Nə üçün Python ?

- Python inanılmaz dərəcədə səmərəli dildir: proqramlarınız digər dillərin tələb etdiyi kod sətrinə nisbətən daha az sətirlə daha çox iş yerinə yetirəcək.
- Python-un sintaksisi sizə “təmiz” kod yazmağa kömək edəcək.
- Kodunuz digər dillərlə müqayisədə daha oxunaqlı, daha asan başa düşülən (debug edilən) və genişləndirilməsi, inkişaf etdirilməsi daha rahat olacaq.

hello_world.py faylını işlətdikdə əslində nə baş verir ?
Məlum olur ki, Python hətta sadə bir proqramı işlətdikdə belə kifayət qədər iş görür:

```
hello_world.py
print("Hello Python world!")
```

Bu kodu işlətdikdə, aşağıdakı nəticəni görməlisiniz:
Hello Python world!

hello_world.py faylını işlətdiyiniz zaman, sondakı .py bu faylın Python proqramı olduğunu göstərir. Daha sonra redaktor (editor) faylı Python tərcüməçisindən (interpreter) keçirir; interpreter proqramı oxuyur və hər bir sözün nə demək olduğunu müəyyənləşdirir. Məsələn, interpreter print sözündən sonra mötərizə gördükdə, mötərizənin içində olan hər şeyi ekrana çap edir.

Dəyişənlər (Variables):
Gəlin hello_world.py faylında dəyişəndən istifadə etməyə cəhd edək. Faylın əvvəlinə yeni sətir əlavə edin və ikinci sətri dəyişdirin:

```
hello_world.py
message = "Hello Python world!"
print(message)
```

Əvvəl gördüyünüz nəticənin eyni olduğunu görməlisiniz:
Hello Python world!

Biz message adlı bir dəyişən əlavə etdik. Hər dəyişən ona bağlı olan bir dəyərə malikdir və bu dəyər həmin dəyişənlə əlaqəli məlumatdır. Bu halda dəyər "Hello Python world!" mətnidir. Dəyişən əlavə etmək Python tərcüməçisi üçün bir az daha çox iş yaradır. Python birinci sətri işləyəndə, message dəyişənini "Hello Python world!" mətni ilə əlaqələndirir. İkinci sətrə çatdıqda isə message dəyişəninə bağlı olan dəyəri ekrana çap edir.
Gəlin bu proqramı genişləndirək və hello_world.py faylını ikinci mesajı çap edəcək şəkildə dəyişdirək.

```
message = "Hello Python world!"
print(message)

message = "Hello Python Crash Course world!"
print(message)
```

İndi hello_world.py faylını işlətdikdə, iki sətirlik nəticə görməlisiniz:
Hello Python world!
Hello Python Crash Course world!

Siz proqramınızda istənilən vaxt dəyişənin dəyərini dəyişə bilərsiniz və Python həmişə həmin dəyişənin ən son (aktual) dəyərini yadda saxlayacaq.

Dəyişənlərin adlandırılması və istifadəsi:
Python-da dəyişənlərdən istifadə edərkən bir neçə qaydaya və tövsiyəyə əməl etməlisiniz. Bu qaydaların bəzilərinin pozulması xəta yaradar; digər tövsiyələr isə sadəcə kodunuzu daha oxunaqlı və başa düşülən etməyə kömək edir. Aşağıdakı dəyişən qaydalarını mütləq yadda saxlayın:

- **Dəyişən adları yalnız hərflərdən, rəqəmlərdən və alt xətdən (_) ibarət ola bilər.**

Onlar hərf və ya alt xətt ilə başlaya bilər, amma rəqəmlə başlaya bilməz.

- **Dəyişən adlarında boşluq istifadə etmək olmaz, lakin sözləri ayırmaq üçün alt xətdən istifadə etmək olar.**

Məsələn, greeting_message düzgündür, amma greeting message xəta verəcək.

- **Python-un açar sözlərini və funksiyalarının adlarını dəyişən adı kimi istifadə etməkdən çəkinin;**

yəni Python-un xüsusi məqsədlər üçün ayırdığı sözləri, məsələn print sözünü dəyişən adı kimi istifadə etməyin.

- **Dəyişən adları qısa, amma izahlı olmalıdır.**

Məsələn, name n-dən daha yaxşıdır; student_name s_n-dən daha yaxşıdır; name_length isə length_of_persons_name-dən daha yaxşıdır.

