آموزش پایتون از پایه : توابع و ماژول ها

لینک مطلب :

http://code.tutsplus.com/tutorials/python-from-scratch-functions-and-modules--net-21045

دوباره به آموزش پایتون از پایه خوش آمدید . در درس قبل ، یاد گرفتید که چطور از متغیر ها و ساختار های کنترلی برای ذخیره و دستکاری داده ها استفاده کنید . مطمئن شوید که درس قبل را مرور کرده اید اگر به آن احتیاج دارید

رونوشت (Transcript)

در آموزش امروز ، ما میریم به توابع یک نگاهی بیندازیم – چی هستن ، و چطور کار میکنن ، و چطور توابع خودمون رو بسازیم . ما همچنین چند تا از توابع داخلی (built-in functions) را مرور میکنیم ، و چطور از آنها بهترین استفاده را بکنیم . در آخر ، ما یک نگاهی به ساخت و وارد کردن ماژول ها می اندازیم .

توابع : نوشتن توابع خود

توابع یک قدم مهم برای پوشش هستند ، وقتی که پیچیدگی بیشتری به برنامه خود اعمال میکنیم . اگر متغیر یک کانتینر دارای نام برای داده هاست ، بعد تابع یک کانتینر دارای نام برای بلوک کدی که قابلیت اجرا را دارد در زمانی که به آن احتیاج داشته باشیم . این به درد بخوره اگر شما برنامه ای داشته باشید که یک عملیات خاصی رو به کرار اجرا میکنه . به جای اینکه اون تکه کد را کپی و پیست کنیم در هر جایی که لازمه ، شما به راحتی میتونید یک تابع بنویسید تا این کار رو انجام بده .

تابع یک کانتینر دارای نام برای بلوکی از کد است .

دو نام تابع وجود دارد : اون هایی که شما خودتون می نویسید و در کد قرار می دهید ، و اون هایی که به طور پیش فرض در پایتون هستند ، که پردازه های معمول رو انجام میدن ، مثل تبدیل یک عدد به رشته ، یا پیدا کردن طول یک رشته .

ما الان میریم به نوشتن یک تابع ساده نگاهی بیاندازیم ، و ثابت کنیم که چقدر اونها میتونن در دنیای واقعی مفید باشند . بعد ، ما یک نگاهی به بعضی از مهم ترین توابع داخلی می اندازیم .

یک مثال ساده

بیایید فرض کنیم ما میخواهیم یک اسکریپت برای سبد خرید بسازیم که قیمت تمام وسایل داخل سبد شما رو میگیره ، و بعد همه ی آنها را با هم جمع میکنه تا یک قیمت کلی بدست بیاد . برای نوشتن تابعی که دو تا قیمت رو میگیره ، اون ها رو باهم جمع میکنه ، و بعد خروجی رو به نمایش میگذاره یا پرینت (print) میکنه ، باید چیزی شبیه زیر باشه :

```
01  #our two costs to add up
02  cost1 = 15
03  cost2 = 20
04
05
06  def sumCart():
07      totalCost = cost1 + cost2
08      print totalCost
09
10  sumCart()
```

برای تعیین تابع ، ما باید از کله کلیدی def استفاده کنیم ، و بعد نام تابع . بعد ، دو تا پرانتز تایپ میکنیم (با این ها بعدا بر میگردیم) ، و بعد یک دو نقطه . در آخر ، و تمام کدی که میخواهیم باید در داخل این تابع قرار بدیم و شامل این تابع باشند ، درست مثل while ، if و for .

برای اجرای کد داخل تابع ، ما نام تابع رو تایپ میکنیم ، با دو تا پرانتز در آخر . اگر کد رو اجرا کنید ، میبینید که ۳۵ رو نمایش میده ، که خروجی درست هستش .

