পাইথনে হাতেখড়ি

আহমাদ আল-সাজিদ

একটি প্রোগ্রামের আউটপুট দেখানোর জন্য বেশ কয়েকটি পদ্ধতি আছে, যা ডাটা পড়ার উপযোগী ফরম্যাটে প্রিন্ট করা বা ফাইলে রাইট করে ভবিষ্যতে ব্যবহারের জন্য রেখে দেয়া যায়। পাইথনের ওপর ধারাবাহিক এ লেখায় সে বিষয়ে আলোচনা করা হয়েছে।

এতদিন আউটপুট দেখানোর জন্য print ফাংশন ব্যবহার করা হয়েছে। প্রায়ই আমাদের বিভিন্ন ধরনের ফরম্যাটের প্রয়োজন হয়। দুইভাবে ফরম্যাটের কাজ করা যায়। প্রথমত স্ট্রিংয়ের সব ধরনের কাজ নিজেকে বলে দিতে হবে, স্ট্রিংকে প্রথমে ভাগ করে নিতে হবে, এরপর দরকার অনুযায়ী আবার জুড়ে দিতে হবে। এর জন্য স্ট্রিংয়ের বেশ কিছু ফাংশন আছে, যা পরে আলোচনা করা হয়েছে। দ্বিতীয় পদ্ধতি হচ্ছে str.format() মেথড ব্যবহার করা।

পাইথনে কোনো ভ্যালুকে স্ট্রিংয়ে রূপান্তর করার জন্য দুটি ফাংশন আছে– repr() এবং str()। str() ফাংশনের কাজ হচ্ছে ভ্যালুকে এমনভাবে রিপ্রেজেন্ট করে রিটার্ন করা যেটি অনেকটা পড়ার উপযুক্ত, যেখানে repr()-এর কাজ হচ্ছে ইন্টারপ্রেটারের বোঝার উপযোগী করে ডাটা রিপ্রেজেন্ট করা। যেসব অবজেক্টের ক্ষেত্রে আমাদের উপযোগী করে ডাটা রিপ্রেজেন্ট করা যায় না, সে ক্ষেত্রে repr() এবং str() একই ভ্যালু রিটার্ন করে। নাম্বার, লিস্ট বা ডিকশনারি টাইপ ডাটা স্ট্রাকচারের ক্ষেত্রে ফাংশন দুইটি একইভাবে ডাটা রিপ্রেজেন্ট করে। উদাহরণস্বরূপ–

```
>>> s = ‘Hello world’
>>> str(s)
‘Hello world’
>>> repr(s)
“‘Hello world’”
>>> str(1/7)
‘0.14285714285714285’
>>> x = 10 * 3.25
>>> y = 200 * 200
>>> s = ‘x is ‘+repr(x)+’, y
is ‘+repr(y)
>>> print(s)
x is 32.5, y is 40000
```

repr() ফাংশন ব্যবহার করলে স্ট্রিং কোট এবং ব্যাকস্ল্যাশ স্ট্রিংয়ের সাথে যুক্ত হয়, সাধারণত যেগুলো স্ট্রিংয়ের সাথে যুক্ত হয় না।

```
>>> hello = ‘hello, world\n’
>>> hellos = repr(hello)
>>> print(hellos)
‘hello, world\n’
```

repr() ফাংশনে যেকোনো পাইথন অবজেক্ট আর্গুমেন্ট হিসেবে পাঠানো যায়।

```
>>> repr((x, y, (‘spam’, ‘eggs’)))
“(32.5, 40000, (‘spam’, ‘eggs’))”
```

১ থেকে ৫ পর্যন্ত নাম্বারের স্কয়ার এবং কিউব প্রিন্ট করার জন্য দুইটি পদ্ধতি ব্যবহার করতে পারেন।

```
>>> for x in range(1,6):
print(repr(x).rjust(3),repr(x
*x).rjust(4),end=”)
  print(repr(x*x*x).rjust(5))
  1   1    1
  2   4    8
  3   9   27
  4  16   64
  5  25  125
>>> for x in range(1,6):
print(‘{0:3d}{1:4d}{2:5d}
’.format(x,x*x,x*x*x))
  1  1   1
  2  4   8
  3  9  27
  4 16  64
  5 25 125
```

