা হয়।

mahmud = "My self score on PHP: {0}, Python: {1}, Java: {2}, Swift: {3}". format(6, 6.5, 5, 6)

print(mahmud)

print('{2}, {1}, {0}'.format('a', 'b', 'c'))

print('{0}{1}{0}'.format('abra', 'cad'))

My self score on PHP: 6, Python: 6.5, Java: 5, Swift: 6

c, b, a

abracadabra

মেথডের মধ্যে নাম ওয়ালা আর্গুমেন্ট পাঠিয়ে এবং স্ট্রিং এর মধ্যের প্লেস হোল্ডার গুলোতে সেই নামে সেগুলোকে ব্যবহার করেও কাজ করা যায়

message = "If x = {x} and y = {y}, then x+y = {z}".format(x = 20, y = 300, z = 20+300)

print(message)

num1=2

num2=2

print(f"{num1}+{num2} = {num1+num2}")

If x = 20 and y = 300, then x+y = 320

2+2 = 4

## Replace

মেথডের মাধ্যমে একটি সাব স্ট্রিং কে খুঁজে সেখানে অন্য কিছু রিপ্লেস করা যায়। যেমন উপরের উদাহরণে - ME রিপ্লেস করে world বসানো হয়েছে।

## Join

print("frut, ".join(["apple", "orange", "pineapple","cool"]))

applefrut, orangefrut, pineapplefrut, cool

## **startswith, endswith**

মেথডের মাধ্যমে কোন একটি ব্যাক্যর শুরু বা শেষ নির্দিষ্ট কোন সাবস্ট্রিং দিয়ে হয়েছে কিনা তা চেক করা যায়।

print("This is a my sentence.".startswith("This"))

print("This is a sentence.".startswith("is"))

print("This is a sentence.".endswith("sentence."))

print("This is a sentence.".endswith("my"))

True

False

True

False

## upper() and lower()

Upper () মেথড স্ট্রিংয়ের সবগুলো ক্যারেক্টারকে uppercase এ পরিবর্তিত করে।lower() মেথড ট্রিংয়ের সবগুলো ক্যারেক্টারকে lowercase এ পরিবর্তিত করে।

print("this is a sentence.".upper())

print("This is A sentence.".upper())

print("AN ALL CAPS SENTENCE".lower())

print("AN all CAPS SENTENCE".lower())

THIS IS A SENTENCE.

THIS IS A SENTENCE.

an all caps sentence

an all caps sentence

## split

মেথড হচ্ছে join মেথডের উল্টো। অর্থাৎ একটি বাক্যেকে নির্দিষ্ট কোন সেপারেটর এর সাপেক্ষে ভেঙ্গে একটি লিস্ট তৈরি করা যায় এই মেথডের মাধ্যমে।

print("m,a,h,m,u,d".split(", "))

['m,a,h,m,u,d']

## টাইপ কনভার্সন/type change

ডাটাটাইপ কনভার্সন বলতে ভ্যারিয়েবল কে এক টাইপ থেকে অন্য টাইপ এ কনভার্ট করা বুঝায়। একে টাইপ কাস্টিং ও বলা হয়ে থাকে। পাইথনে টাইপ কাস্টিং এর জন্যে কিছু বিল্টইন ফাংশন বানানো আছে।

print (int("123"))

print (int(12.3))

print (str(123))

print (str(12.3))

print (int("123"))

print (float(12.3))

স্ট্রিং থেকে ইন্টেজার এ কনভার্ট এর সময় খেয়াল রাখতে হবে স্ট্রিং এ যাতে কোনো নননিউমেরিক ক্যারেকটার না থাকে।

print (int("123ab"))