آرگومان ها (Arguments)

این عالیه ، ولی در حال حاضر توابع ما کمی انعطاف ناپذیرند . هر چیزی درباره توابع در آنها به صورت سخت کد شده . برای مثال ، بیایید تابع sumCart را در جایی دیگه استفاده کنیم ، در یک قسمت دیگه برنامه ، اما به جای جمع قیمت اول و دوم ، ما میخوایم و تا قیمت دیگه که در یک متغیر قرار دارند را اظافه کنیم . ما باید یک تابع کاملا متفاوت بنویسیم ، اگرچه تابع جدید دقیقا مثل تابع قبلی باشه ، با تفاوت نام های متغیر ها — این به سختی میتونه راه حل ما باشه .

برای حل این مشکل ، ما از آرگومان ها استفاده می کنیم ، و این دلیلی هستش که پرانتز ها اونجا هستند . یک آرگومان راهی برای انتقال داده داخل یک تابع است وقتی که ما نمیدونیم

کدوم متغیر یا متغیر هایی داده ها در آنها می باشند . اگر گیج کننده هست ، بیایید مثالی که گفتم رو ببینیم . ما دو تا قیمت دیگه اظافه میکنیم : قیمت ۳ و قیمت ۴ .

حالا ، ما میریم که دو تا آرگومان برای دو تا آیتمی که میخوایم با هم جمع کنیم می افزاییم .

آرگومان ها داخل پرانتز معرفی میشن ، و هر آرگومان یک نام دارد ، با یک کاما یا ویرگول جدا میشن . یک آرگومان به عنوان یک منبع موقت برای داده ای که وارد کرده اید عمل میکنه وقتی که تابع در حال اجراست .

```
01  cost1 = 15
02  cost2 = 20
03  cost3 = 5
04  cost4 = 10
05
06  def sumCart(item1, item2):
07      totalCost = item1 + item2
08      print totalCost
09
10  sumCart(cost1, cost2)
```

وقتی ما تابع sumCart رو صدا می زنیم ، دو تا متغیری که وارد کرده ایم (قیمت ۱ و قیمت ۲) در آرگومان های آیتم ۱ و آیتم ۲ قرار میگیرند . این همیشه به ترتیبی که آرگومان ها را معرفی کردید قرار میگیره . یا با کلمات دیگر ، اولین متغیری که شما وارد کردید به آرگومان اول تلق داره ، و دومی به آرگومان دوم ، و به همین ترتیب برای هر چند تای دیگر .

دقیقا چه اتفاقی می افته وقتی که تابع اجرا میشه اینه که اون دو تا آرگومان به متغیر های محلی تبدیل میشن و مقدار های متغیر ها متعلق است که در تابع وارد کرده بودید وقتی که صداش میزنیم — در این مثال مقدار قیمت ۱ در متغیر محلی آیتم ۱ قرار میگیره ، و مقدار قیمت ۲ در متغیر محلی آیتم ۲ قرار میگیره . این یعنی شما به راحتی می تونید در تابع از آنها استفاده کنید .

طور دیگری که میشه به این ها نگاه کرد اینه که ، وقتی یک متغیر را به عنوان یک آرگومان وارد میکنید ، هر جایی که نام آن آرگومان در تابع نمایان بشه ، با اون متغیری که وارد کرده بودید عوض میشه . پس ، دی این مثال ، هر جایی که آیتم ۱ در داخل تابع نوشته بشه ، با قیمت ۱ عوض میشه ، و دقیقا همین طور برای آیتم ۲ و قیمت ۲ .

برای ثابت کردن اینکه هر عددی بخواهید میتونید وارد کنید ، اگر این کد را اجرا کنید ، الان شما باید جمع قیمت ۳ و قیمت ۴ رادریافت کنید .

```
cost1 = 15
cost2 = 20
cost3 = 5
cost4 = 10

def sumCart(item1, item2):
    totalCost = item1 + item2
    print totalCost

sumCart(cost3, cost4)
```

پیش فرض های آرگومان (Argument Defaults)

توجه داشته باشید که یک تابع میتونه هر چند تا آرگومان رو قبول کنه ، اما به یاد داشته باشید : وقتی تابع را صدا میزنید ، شما باید به همان تعداد آرگومان وارد کنید که معین کردید ، در غیر این صورت ، خطا دریافت می کنید . یک راه برای این هست : شما میتونید یک مقدار پیش فرض به هر آرگومانی معین کنید ، که در مواقعی استفاده میشه که شما مقداری رو ندارید .