উপরের উদাহরণে স্ট্রিং অবজেক্টের str.rjust() মেথডের ব্যবহার দেখানো হয়েছে। এটা স্ট্রিংয়ের বাম পাশে দরকার মতো স্পেস যোগ করে স্ট্রিংকে রাইট জাস্টিফাই করে দেয়। একইভাবে লেফট জাস্টিফাই এবং সেন্টার করার জন্য যথাক্রমে str.ljust() এবং str.center() মেথড ব্যবহার করা হয়। এই মেথডগুলো কোনো কিছু লিখে না বা প্রিন্ট করে না। এগুলো শুধু নতুন একটি স্ট্রিং তৈরি করে রিটার্ন করে। আরেকটি মেথড str.zfill(), যেটা নিউমেরিক স্ট্রিংয়ের বাম পাশে প্রয়োজন অনুযায়ী শূন্য বসিয়ে দিতে পারে। এই মেথড প্লাস এবং মাইনাস সাইন বুঝতে পারে।

```
>>> ‘12’.zfill(5)
‘00012’
>>> ‘-12’.zfill(5)
‘-0012’
```

যদি স্ট্রিংয়ের দৈর্ঘ্য str.zfill()-এর প্যারামিটারের থেকে বেশি হয়ে যায়, তাহলে আর সামনে শূন্য বসবে না।

```
>>> ‘123456’.zfill(5)
‘123456’
```

str.format() মেথডের বেসিক ব্যবহার নিম্নরূপ–

```
>>> print(‘we are {}’.for-
mat(‘student’))
we are student
```

অর্থাৎ ব্র্যাকেট এবং এর ভেতরের ক্যারেক্টারগুলো (যাদের বলা হয় ফরম্যাট ফিল্ডস) str.format() মেথডের ভেতরে প্লাস করা অবজেক্ট দিয়ে পরিবর্তিত হয়। ব্র্যাকেটের ভেতরে নাম্বার দিয়ে অবজেক্টের পজিশন বলে দেয়া যায়।

```
>>> print(‘{0} and
{1}’.format(‘spam’,’eggs’))
spam and eggs
>>> print(‘{1} and
{0}’.format(‘spam’,’eggs’))
eggs and spam
```

যদি str.format() মেথডে কিওয়ার্ড আর্গুমেন্ট ব্যবহার করা হয়, তাহলে তাদের ভ্যালুগুলো আর্গুমেন্ট নামে রেফার করা হয়।

```
>>> print(‘this {item} is
{quality}’.format(item =
‘book’, quality = ‘good’))
this book is good
```

পজিশনাল এবং কিওয়ার্ড আর্গুমেন্ট একসাথে সহজেই ব্যবহার করা যায়।

```
>>> print(‘the story of {0}, {1}
and {third}’.format(‘Bill’,’Mike’,
third = ‘Georg’))
the story of Bill, Mike and Georg
```

ascii(), str() এবং repr()-এর বদলে !a, !s এবং !r যুক্ত করে ভ্যালুকে ফরম্যাট করার আগেই কনভার্ট করে নেয়া যায়।

```
>>> import math
>>> print(‘approx PI :
{}’.format(math.pi))
approx PI : 3.14159265359
>>> print(‘approx PI :
{!r}’.format(math.pi))
approx PI : 3.141592653589793
```

‘:’ এবং ফরম্যাট স্পেসিফায়ার ব্যবহার করে কীভাবে ভ্যালু ফরম্যাট হবে তার ওপরে আরও বেশি নিয়ন্ত্রণ আনা যায়। নিচের উদাহরণে পাই-এর মান তিন ঘর পর্যন্ত প্রিন্ট করে দেখা যায়

```
>>> print(‘approx PI :
{0:.3f}’.format(math.pi))
approx PI : 3.142
```

‘:’-এর পরে ইন্টেজার ভ্যালু পাস করলে সেটা ওই ফিল্ডকে ততগুলো ঘর পর্যন্ত প্রশস্ত করে দেবে। এটি ব্যবহার করে টেবিল তৈরি করলে তা দেখতে তুলনামূলক বেশি সুন্দর হয়।

```
>>> table = {‘Sjoerd’: 4127,
‘Jack’: 4098, ‘Dcab’: 8637678}
>>> for name, phone in
table.items():
        print(‘{0:10} ==>
{1:10d}’.format(name,
phone))
Dcab      ==>   8637678
Sjoerd    ==>      4127
Jack      ==>      4098
```

যদি আমাদের হাতে অনেক বড় ফরম্যাটিং স্ট্রিং থাকে, যেটিকে ভাঙতে চাই না, তাহলে সুবিধাজনক ভ্যারিয়েবলগুলোকে পজিশনের বদলে নাম দিয়ে রেফার করা যায়। এটা সহজেই ডিকশনারি পাস করে []-এর মাধ্যমে কীগুলোকে অ্যাক্সেস করা যায়।

```
>>> print(‘Jack: {0[Jack]:d};
Sjoerd: {0[Sjoerd]:d}; Dcab:
{0[Dcab]:d}’.format(table))
Jack: 4098; Sjoerd: 4127;
Dcab: 8637678
```