- **Kiçik “l” və böyük “O” hərflərindən istifadə edərkən diqqətli olun, çünki onlar rəqəm 1 və 0 ilə qarışdırıla bilər.**

Yaxşı dəyişən adları yaratmağı öyrənmək üçün bir qədər məşq lazımdır, xüsusilə proqramlarınız daha maraqlı və mürəkkəb olduqca. Daha çox proqram yazdıqca və başqalarının kodlarını oxumağa başladıqca, mənalı dəyişən adları tapmaqda daha da yaxşılaşacaqsınız.

Dəyişənlərdən istifadə edərkən ad xətalarından qaçmaq:

Hər proqramçı səhv edir və əksəriyyəti hər gün səhvlər edir. Yaxşı proqramçılar səhv edə bilsələr də, bu səhvlərə səmərəli şəkildə necə cavab verəcəyini də bilirlər. Gəlin erkən mərhələdə tez-tez rastlaşacağınız bir xətaya baxaq və onu necə düzəltməyi öyrənək. Gəlin bilərəkdən xəta yaradan bir kod yazaq.

```
message = "Hello Python Crash Course reader!"
print(mesage)
```

Proqramınızda xəta baş verdikdə, Python tərcüməçisi problemin harada olduğunu müəyyənləşdirməyə kömək etməyə çalışır. Proqram

uğurla işləmirəmsə, interpreter bir **traceback** təqdim edir. Traceback — interpreterin kodu işlədərkən qarşılaşdığı problemlərin qeydidir. Bir dəyişənin adını səhv yazdıqdan sonra Python-un təqdim etdiyi traceback nümunəsi belədir:

```
Traceback (most recent call last):
  File "hello_world.py", line 2, in <module>
    print(mesage)
NameError: name 'mesage' is not defined
```

Traceback göstərir ki, xəta hello_world.py faylının 2-ci sətrində baş verib. Interpreter bu sətri göstərir ki, xətanı tez tapa bilək və hansı növ xəta olduğunu bildirir. Bu halda **NameError** baş verib və çap ediləcək mesage dəyişənin müəyyən edilmədiyini bildirir. Python verilmiş dəyişən adını tanıya bilmir. NameError adətən o deməkdir ki, ya dəyişənə dəyər verməyi unutmuşuq, ya da dəyişənin adını yazarkən səhv etmişik. Əlbəttə, bu nümunədə ikinci sətrdə message dəyişənindəki **s** hərfini unutmuşuq. Python tərcüməçisi kodunuzu imla yoxlamır, amma dəyişən adlarının ardıcıl yazılmasını təmin edir.

Çox proqramlaşdırma səhvləri proqramın bir sətrindəki sadə, tək simvol səhvləridir. Əgər bu cür səhvi tapmaq üçün uzun vaxt sərf edirsinizsə, bilin ki, siz tək deyilsiniz. Təcrübəli və istedadlı proqramçılar da bu cür kiçik səhvləri tapmaq üçün saatlarla vaxt sərf edirlər. Buna gülərək keçməyə çalışın, çünki proqramlaşdırma həyatınız boyunca bu tez-tez baş verəcək.

Variables Are Labels

Dəyişənlər tez-tez dəyərləri saxlaya biləcəyiniz qutular kimi təsvir olunur. Bu fikir dəyişəni ilk bir neçə dəfə istifadə edəndə faydalı ola bilər, amma Python-da dəyişənlərin daxili şəkildə necə təqdim olunduğunu düzgün təsvir etmir. Dəyərlərə mənimsədə biləcəyiniz etiketlər kimi dəyişənlər haqqında düşünmək daha düzgündür. Bunu

da deyə bilərsiniz ki, dəyişən müəyyən bir dəyərə istinad edir. Bu fərq yəqin ki, ilk proqramlarınızda o qədər də əhəmiyyətli olmayacaq, amma bunu gec yox, erkən öyrənmək daha faydalıdır. Elə bir vaxt gələcək ki, dəyişəndən gözlənilməz davranış görəcəksiniz, və dəyişənlərin necə işlədiyini düzgün başa düşmək kodunuzda nə baş verdiyini müəyyənləşdirməyə kömək edəcək.

Data Types

- **None**
- **Strings (str)**
- **Bytes**
- **Float**
- **Boolean (bool)**
- **Integer (int)**

Strings

Data tiplərindən biridir.