## ভ্যারিয়েবল

ভ্যারিয়েবল হচ্ছে কম্পিউটার মেমোরিতে তৈরি হওয়া ছোট ছোট বাক্সের মতো যার ভেতর যে কোন কিছু জমা করে রাখা যায়। যখন আমরা ভ্যারিয়েবল ডিক্লেয়ার করি তখন কম্পিউটার সেই ভ্যারিয়েবলের জন্য কিছু নির্দিষ্ট মেমোরি নির্ধারন করে দেয়। প্রতিটি ভ্যারিয়েবল এর মেমোরি অ্যাড্রেস ইউনিক হয়। প্রোগ্রামের প্রয়োজনে ওই ভ্যারিয়েবল তথা নাম সম্পন্ন মেমোরি লোকেশনে ভ্যালু জমা করে রাখা যায়। আবার প্রয়োজনের সময় সেই নাম ব্যবহার করে ওই লোকেশনের ভ্যালুকে অ্যাক্সেস করা যায় এবং কাজে লাগানো যায়।

একটি ভ্যারিয়েবলের মধ্যে কোন ভ্যালু জমা রাখার জন্য একটি সমান (=) চিহ্ন ব্যবহার করা হয়।

যদি আমরা কোন ভ্যালু কোন একটা ভ্যারিয়েবলে স্টোর করি (সমান চিহ্ন দিয়ে) এবং এন্টার প্রেস করি তখন কিন্তু পরের লাইনে আউটপুট আসবে না। বরং সমান চিহ্নের ডান পাশের ভ্যালুটি সমান চিহ্নের বাম পাশের ভ্যারিয়েবলে জমা হয়ে যাবে যেটাকে আমরা print deya call korbo.

x = 7

print(x)

print(x + 3)

7

10

## রি-অ্যাসাইনমেন্ট

x = 7.5

print(x)

x="hello"

print(x + "3")

7.5

hello3

এখানে প্রথমে x এর মান হিসেবে একটি ফ্লট জমা রাখা হয়েছে এবং সাধারণ ভাবেই প্রিন্ট করে তার ভ্যালু পাওয়া গেছে। কিন্তু পরের লাইনে সেই একই x এর মধ্যে একটি স্ট্রিং জমা রাখা হয়েছে এবং সেটিকেও পরবর্তী প্রিন্ট ফাংশনের মাধ্যমে অ্যাক্সেস করে স্ক্রিনে প্রিন্ট করা গেছে। একে ভ্যালু রি-অ্যাসাইনমেন্ট বলা হয়। অর্থাৎ, একটি ভ্যারিয়েবলের মধ্যে একাধিক বার নতুন নতুন ভ্যালু জমা রাখা যায় এবং সর্বশেষ স্টোর করা ভ্যালুটিই ওই ভ্যারিয়েবলের মধ্যে জমা থাকে (আগের ভ্যালুটি মুছে যায়)।

### পাইথনে ভ্যারিয়েবল লেখার সময় কিছু নিয়ম মেনে ভ্যারিয়েবল ডিফাইন করা হয়।

* ভ্যারিয়েবলের নাম অবশ্যই এক শব্দের হতে হবে। অর্থাৎ এরকম ভাবে ভ্যারিয়েবল লেখা যাবে নাঃ

my variable = 10

* প্রথম অক্ষর অবশ্যই একটি alphabetic letter(uppercase or lowercase) অথবা underscore ( \_ ) হতে হবে । যেমনঃ nafis, a, b, \_variable লেখা যাবে ভ্যারিয়েবল হিসেবে কিন্তু 1name, @nafis, 7a, %b এই ভাবে লেখা যাবে না । প্রথম অক্ষর ছাড়া পরে letter, underscore, number ব্যবহার করা যাবে। যেমনঃ variable1, my\_variable, not\_Very\_Good\_Name10
* পাইথন Case Sensitive অর্থাৎ a = 4 এবং A = 4 একই ভ্যারিয়েবল না।
* পাইথনের কিছু reserved কী-ওয়ার্ড আছে, এগুলো ব্যবহার করা যাবে না । যেমনঃ if, else, elif, for, while, break, continue, except, as, in, is, True, False, yield, None, def, del, class ইত্যাদি।

## Birthday calculator

নিচের উদাহরণে আমরা দেখবো কিভাবে সেই ফাংশন ব্যবহার করে এবং একটি ভ্যারিয়েবল ব্যবহার করে ইউজার এর ইনপুট মেমোরিতে জমা রাখা যায় এবং পরবর্তীতে ব্যবহার করা যায়,

user\_input = input("Enter your birth year: ")

age = 2020 - float(user\_input) #here float,int etc use

print("You are " + str(age) + " years old!")