এই কাজটি টেবিলটিকে ‘**’ নোটেশনের মাধ্যমে কিওয়ার্ড আর্গুমেন্ট হিসেবে পাঠিয়েও করা যায়।

```
>>> print(‘Jack: {Jack:d};
Sjoerd: {Sjoerd:d}; Dcab:
{Dcab:d}’.format(**table))
Jack: 4098; Sjoerd: 4127;
Dcab: 8637678
```

এবার ফাইল নিয়ে আলোচনা করা যাক। কোনো ডাটা পরে ব্যবহার করতে চাইলে তা ফাইলে রাইট করে রাখা যায়। এর জন্য open() মেথড ব্যবহার করা হয়। এর প্রধান ব্যবহার হয় মূলত দুইটি আর্গুমেন্ট ব্যবহার করে, open(filename,mode)।

প্রথম আর্গুমেন্ট দিয়ে যে ফাইল নিয়ে কাজ করা হবে তাকে নির্দিষ্ট করে দেয়া হয়। পরের আর্গুমেন্টে ফাইলের ওপরে কী ধরনের অপারেশন চালানো হবে তা বলে দেয়া হয়। যেমন– ফাইল রিড করতে r, রাইট করতে w, অ্যাপেন্ডিংয়ের জন্য a ইত্যাদি লেখা হয়।

সাধারণত ফাইল টেক্সট মোডে ওপেন হয়। এ ছাড়াও বাইনারি মোডে ওপেন করা যায়। তবে বাইনারি মোডে ওপেন করলে সাবধান থাকতে হবে। কেননা, এতে মাঝে মাঝে ভিন্ন ভিন্ন অপারেটিং সিস্টেমের জন্য ফাইল করাপ্ট হয়ে যেতে পারে।

ফাইলে রাইট করার উদাহরণ নিম্নরূপ–

```
f = open(‘workfile’, ‘w’)
>>> f.write(‘this is first line\n’)
19
>>> f.write(‘this is second line\n’)
20
>>> f.write(‘this is third line\n’)
```

প্রতি লাইনে কতগুলো ক্যারেক্টার প্রিন্ট হচ্ছে তা দেখাচ্ছে। আবার যদি ফাইল থেকে কিছু পড়তে চাইলে নিচের মতো

(বাকি অংশ ৬৯ পৃষ্ঠায়)

করে করতে পারেন।

```
>>> f = open(‘workfile’, ‘r’)
>>> f.read()
‘this is first line\nthis is second line\nthis is third line\n’
>>> f.read()
‘’
```

read() মেথড ব্যবহার করে পুরো ফাইল একসাথে রিড করা যাই। দ্বিতীয়বার একই কমান্ড দিলে ফাঁকা স্ট্রিং দেখাবে।

প্রতিটি লাইন আলাদা করে পড়তে চাইলে নিচের মতো করতে হবে।

```
>>> f = open(‘workfile’, ‘r’)
>>> f.readline()
‘this is first line\n’
>>> f.readline()
‘this is second line\n’
>>> f.readline()
‘this is third line\n’
>>> f.readline()
‘’
```

লুপের মাধ্যমেও এটা করতে পারবেন।

```
>>> for line in f:
    print(line, end=’’)
this is first line
this is second line
this is third line
```

সব লাইনগুলোকে একটি লিস্টে রাখতে চাইলে list(f) ba f.readlines() ব্যবহার করেই তা করতে পারবেন।

যখন ফাইলের ওপর অপারেশন চালানো শেষ হবে, তখন f.close() ফাংশন ব্যবহার করতে হবে। তাহলে এটি সিস্টেমের যে রিসোর্সগুলো open()-এর মাধ্যমে ধরে রেখেছিল তা ছেড়ে দেবে। এ জন্য ভালো পদ্ধতি হচ্ছে with কিওয়ার্ড ব্যবহার করা। এর মাধ্যমে ফাইল ওপেন করলে কাজ শেষে নিজে থেকেই ফাইল ক্লোজ করে দেয়।

```
>>> with open(‘workfile’, ‘r’) as f:
    read_data = f.read()
>>> f.closed
True
```

কজ

ফিডব্যাক : ahmadalsajid@gmail.com

▶