Baxacağımız ilk data tipi stringdir. Stringlər ilk baxışda olduqca sadədir, lakin onları çox müxtəlif formalarda istifadə edə bilərsiniz. String simvollar ardıcıllığıdır. Python-da dırnaq içində olan hər şey string hesab olunur və stringlərinizi bu şəkildə tək və ya cüt dırnaqlarla yaza bilərsiniz:

```
"This is a string."
'This is also a string.'
```

Bu çeviklik stringlərinizin içində dırnaqlardan və apostroflardan istifadə etməyə imkan verir:

```
'I told my friend, "Python is my favorite language!"'
"The language 'Python' is named after Monty Python, not the snake."
"One of Python's strengths is its diverse and supportive community."
```

Changing Case in a String with Methods:
Stringlərlə edə biləcəyiniz ən sadə tapşırıqlardan biri sətirdəki sözlərin böyük-kiçik hərflərini dəyişdirməkdir.

Aşağıdakı koda baxın və nə baş verdiyini anlamağa çalışın:

```
name = "ada lovelace"
print(name.title())
```

Aşağıdakı nəticəni görməlisiniz:
(Ada Lovelace)
Bu nümunədə name adlı dəyişən "ada lovelace" adlı kiçik hərflərlə yazılmış sətirə istinad edir. title() metodu print() funksiyası içində dəyişəndən sonra yazılıb. **Metod** — Python-un verilən üzərində yerinə yetirdiyi bir əməliyyatdır. name.title() ifadəsindəki nöqtə (.) Python-a title() metodunun name dəyişəni üzərində işləməli olduğunu bildirir. Hər metoddan sonra mötərizələr gəlir, çünki metodların çox vaxt işlərini görmək üçün əlavə məlumata ehtiyacı olur. Bu məlumat mötərizələrin içində verilir. title() metodunun əlavə məlumata ehtiyacı yoxdur, buna görə də mötərizələri boşdur.
title() metodu hər bir sözü baş hərfi böyük olacaq şəkildə dəyişdirir. Bu faydalıdır, çünki çox vaxt adı ayrıca bir məlumat vahidi kimi qəbul etmək istəyəcəksiniz. Məsələn, proqramınızın Ada, ADA və ada kimi daxil edilən adları eyni ad kimi tanımasını və hamısını Ada kimi göstərməsini istəyə bilərsiniz. Hərflərin böyük-kiçik formaları ilə işləmək üçün bir neçə digər faydalı metod da mövcuddur. Məsələn, sətiri tam böyük hərflərə və ya tam kiçik hərflərə belə çevirə bilərsiniz:

```
name = "Ada Lovelace"
print(name.upper())
print(name.lower())
```

Bu aşağıdakı nəticəni göstərəcək:

```
ADA LOVELACE
 ada lovelace
```

lower() metodu xüsusilə məlumatları saxlamaq üçün faydalıdır. Çox vaxt istifadəçilərin daxil etdiyi böyük-kiçik hərf yazılışına etibar etmək istəməzsiniz, buna görə də məlumatı saxlamadan əvvəl sətirləri kiçik hərflərə çevirirsiniz. Sonra, məlumatı göstərmək istədikdə, hər sətir üçün ən uyğun olan böyük-kiçik hərf formatından istifadə edəcəksiniz.

Using Variables in Strings

Bəzi hallarda dəyişənin dəyərini bir stringin içində istifadə etmək istəyə bilərsiniz. Məsələn, iki dəyişənin biri adı, digəri soyadı təmsil etməsini, sonra isə bu dəyərləri birləşdirərək bir şəxsin tam adını göstərməsini istəyirsinizsə:

```
first_name = "ada"

last_name = "lovelace"

full_name = f"{first_name} {last_name}"

 print(full_name)
```

Dəyişənin dəyərini stringin içində istifadə etmək üçün açıq dırnaq işarəsinin əvvəlinə dərhal f hərfini yazın. Sətirdə istifadə etmək istədiyiniz dəyişənin adını mötərizələrə {} alın. Sətir göstərilərkən Python həmin dəyişənlərin adlarını onların dəyərləri ilə əvəz edəcək. Bu cür stringlər **f-strings** adlanır.
Buradakı f “format” sözünün baş hərfidir, çünki Python mötərizə içindəki dəyişən adlarını onların dəyərləri ilə əvəz edərək stringi formatlayır.
f-stringlərlə çox şey etmək mümkündür. Məsələn, dəyişənlə bağlı məlumatdan istifadə edərək tam mesaj yaratmaq üçün f-stringlərdən istifadə edə bilərsiniz, aşağıdakı kimi:

```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
```

```
print(f"Hello, {full_name.title()}!")
```