با توجه به کدمون ، بیایید تصور کنیم که میخواهیم آیتم دوم برای قیمت ۵ باشه اگر به تابع یک مقدار متفاوت ندیم .

```
cost1 = 15
cost2 = 20
cost3 = 5
cost4 = 10

def sumCart(item1, item2 = 5):
    totalCost = item1 + item2
    print totalCost

sumCart(cost1)
sumCart(cost3, cost4)
```

اگر این کد رو اجرا کنیم ، میبینید که ، در صدا زدن اول ، خروجی ۲۰ هستش ، که از قیمت ۱ + 5 میاد . این اصلا بر روی رفتار تاثیر نمیزاره وقتی که هر دو آرگومان تغذیه شدن .

بازگرداندن مقدار یا مقدار بازگشت داده شده (Returning Values)

یک خاصیت نهایی در تابع ها هستش که ما امروز بررسی میکنیم : بازگرداندن مقدار . ما یاد گرفتیم که یک تابعی بنویسیم که براش فرقی نمیکنه چه مقداری دریافت میکنه ، اما چی میشد اگه ما میخواستیم که جواب رو در متغیری به نام قیمت کل (total cost) ذخیره کنیم و روی

صفحه پرینت نکنیم ؟ شاید کاری که باید انجام بدیم اینه که قیمت دو تا آیتم اول رو در یک متغیر ذخیره کنیم ، و همین کار رو با دو تا آیتم دیگر کنیم و آنها را در متغیر دوم قرار دهیم . با این تابع فعلی ، ما نمیتونیم اینکار رو به صورت قابل فهم و خوندن بنویسیم .
درست مثل وارد کردن آرگومان ها در تابع ها که مجبور نباشیم کد سخت (hard-code) کنیم که داده از کجا میاد ، ما میتونیم همین کار رو با خروجی انجام بدیم . با یک مثال بهتون نشان میدم – در تابع قبل ، من میخوام کلمه print رو با کلمه return عوض کنم :

```
01  cost1 = 15
02  cost2 = 20
03  cost3 = 5
04  cost4 = 10
05
06
07  def sumCart(item1, item2):
08      totalCost = item1 + item2
09      return totalCost
10
11  cart1 = sumCart(cost1, cost2)
12  cart2 = sumCart(cost3, cost4)
13
14  print cart1
15  print cart2
```

همون طور که معلومه ، تابع دیگه جواب رو برای ما به نمایش در نمیاره (پرینت نمیکنه) – کاری که میکنه اینه که مقدار متغیر قیمت کل رو به هر جایی که تابع رو صدا زدیم بر میگردونه . با این راه ، میتونیم از جواب در هر متنی که میخواهیم استفاده کنیم . حالا ، در برنامه قبلیمون ، این قابل استفاده نیست ، به دلیل اینکه جایی رو برای ذخیره مشخص نکردیم ، پس من همچنین روشی رو که ما تابع رو صدا می زدیم رو هم عوض کردم .
ما باید به پایتون یک متغیری رو معرفی کنیم که جواب رو در اون ذخیره کنه ، و بعد ما به راحتی میریم که اون متغیر رو تنظیم کنیم مساوی جمله ای که تابع رو صدا میزنه . وقتی که تابع یک مقدار رو برمیگردونه ، اون مقدار در متغیری که ما معین کردیم ذخیره میشه .