You are 26.0 years old!

## ইনপ্লেস অপারেটর

a = 3

a += 2

print(a)

5

### কমন ইনপ্লেইস অপারেটর:

* +=
* -=
* \*=
* /=
* %=

শুধুমাত্র নাম্বার বাদেও অন্যান্য টাইপের ক্ষেত্রেও ইন প্লেইস অপারেটর ব্যবহার করা যায়

language = "Python"

language += "3"

print(language)

## কন্ট্রোল স্ট্রাকচার

### if স্টেটমেন্ট

num = 12

if num > 5:

    print("Bigger than 5")

    if num <= 47:

        print("Between 6 and 47")

Bigger than 5

Between 6 and 47

### else স্টেটমেন্ট

num = 12

if num > 5:

    print("Bigger than 5")

else:

    print("small then 5")

Bigger than 5

শুধুমাত্র if এর সাথে ব্যবহার বাদেও else কে ব্যবহার করা যায় for এবং while লুপের সাথেও।

for mah in range(10):

    print(mah)

else:

    print("not")

### if else চেইন

num = 7

if num == 5:

    print("Number is 5")

else:

    if num == 11:

        print("Number is 11")

    else:

        if num == 7:

            print("Number is 7")

        else:

            print("Number isn't 5, 11 or 7")

Number is 7

মজার ব্যাপার হচ্ছে এরকম if else if এর চেইনকে একটু সংক্ষেপে elif দিয়েও লেখা যায়

### Elif স্টেটমেন্ট

num = 7

if num == 5:

    print("Number is 5")

elif num == 11:

    print("Number is 11")

elif num == 7:

    print("Number is 7")

else:

    print("Number isn't 5, 11 or 7")

Number is 7

### ****টারনারি অপারেটর****

a = 100

b = 200

if (a >= 100 and a < 200):

    print(b)

else:

    print(300)

#we can rewrite the above function given bellow

a = 100

b = 200 if (a >= 100 and a < 200) else 300

print(b)

ধরে নেই, প্রথমেই a এর মান 100 অ্যাসাইন করা হয়েছে। এরপর b এর জন্য একটি মান অ্যাসাইন করতে চাচ্ছি। সেটা হতে পারে 200 অথবা 300. তো, আসলে কোনটা হবে সেটি নির্ধারণ করার জন্য একটি কন্ডিশন বসিয়েছি। কন্ডিশনটি হচ্ছে - if (a >= 100 and a < 200) অর্থাৎ a এর মান ১০০ থেকে বড় বা সমান এবং ২০০ থেকে ছোট হলে এই কন্ডিশনটি সত্য হবে আর তখন b এর মান হিসেবে 200 অ্যাসাইন হবে। কন্ডিশনটি মিথ্যা হলে b এর মধ্যে 300 ঢুকবে।

status = 1

msg = "Logout" if status == 1 else "Login"

print(msg)

### বুলিয়ান লজিক

if স্টেটমেন্টের জন্য জটিল কন্ডিশন তৈরির ক্ষেত্রে বুলিয়ান লজিক ব্যবহৃত হয়ে থাকে। অর্থাৎ একটি if স্টেটমেন্ট যদি একাধিক কন্ডিশনের উপর নির্ভর করে সেখানে আমরা বুলিয়ান লজিক ব্যবহার করতে পারি। আগেও বলা হয়েছে, পাইথনে and, or এবং not এই তিন ধরণের বুলিয়ান অপারেটর আছে।

and এই অপারেটর দুটো আর্গুমেন্ট নিয়ে যাচাই করে এবং সত্য হয় যখন দুটো আর্গুমেন্টই সত্য হয়।