Tam ad istifadəçini salamlayan bir cümlədə işlədilir və title() metodu adı baş hərfləri böyük olmaqla formatlayır. Bu kod sadə, amma səliqəli şəkildə formatlanmış bir salamlamanı qaytarır:

Hello, Ada Lovelace!

Həmçinin, komplekt bir mesaj yaratmaq üçün f-stringlərdən istifadə edib həmin mesajı bir dəyişənə mənimsədə bilərsiniz:

```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
message = f"Hello, {full_name.title()}!"
print(message)
```

Bu kod da “Hello, Ada Lovelace!” mesajını göstərir, lakin mesajı ayrıca bir dəyişənə mənimsədiyimiz üçün son print() çağırışı daha sadə olur.

Note

format() istifadə etmək üçün, sətirdə istifadə ediləcək dəyişənləri format sözündən sonrakı mötərizələrin içində ardıcıllıqla yazın. Hər dəyişən {} şəklində mötərizə ilə göstərilir; bu mötərizələr mötərizələrin içində yazdığınız dəyişən dəyərləri ilə ardıcıllıqla doldurulur:

full_name = "{} {}".format(first_name, last_name)

Adding Whitespace to Strings with Tabs or Newlines:

Proqramlaşdırmada **boşluq (whitespace)** çap olunmayan simvollara — məsələn, boşluq işarəsi, tab və sətirsonu simvollarına — deyilir. Outputu daha oxunaqlı etmək üçün boşluqlardan istifadə edə bilərsiniz. Mətninizə tab əlavə etmək üçün \t simvol kombinasiyasından istifadə edin. Stringdə yeni sətir yaratmaq üçün \n simvol kombinasiyasından istifadə edin. Tab və yeni sətiri eyni sətirdə birləşdirə bilərsiniz. "\n\t" sətiri Pythona bir sətir aşağı keçməyi və yeni

sətirə tab ilə başlamasını bildirir. Aşağıdakı nümunə bir sətirdən ibarət mətnlə dörd sətirlik çıxış yaratmağın mümkün olduğunu göstərir:

```
>>> print("Python")
Python
>>> print("\tPython")
Python
>>> print("Languages:\nPython\nC\nJavaScript")
Languages:
Python
C
JavaScript


>>> print("Languages:\n\tPython\n\tC\n\tJavaScript")
Languages:
	Python
	C
	JavaScript
```

Stripping Whitespace (boşluqları təmizləmək):

Artıq boşluqlar proqramlarınızda qarışıqlığa səbəb ola bilər. Proqramçılar üçün 'python' və 'python ' demək olar ki, eyni görünür. Amma proqram üçün bunlar iki fərqli sətirdir. Python 'python ' sətirindəki əlavə boşluğu aşkar edir və siz ona başqa cür demədikcə bunu əhəmiyyətli hesab edir. Boşluğu nəzərə almaq vacibdir, çünki tez-tez iki sətirin eyni olub-olmadığını müqayisə etmək lazım olacaq. Məsələn, bunun vacib olduğu hallardan biri istifadəçilər vebsayta daxil olanda onların istifadəçi adlarını yoxlamaqdır. Əlavə boşluqlar daha sadə vəziyyətlərdə də qarışıqlığa səbəb ola bilər. Xoşbəxtlikdən, Python insanların daxil etdiyi məlumatlardan artıq boşluğu asanlıqla silməyə imkan verir.