توابع : داخلی (Functions Built-in)

ما یاد گرفتیم چطور توابع خود را بسازیم ، و چطور از آن ها استفاده کنیم ، اما بعضی از عملیات ها هستند که به کرار لازمند و پایتون خودش قرار داده تا در هر برنامه ای استفاده بشن . برای صدا زدنشون مثل صدا زدن توابع خودمون عمل میکنیم ولی نباید اول آن ها را معین کنیم . حال به

بررسی مهمترین این توابع می پردازیم ، اما مطمئن بشید که نگاهی به مستندات پایتون بیاندازید که هزاران توابع مفید دیگر هستند که ما الان وقت پوشش آن ها را نداریم .

به متتندات پایتون برای توابع سر بزنید

()str

اول ، نگای به یکی از پر استفاده ترین توابع پایتون می اندازیم : تابع رشته . در برنامه نویسی مواقع زیادی به این بر میخورید که متغیری با مقدار عددی دارید ، یا نوع دیگری از داده ، اما نیاز دارید که اون رو به رشته تبدیل کنید و کاری رو باهاش انجام بدهید – بیشتر برای نمایش اون روی صفحه نمایش .

مثلا ما یک عدد در یک متغیر داریم ، و میخواهیم آن را به نمایش بگذاریم (پرینت کنیم) ، با پیام (عدد هست :) و بعد عدد . اگر بنویسیم :

```
number = 10
print 'The number is ' + number
```

... بعد ما با یک خطا مواجه میشیم ، چون چیزی که از پایتون میخواید جمع بستن یک عدد و رشته با همه – که اصلا معنی نداره . کاری که باید انجام بدین تبدیل عدد ۱۰ به رشته ۱۰ هستش . اگر اینکار رو انجام بدیم ، پایتون میفهمه کاری که ما میخواهیم انجام بدیم پیدا کردن مجموع نیست ، بلکه قاطی کردن دو رشته با هم است .

اینجا جایی است که تابع str() به کار میاد . اون یک مقدار رو قبول میکنه ، و یک رشته رو برای نمایش اون برمیگردونه . در اینجا شما یک عدد رو بهش میدید ، اون به سادگی عدد رو برمیگردونه البته با فرمت رشته .

```
number = 10
print 'The number is ' + str(number)
```

حالا که عدد رو در یک تابع رشته قرار دادیم ، کد اجرا میشه . تابع رشته حتما نباید عدد بگیره میتونه انواع داده رو دریافت کنه مثل بولین .

```
bool = True
print 'The value is ' + str(bool)
```

تابع رشته نزدیک ترین رشته ای که برای مقدار صحیح میتونه پیدا کنه رو بر میگردونه ، این به

معنی اینه که به بهترین نحو میتونیم خروجی بگیریم .

Len()

یک کار مهم دیگر برای رشته ها توانایی اندازه گیری طول آنهاست ، و ، دوباره ، پایتون یک تابع داخلی برای اینکار دارد . بیایید رشته (Hello World) رو سعی کنیم که طولش رو پیدا کنیم . برای اینکار ، ما به تابع داخلی **طول len()** نیاز داریم ، که این کلمه مخفف length هستش . تمام کاری که باید انجام بدیم دادن رشته مورد نظر به اون هستش ، و تعداد حروف رو بهمون بر میگردونه .

```
string = 'Hello World'
print len(string)
```

تابع طول بیشتر از اینا استحقاق دارد . کاری که میکنه در حقیقت شمارش تعداد آبجکتی هست که شما بهش میدید – اگر بهش رشته بدید بعدش تعدادشون رو میگه ، شما همچنین میتونین بهش لیست list یا تیوپِل tuple بدهید ، برای مثال ، و اون تعداد آبجکت های داخلشون رو برمیگردونه .