### and

a= 1 == 1 and 2 == 2

print(a)

true

a= 1 == 1 and 2 == 3

print(a)

false

### or

a= 1 == 1 or 2 == 3

print(a)

true

### not

a= not 1 == 1

print(a)

false

### is,isnot

a= 1 is 1 and 2 is 2

print(a)

a= 1 is not 1 and 2 is 2

print(a)

### while লুপ

যতক্ষণ সেই while লুপের কন্ডিশন সত্য থাকে ততক্ষণ পর্যন্ত এর আওতাভুক্ত কোড রান করতেই থাকে। আর যখন কন্ডিশনটি মিথ্যা হয়ে যায় তখন while লুপের বাইরে গিয়ে প্রোগ্রামের পরবর্তী স্টেটমেন্ট গুলো রান করা শুরু করে।

i=10

while i<100: #jotokhon I er man 100 teke soto takbe ,totokhon run korbe

    print(i)

    i=i+1

m=5

while m<2:# jotokhon m er man 2 teke soto hobe,totokhon run hobe, but initial m=5

    print(m)

    m=m+1

### Break

কিন্তু এরকম জোড় করে বন্ধ করা পছন্দ না হলে জানিয়ে রাখা ভালো যে - প্রোগ্রাম্যাটিক্যালিও while লুপের কাজ যেকোনো সময় কন্ডিশনের পরোয়া না করেও বন্ধ করা সম্ভব। এর জন্য শুধু লিখতে হবে break.

i=10

while i<100:

    print(i)

    i=i+1

    if i==50:

        print("ok")

        break

### continue

ধরা যাক, একটি লুপের মধ্যে আমরা বেশ কিছু কাজ করার স্টেটমেন্ট লিখেছি এবং চাচ্ছি যে লুপ যতক্ষণ সত্য থাকে (ধরে নেই ১০০ বার) ততক্ষণ এর মধ্যেকার কাজ গুলো বার বার হোক। কিন্তু, এমনও তো হতে পারে যে, সেই ১০০ বারের মধ্যে বিশেষ কয়েকবার আমরা সেই পুরো কাজটা করতে চাই না কিন্তু নির্ধারিত ১০০ বারই লুপকে কাজ করাতে দিতে চাই; তাহলে কি করতে পারি? এর জন্যই আছে continue. একটি while লুপের মধ্যে যখনই continue এক্সিকিউট হবে তখনিই লুপের মধ্যে থাকা এর পরের কোড গুলো এক্সিকিউট হবে না এবং লুপের কন্ট্রোল একদম শুরুতে চলে যাবে,

i=10

while i<100:

    print(i)

    i=i+1

    if i==50:

        print("ok, start again")

        continue

another example

i = 0

while True:

   i = i +1

   if i == 2:

      print("Skipping 2")

      continue

   if i == 5:

      print("Breaking")

      break

   print(i)

print("Finished")

## লিস্ট

পাইথনে ৬ ধরণের বিল্ট ইন টাইপ আছে। সেগুলো হচ্ছে - numeric, sequence, mapping, class, instance এবং exception. সব থেকে ব্যাসিক ডাটা স্ট্রাকচারটি হচ্ছে sequence. এর প্রত্যেকটি এলিমেন্টের জন্য একটি নাম্বার অ্যাসাইন করা হয় যাকে ইনডেক্স বা পজিশন বলা যায়. পাইথনে আবার ৩ ধরণের ব্যাসিক sequence টাইপ আছে যেগুলো হচ্ছে list, tuple, এবং range.

দুটো স্কয়ার ব্র্যাকেট এবং এর মধ্যে কমা দিয়ে আলাদা আলাদা এলিমেন্ট যুক্ত করে একটি লিস্ট তৈরি করা যায়।

name=["mah","do","did","done"]

print(name[0])

print(name[2])

### list হিসেবে string

str = "Hello world!"

print(str[6])

### লিস্ট অপারেশন(replace,append,power)

#### Replace

replace\_num = [1, 2, 3, 5]

replace\_num[3] = 100

print(replace\_num)

#### Append and power

append\_list = [1, 2, 3]

print(append\_list + [100, 200, 300])

print(append\_list \* 3)