Python bir sətirin sağ və sol tərəfindəki əlavə boşluqları yoxlaya bilər. Sətirin sağ tərəfində boşluq qalmadığından əmin olmaq üçün **rstrip()** metodundan istifadə edin.

```
>>> favorite_language = 'python '
>>> favorite_language
'python '
>>> favorite_language.rstrip()
'python'
>>> favorite_language
'python '
```

1-ci sətrində favorite_language dəyişəninə aid olan dəyərin sonunda artıq boşluq var. Terminalda bu dəyəri göstərməyi istədikdə dəyərin sonunda boşluğu görə bilərsiniz. favorite_language üzərində rstrip() metodu işləndikdə bu artıq boşluq silinir. Lakin, bu, yalnız müvəqqətidir. favorite_language dəyərini yenidən göstərməsini istəsəniz, daxil edilən halı ilə — yəni əlavə boşluqla birlikdə — eyni göründüyünü görəcəksiniz. Sətirdəki boşluğu qalıcı şəkildə silmək üçün təmizlənmiş dəyəri dəyişən adına mənimsətməlisiniz:

```
>>> favorite_language = 'python '
>>> favorite_language = favorite_language.rstrip()
>>> favorite_language
'python'
```

Sətirdəki boşluğu silmək üçün sətirin sağ tərəfindəki boşluğu təmizləyirsiniz və sonra bu yeni dəyəri orijinal dəyişənə mənimsədirsiniz. Proqramlaşdırmada dəyişənin dəyərini dəyişmək tez-tez edilir. Bir proqram işlədikcə və ya istifadəçi daxilinə cavab olaraq dəyişənin dəyəri bu cür yenilənir.

Sol tərəfdəki boşluğu **lstrip()** metodu ilə, hər iki tərəfdəki boşluğu isə **strip()** metodu ilə təmizləyə bilərsiniz:

```
>>> favorite_language = ' python '
>>> favorite_language.rstrip()
```

```
' python'
>>> favorite_language.lstrip()
'python '
>>> favorite_language.strip()
'python'
```

Bu nümunədə həm əvvəlində, həm də sonunda boşluq olan bir dəyərlə başlayırıq. Daha sonra boşluğu sağdan, soldan və hər iki tərəfdən təmizləyirik. Bu təmizləmə funksiyaları ilə təcrübə aparmaq sətirlərlə işləməyə daha yaxşı alışmağınıza kömək edir. Real həyatda bu təmizləmə funksiyaları ən çox istifadəçi daxilini proqramda saxlamadan əvvəl təmizləmək üçün istifadə olunur.

Avoiding Syntax Errors with Strings:

Vaxtaşırı görə biləcəyiniz səhv növlərindən biri sintaksis səhvidir. Sintaksis səhvi Python-un proqramınızın bir hissəsini düzgün Python kodu kimi tanımadığı zaman baş verir. Məsələn, tək dırnaq içində apostrof işlətsəniz, səhv yaranacaq. Bu ona görə baş verir ki, Python birinci tək dırnaqla apostrof arasındakı hər şeyi sətir kimi qəbul edir. Daha sonra qalan mətni Python kodu kimi oxumağa çalışır və bu da səhvlərə səbəb olur. Tək və ikiqat dırnaqlardan düzgün istifadə etməyin yolu belədir. Bu proqramı **apostrophe.py** adı ilə yadda saxlayın və sonra işə salın:

```
message = "One of Python's strengths is its diverse community."
print(message)
```

Apostrof ikiqat dırnaqların içində olduğuna görə Python interpretatoru sətiri düzgün oxumaqda çətinlik çəkmir:

```
One of Python's strengths is its diverse community.
```

Lakin tək dırnaq işlətsəniz, Python sətirin harada bitdiyini müəyyən edə bilmir:

```
message = 'One of Python's strengths is its diverse community.'
print(message)
```

```
  File "apostrophe.py", line 1
    message = 'One of Python's strengths is its diverse community.'
                            ^
SyntaxError: invalid syntax
```

Çıxışda görürsünüz ki, səhv işarəsi ikinci tək dırnaqdan dərhal sonra meydana gəlir. Bu sintaksis səhvi interpretatorun kodun bir hissəsini düzgün Python kodu kimi tanımadığını göstərir.

Numbers

Proqramlaşdırmada rəqəmlər çox tez-tez istifadə olunur. Məsələn:

- oyunlarda xal saxlamaq üçün
- qrafik və vizuallaşdırmalarda məlumat göstərmək üçün
- veb tətbiqlərində məlumatları yadda saxlamaq üçün
- hesablamalar aparmaq üçün

Python rəqəmlərlə işləyəndə onları fərqli şəkildə qəbul edir. Bu, onların hansı məqsədlə istifadə olunmasından asılıdır.

İlk əvvəl **tam ədədlər (integers)** ilə başlayırıq, çünki işləməsi ən sadə olan rəqəm növüdür.