Int()

شما بعضی اوغات اعداد 10.6 یا 3.896 رو بهتون میدن ، که عدد صحیح نیستند ، اما شما عدد صحیح آنها را میخواهید . تابع داخلی برای تبدیل به عدد صحیح int() نام دارد ، و تغریبا قابل حدس زدن کار میکند . تابع عدد صحیح تبدیل شده رو برمیگردونه . دقت کنید که این تابع عدد ورودی رو به نزدیک ترین عدد صحیح گرد نمیکنه – این تابع به راحتی تمام عدد های بعد اعشار رو از بین میبره و عدد صحیح و خالص رو بر میگردونه . پس عدد ورودی 10.6 میشه 10 نه 11 . همین طور ، 3.25 میشه 3 .

```
number = 10.6
print int(number)
```

تابع int همچنین میتونه رشته رو به نوع داده عدد صحیح تبدیل کنه ، برای مثال ، خروجی همون ۱۰ میمونه .

```
number = '10'
print int(number)
```

اگر چه ، مراقب باشید ، چون ، وقتی که از یک رشته به یک عدد صحیح تبدیل میکنه ، تابع

عدد صحیح با اعداد بعد از اعشار کاری نداره . اگر فراهم کنید :

```
number = '10.6'
print int(number)
```

یک خطا دریافت می کنید ، چون ورودی یک عدد صحیح نبوده ، پس این تابع نمیدونسته چیکارش کنه .

Range()

در آخر ، ما یک نگاه سریع به تابع محدوده می اندازیم : این تابع وقتی کار های پیچیده با پایتون انجام می دهید لازم است ، و ، چون در الان زیاد قابل استفاده نیست ، ارزش نگاه انداختن و شناخت چگونگی کار اون مهمه . تابع محدوده یک تابعی هستش که شما برای ساختن لیستی از اعداد با یک پترن مشخص میسازید . راحت ترین مثال این است که شما یک لیست شامل همه ی اعداد از 0 تا 10 رو میخواهید . به جای نوشتن همه ی آنها به صورت دستی ، میتونید از تابع محدوده برای نوشتن آنها روی یک خط استفاده کنید .

تابع محدوده عدد صحیح رو به عنوان پارامتر قبول میکنه ، و در اینجا ، ما 11 رو ورد میکنیم . تابع از 0 شروع میکنه و یک لیست تا یک عدد کمتر از عددی که وارد کرده اید میشمره .پس اینجا ، اگر 11 قرار بدیم ، تابع ما اعداد 0 تا 10 رو ذخیره میکنه :

```
numbers = range(11)
print(numbers)
```

چی میشد اگه میخواستیم اعداد رو از ۵ تا ۱۰ نمایش بدیم ؟ خوب ، ما میتونیم یک آرگومان دیگه از نقطه شروع در جمله خود اضافه کنیم ، پس این اعداد 5 تا 10 رو به نمایش میگذاره :

```
numbers = range(5, 11)
print(numbers)
```

در آخر ، چی میشد اگر فقط اعداد فرد رو خواستیم به نمایش در بیاریم ؟ خوب ، این تابع یک مقدار سوم رو هم قبول میکنه ، که مقداری هست که در هر مرحله بالا بره . پس اگر از 1 شروع کنیم ، هر قدم رو طو تا و آخر محدوده رو 11 قرار بدیم ، ما باید تمام اعداد فرد زیر 10 رو بدست بیاریم :

```
numbers = range(1, 11, 2)
print(numbers)
```

برای نوشته آخر ، هر یک از اعداد میتونن منفی هم باشن ، پس اگز خواستیم از 10 تا 1 برعکس بشمریم ، ما میتونیم هر مرحله رو به −1 تغییر دهیم . مقدار شروع از 10 و تا محدوده 0 مثل زیر :

```
numbers = range(10, 0, -1)
print(numbers)
```

توجه کنید که تابع محدوده فقط با اعداد صحیح سر و کار داره ، پس مطمئن بشید که فقط اعداد صحیح درش وارد کنید مگر نه با خطا مواجه میشید .

ماژول ها (Modules) – ساخت ماژول های خود

خب ، حالا که توابع رو بررسی کردیم ، بریم سراغ موضوع دوم درس امروز : ماژول ها . اگر توابع گروه هایی از کد هستند ، ماژول ها گروه هایی از توابع هستند . خیلی بیشتر از این ها هستند ولی برای الان همین که گفتم رو بدونید کافیه .

