توابع از پیش تعریف شده (Built-in Function)

در درس های قبل در مورد چگونگی تعریف تابع و ارسال آرگومان به آن و ... بحث کردیم. پایتون علاوه بر توابعی که توسط کاربر تعریف می شوند دارای توابع دیگری نیز هست که به آنها توابع از پیش تعریف شده می گویند. طراحان زبان برنامه نویسی پایتون برای سادگی کار برنامه نویسان، این توابع را نوشته و به همراه پایتون ارائه می دهند. تعداد این توابع به همراه نسخه های جدید پایتون افزایش می یابد. این توابع دارای کابردهای مختلفی از جمله برگرداندن قدر مطلق یک عدد، تبدیل انواع داده به هم، ایجاد لیست و ... به کار می روند. در جدول زیر لیست توابع از پیش تعریف شده در پایتون آمده است :

توابع از پیش تعریف شده				
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	()

نام برخی از این توابع در جدول بالا برای شما آشناست. مثل تابع print() که باعث چاپ مقادیر در خروجی می شود. در مثال زیر هم در عمل نحوه کار با برخی دیگر از این توابع نشان داده شده است :

```
print(abs(-3));
print(bin(5));
print(chr(97));
print(float(10));
print(pow(2, 2));
```

```
3
0b101
a
10.0
4
```

همانطور که احتمالا از خروجی کدهای بالا متوجه شده اید، مثلا تابع pow() دو آرگومان می گیرد که اولی عدد و دومی توان می باشد. در مثلا بالا عدد 2 را به توان دو رسانده ایم، که در خروجی عدد 4 نشان دهده شده است. یا مثلا تابع abs() قدر مطلق 3- را که عدد 3 می باشد را برگشت داده است. تابع از پیش تعریف شده range() یک محدوده از اعداد را ایجاد می کند. به کد زیر توجه کنید :

```
for number in (1, 2, 3, 4, 5):
    print(number);
```

در کد بالا مقادیر یک مجموعه از اعداد را چاپ کرده ایم. برای ایجاد همین محدوده از اعداد با استفاده از تابع range() می توان به صورت زیر عمل کرد:

```
for number in range(1,6):
    print(number);
```

خروجی دو کد بالا، شبیه هم می باشد. ممکن است که این سوال برایتان پیش بیاید که چرا در کد بالا 6 را در داخل تابع نوشته ایم. آرگومان دوم تابع range() جز خروجی نست. یعنی اعداد 1 تا 5 چاپ می شوند. اگر به جای 6 عدد 10 را بنویسید. اعداد 1 تا 9 چاپ می شوند. از تابع type() هم برای تشخیص نوع داده استفاده می شود :

```
intVar        = 10;
floatVar      = 12.5;
boolVar       = True;
StringVar     = "Hello World!";
listVar       = [1,5,8];
tupleVar      = ("Python","Programming","begginer");
dictionaryVar = {'Name': 'jack', 'family': 'Scalia', 'Age': 7}

print(type(intVar));
print(type(floatVar));
print(type(boolVar));
print(type(StringVar));
print(type(listVar));
print(type(tupleVar));
print(type(dictionaryVar));
```

```
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'str'>
<class 'list'>
<class 'tuple'>
<class 'dict'>
```

توابع help() و dir()

در میان توابع بالا، دو تابع help() و dir() از اهمیت ویژه ای برخوردار هستند. این دو تابع اطلاعات مفیدی در مورد انواع مختلف داده، توابع، کلاس ها، ماژول ها و در مجموع هر چیزی که در پایتون با آن سرو کار داریم در اختیار ما قرار می دهند. برای به دست آوردن اطلاعات، کافیست نام آن متغیر، کلاس و یا ... را در داخل پرانتز این توابع بنویسیم. به مثال زیر توجه کنید:

```
str = 'Python programming!'

print(dir(str))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__',
 '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
 '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find',
 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier',
 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

تابع dir() لیست توابعی که با استفاده از آنها می توان عملیاتی را بر روی رشته مورد نظر (str) انجام داد، در اختیار ما می گذارد. حال ممکن است که شما ندانید که مثلا تابع upper() در لیست بالا چه کاری انجام می دهد. برای این منظور کافیست که از تابع help() استفاده نمایید:

```
str = 'Python programming!'

help(str.upper)
```

همانطور که مشاهده می کنید، کافیست که در داخل پرانتز تابع help() نام رشته، سپس علامت نقطه و در نهایت نام تابع را بدون پرانتز بنویسید، تا اطلاعات کاملی در مورد وظیفه آن تابع در اختیار شما قرار داده شود. خروجی برنامه بالا به صورت زیر است:

```
Help on built-in function upper:

upper(...) method of builtins.str instance
    S.upper() -> str

    Return a copy of S converted to uppercase.
```

در توضیح این تابع آمده است که تابع upper() یک تابع داخلی می باشد. علامت (...) بدین معناست که این تابع، پارامتر نمی گیرد و علامت -> نیز نمایانگر نوع خروجی این تابع است که در این مورد str یا همان رشته است. در خط آخر هم آمده است که این تابع یک رشته را گرفته و آن را به حروف بزرگ تبدیل می کند:

```
str = 'Python programming!'

print(str.upper())
```

```
PYTHON PROGRAMMING!
```