### মাল্টিপল প্যারামিটার হ্যান্ডলিং | আর্বিটরারি আর্গুমেন্ট লিস্ট/\*args and \*\*kwargs

মনে করুন, আপনি make\_sum ফাংশনটিতে অনেকগুলো প্যারামিটার পাঠাতে চাচ্ছেন যেমন, 10, 20, 30 ... ইত্যাদি। যদি আপনি make\_sum (a, b) হিসেবে ডিক্লেয়ার করেন তাহলে দুইটার বেশি প্যারামিটার পাঠাতে পারবেন না।

ef multi\_sum(x,y):

    mul=x+y

    print(mul)

multi\_sum(10,20)

30

### Use (\*args)

but using (\*args) we can pass unlimited parameter, it output show a tuple

def coco(\*args):

    print(args)

coco(1,2,3,4,5,6)

(1, 2, 3, 4, 5, 6)

def mah(\*args):

    for x in args:

        print(x)

mah(1,1,1,1,1,)

11111

But not work below program

def mah(\*args):

    for x in args:

        print(x)

mah(a=1,b=1,c=1,d=1)

TypeError: mah() got an unexpected keyword argument 'a'

### Use (\*\*kwargs)

By using this formate we can pass/show output parameter and argument both in a carly braket/second braket

def colo(\*\*kwargs):

    print(kwargs)

colo(a=2,b=3,c=4)

{'a': 2, 'b': 3, 'c': 4}

### পাইথনে \* এর অর্থ

\* এর আর্গুমেন্টে ভ্যালু Tuple হিসেবে প্যাকড থাকে। এর মানে \* দিয়ে প্যারামিটার ডিক্লেয়ার করলে আমরা যেকোন সংখ্যক পজিশনাল আর্গুমেন্ট পাস করতে পারি. শুরুতে make\_sum মাত্র দুইটা আর্গুমেন্ট নিলেও পরবর্তীতে আমরা প্যারামিটারে \* বসিয়ে দিলাম তখন সে অনেকগুলো আর্গুমেন্ট পাস করতে পারছে।

### পাইথনে \*\* এর অর্থ

আমরা চাইলে ফাংশনের প্যারামিটারে ডাবল অ্যাস্টেরিস্কস বসিয়েও ডিক্লেয়ার করতে পারি। ডাবল স্টারের মানে হল যেকোন সংখ্যক named parameter থাকতে পারে। এই মানগুলো ডিকশনারি হিসেবে প্যাকড থাকে।

সিঙ্গেল অ্যাস্টেরিস্কস ব্যবহার করলে আমরা নেমড আর্গুমেন্ট পাস করতে পারব না। তাই আমাদের এসব ক্ষেত্রে ডাবল অ্যাস্টেরিস্কস ব্যবহার করতে হবে, যেমন

def print\_dict(\*\*kwargs):

    print(kwargs)

print\_dict(a=1, b=2, c=3)

{'a': 1, 'c': 3, 'b': 2}

def print\_dict(\*\*kwargs):

    for args in kwargs:

        print("{0} : {1}".format(args, kwargs[args]))

print\_dict(a=1, b=2, c=3)

চাইলে আমরা মিক্সড ভ্যারিয়েডিক আর্গুমেন্ট পাঠাতে পারি। মানে একই ফাংশনে তিন ধরণের আর্গুমেন্ট, তবে খেয়াল রাখতে হবে প্যারামিটারগুলো এমন ভাবে ডিফাইন করা হয় যেন প্রথমে সাধারণ প্যারামিটার তারপরে সিঙ্গেল অ্যাস্টেরিস্কের প্যারামিটার এবং অবশেষে ডাবল অ্যাস্টেরিস্কস এর প্যারামিটার থাকে। মানে আমাদের অবশ্যই ক্রম মানতে হবে এইক্ষেত্রে।

def print\_all(a, \*args, \*\*kwargs):

    print(a)

    print(args)

    print(kwargs)

print\_all(10, 20, 30, 50, b=5, c=10)