همون طور که قبلا گفتیم ، شما بدون مرتب سازی کد هاتون توی یک تابع نمیتونید برنامه های پیچیده بسازید . و همین طور که برنامه شما بزرگ تر میشه و جلو میره ، مسخره میشن . شما باید اون ها رو توی یک لِول دیگه مرتب کنید . این ها ماژول ها هستند . پس میتونیم اونها رو در ابزار مرتب سازی برای کد هامون خلاصه کنیم .

ساخت ماژول ها آسان هست . اون ها فایل پایتون هستند ، مثل اسکریپت های معمولی شما . برای ساخت یک ماژول ، یک یا چند تا تابع در یک فایل متنی بنویسید ، بعد با پسوند . py ذخیرش کنید . بیایید این کار رو با یک مثال انجام دهیم . یک فایل جدید در ویرایشگر متنی یا IDE خود باز کنید ، و یک تابع بسازید . من میرم که یک مثال قبلیمون رو ادامه بدم و یک تابع محاسبه مالیات بر ارزش افزوده محصولات بنویسم .

یک فایل جدید بسازید و تابع محاسبه مالیات رو بنویسید .

```
def addTax(price, tax):
    newPrice = price / 100 * (100 + tax)
    return newPrice
```

اگر با پسوند .py در همون دایرکتوری قبلی اسکریپت دیگرمون ذخیرش کنیم میتونیم ازش به عنوان یک ماژول استفاده کنیم ! این مهمه که یک نام خوب برایش انتخاب کنیم تا بفهمیم که چکر میکنه وقتی که در موابق بعد بهش برخوردیم ، پس اسم این یکی رو اینطور بگذارید

finance.py

وارد کردن ماژول ها

برای استفاده از ماژول ها ، ما میتونیم از یکی از کلمات کلیدی زیر استفاده کنیم :

```
import
from
```

ایمپرت(import) آسون ترین و معمول تر هستش ، پس بیایید اول اون رو امتحان کنیم . بعد نام ماژول رو مشخص میکنید ، که همون نام فایل هستش ، بدون پسوند . برای مثال با ماژول (finance) ما در پوشه ای که اسکریپتمون قرار داره میتونیم بنویسیم :

```
import finance
```

--- و میتونیم به تمام توابع داخل اسکریپت دسترسی داشته باشیم . ما بعدا میتونیم همون طوری که قبلا انجام میدادیم صداشون بزنیم . شما همچنین از کلمه کلیدی (from) هم میتونید استفاده کنید ، که به شما اجازه انتخاب توابع خاصی رو از کتاب خونه میده ، اگر که بدونید از قبل کدوم ها لازمه . برای یک ماژولی که صدها تابع داره ، این پیشنهاد میشه . این کار باعث زود تر بارگذاری شدن میشه چون توابعی رو که لازم نیست رو بارگذاری نمیکنه :

```
from finance import addTax
```

شما میتونید چند تا تابع رو وارد کنید با گذاشتن کاما یا ویرگول بین آن ها :

```
from finance import addTax, calculateDiscount
```

شما حتی میتونید از ستاره استفاده کنید تا توابع رو به صورت تصادفی وارد کنید :

```
from finance import *
```

بعد از وارد کردن ماژول ، برای استفاده از توابع همراهش ، شما از نام ماژول استفاده میکنید ، بعدش یک نقطه ، و بعد نام تابع :

```
import finance
print finance.addTax(100, 5)
```

این باید نتیجش 105 بشه وقتی که اسکریپت اجرا بشه . . یک نتیجه ساده ، اما این یعنی اینکه شما یک تابع سالم دارید .

ماژول های داخلی (Built-in Modules)

ماژول های داخلی زیاد هستند ، مثل توابع داخلی . اینجا نقطه قوت پایتون هستش ! این چیزیه که بهش میگن **دارای باتری** (batteries included).

چون پایتون ماژول های خیلی زیادی داره ، هیچ راهی نیست که همشون رو در یک درس بررسی کنیم . خیلی هاشون کم استفاده میشن ، و هیچ فایده ای نداره چیز های غیر ضروری رو درس داد . به جای اینکار ، بیایید چند تا از کاربردی ترین آنها رو بررسی کنیم :

random
math
os
datetime
urllib2

random

یکی از بهترین ها برای شروع تصادفی یا همون رندم هستش ، چون یادگیریش آسونه و در بیشتر اسکریپت ها کاربردیه . همون طوری که انتظار داشتید ، این ماژول به شما اجازه تولید اعداد تصادفی میده . برای مثال ، وقتی از پایتون برای ساخت یک سایت استفاده میکنید ، این میتونه در دیتابیس پسورد های شما برای امن تر شدن اون باشه . اگر ما یک عدد صحیح تصادفی خواستیم ، میتونیم از تابع (randint) استفاده کنیم . مثل مثال زیر :

```
import random
print random.randint(0, 5)
```

این خروجیش میشه 1و 2و 3و 4 یا 5 . randint دو تا پارامتر رو قبول میکنه : کمترین و بیشترین عدد . همان طور که میبینید ، این شامل بیشترین عدد میشه اما یکی از کمریتن عدد بیشتر خروجی میده . اگر عدد اعشاری تصادفی میخواین ، میتونید از تابع رندم استفاده کنید :

```
import random
random.random()
```

... که خروجیش عدد تصادفی بین 0 و 1 میشه ، با اعداد اعشارش . اگر عدد بزرگتری میخواهید ، میتونید ضربش کنید . برای مثال ، یک عدد تصادفی بین 0 تا 100 :

```
import random
random.random() * 100
```

ماژول رندم همچنین یک تابع برای انتخاب چیزی از یک ست خاص مثل لیست رو داره که choice

نام داره .

```
Import random
myList = [1, 8, True, 77, “Loren”, 482, “Ipsum”]
random.choice(myList)
```

math

ماژول math دسترسی به کار های ریاضی و توابعش رو فراهم میکنه .

```
import math

math.pi #Pi, 3.14...
math.e  #Euler's number, 2.71...

math.degrees(2)  #2 radians = 114.59 degreees
math.radians(60) #60 degrees = 1.04 radians

math.sin(2)    #Sine of 2 radians
math.cos(0.5)  #Cosine of 0.5 radians
math.tan(0.23) #Tangent of 0.23 radians

math.factorial(5) #1 * 2 * 3 * 4 * 5 = 120
math.sqrt(49)   #Square root of 49 = 7
```

توابع خیلی بیشتری در این ماژول هستش ، اما این ها از کاربردیاش بودن . میتونید لیستی از اونها رو در اینجا ببنیدید :

http://docs.python.org/library/math.html

datetime

اگر میخواهید با تاریخ و زمان کار کنید ، ماژول datetime دوست شماست . این برای طراحی وب خیلی کاربردی هستش . اگر یک سایت با قسمت نظرات میسازید ، یا یک وبلاگ ، بعد شما شاید سنشون ، یا زمانی که ساخته شده اند رو نمایش بدهید .

Import datetime

برای راحتی ، میتونید اجزای این ماژول رو به صورت جدا وارد کنید ، که دیگه لازم نباشه بعدا زیاد بنویسید .

Import datetime
from datetime import date
import time

بیایید نگاهی به توابعی که برامون فراهم میکنه بیاندازیم :

```
time.time() #Returns the number of seconds since the Unix Epoch, January 1st 1970

date.fromtimestamp(123456789)   #Converts a number of seconds to a date object
date.fromtimestamp(time.time()) #We can combine the two to get an object representing the current time

date.fromordinal(10000) #Returns the date a given number of days since January 1st in the Year 1 - 28th of May 0018 in this case
```

چند تا تابع کاربردی برای تبدیل تاریخ به رشته های قابل استفاده هست . **Strftime** به شما اجازه میده که یک رشته رو به همراه نشانه درصد استفاده کنید .

%d

روز رو نمایش میده ، برای مثال :

```
currentDate = date.fromtimestamp(time.time()) #Create a variable representing the time now
currentDate.strftime("%d/%m/%y") #Format the date like DD/MM/YY - in this case 09/07/11

currentDate.isoformat() #Format as an ISO standard date - in this case 2011-07-09
```

os

ماژول دیگری که بررسی میکنیم os هستش ، که به شما اجازه میده با سیستم عاملی که پایتون در اون در حال اجراست کار کنید – مثل ویندوز ، مک یا لینوکس . ما روی زیرماژول path تمرکز میکنیم . به شما اجازه یافتن مشخصه های فایل ها و پوشه ها رو روی سیستم میده ، پس برای طراحی وب پر کاربرد تزین هستش . بعضی اوغات باید فایل ها رو مدیریت کنید ، برای مثال آپلود های کاربر .

```
from os import path

path.exists("/Users/Giles") #Checks if the directory exists, in my case it's True
path.exists("/Users/Bob")   #False this time

from os import path
path.getatime("/Users") #Get the last time the given directory was accessed as a timestamp
path.getmtime("/Users") #Get the last time the given directory was modified as a timestamp
```

این زمان ها میتونه به رشته های قابل استفاده تبدیل بشه توسط ماژول datetime – میتونید ببینید که چطور میشه ماژول ها رو با هم قاطی کرد :

```
from os import path
path.getsize("/Users/Giles/Desktop/boot") #Returns the size of a file in bytes - for this file it was 314400 bytes, or 314kb
```

آخرین تابعی که در ماژول os میخوایم بررسی کنیم **join** هستش . این تابع دو تا path یا همون مسیر رو با هم ادغام میکنه . شما شاید فکر میکنید چرا من نمیتونم رشته ها رو با هم ادغام کنم ؟ اما به همین راحتی هم نیست . در ویندوز ، مسیر ها رو با \ مشخص میکنن ، در مک و لینوکس با / . این ماژول باعث رفع چنین مشکلاتی برای اسکریپت پایتونتون میشه .

```
path.join("C:", "Users") #Returns "C:/Users"
```

urllib2

برای اتمام تور کتاب خانه های استاندارد پایتون ، ما یک نگاه کوتاهی به urllib2 می اندازیم . این ماژول به شما اجازه کار با وب رو میده ، پس کاملا مشخصه برای ما . کاربردی ترین تابع این ماژول urlopen هستش که یک صفحه رو دانلود میکنه . شما مثل زیر ازش

استفاده میکنید :

```
import urllib2
urllib2.urlopen("http://net.tutsplus.com")
```

شما میتونید به جای سایتی که در رشته نوشتم هر سایت دیگری بگذارید . این تابع محتوای اچ تی ام ال صفحه رو دانلود میکنه . اگر چه یک رشته رو بر نمیگردونه ، پس ما باید اطلاعات رو بخونیم تا بتونیم خارجش کنیم :

```
import urllib2
urllib2.urlopen("http://net.tutsplus.com").read(100)
```

این خط بالایی 100 کاراکتر اول فایل اچ تی ام ال رو برمیگردونه . وقتی که این اطلاعات رو داشتید ، میتونید توش بگردید و اطلاعاتی رو که لازم دارید ازش استخراج کنید .

نکات آخر

اگر ماژول شما در یک دارکتوری دیگری از اسکریپت شما قرار داره ، وارد کردنشون کمی سخت تره . شما باید یک فایل درست کنید که به پایتون بگه که پوشه یک پکیج هست . برای مثال ، اگر یک پوشه دارید ، به نام subdirectory ، در همون پوشه اسکریپتتون ، و بعد ماژول شما داخل پوشه subdirectory هستش ، شما باید یک فایل به نام

```
__init__.py
```

در همون پوشه ای که ماژول قرار داره بسازید . این فایل میتونه خالی باشه . پس در اسکریپتتون اینطور واردش میکنید :

```
%review it if you need a refresher!
```

مترجم : علی مرادی

ایمیل : adeadmarshal@gmail.com