Python tam ədədləri adi riyazi ədədlər kimi qəbul edir:

`-3, 0, 7, 15, 100000` — bunlar hamısı tam ədədlərdir.

Integers

*Python-da tam ədədlərlə toplama (+), çıxma (-), vurma (*) və bölmə (/) əməliyyatları edə bilərsiniz:

```
>>> 2 + 3
```

```
5
>>> 3 - 2
1
>>> 2 * 3
6
>>> 3 / 2
1.5
```

Terminalda Python sadəcə əməliyyatın nəticəsini qaytarır. Python qüvvətləri (üstlü ədədləri) göstərmək üçün iki vurma işarəsi ** istifadə edir:

```
 >>> 3 ** 2
9
>>> 3 ** 3
27
>>> 10 ** 6
1000000
```

Python əməliyyatların prioritet qaydasını dəstəkləyir, buna görə bir ifadədə birdən çox əməliyyat edə bilərsiniz. Həmçinin mötərizələrdən istifadə edərək əməliyyatların ardıcıllığını dəyişə bilərsiniz; Python ifadəni sizin göstərdiyiniz ardıcıllıqla qiymətləndirəcək.

```
>>> 2 + 3*4
14
>>> (2 + 3) * 4
20
```

Floats

Python onluq nöqtəsi olan hər hansı ədədi float adlandırır. Bu termin əksər proqramlaşdırma dillərində istifadə olunur və onluq nöqtəsinin ədədin istənilən yerində yerləşə biləcəyini göstərir. Hər bir proqramlaşdırma dili onluq ədədləri düzgün idarə edə biləcək şəkildə dizayn edilməlidir ki, onluq nöqtəsi harada yerləşsə də ədədlər düzgün

işləsin. Əksər hallarda, onluq ədədlərlə işləyərkən onların necə davrandığını düşünməyə ehtiyac yoxdur. Sadəcə istifadə etmək istədiyiniz ədədləri daxil edin və Python çox güman ki, gözlədiyiniz nəticəni verəcək:

```
>>> 0.1 + 0.1
0.2
>>> 0.2 + 0.2
0.4
>>> 2 * 0.1
0.2
>>> 2 * 0.2
0.4
```

Integers and Floats

Hər hansı iki ədədi bölləndə, hətta nəticə tam ədəd olsa belə, nəticə həmişə **float** (ondalıklı ədəd) olacaq:

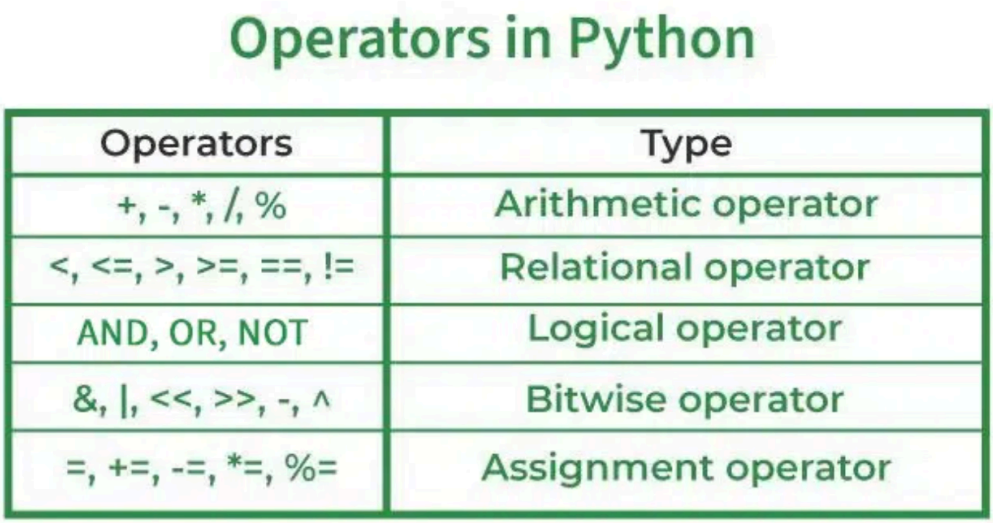
```
>>> 4/2
2.0
```

Əgər başqa bir əməliyyatda bir **tam ədəd** və bir **float** qarışdırırsınızsa, nəticə yenə **float** olacaq:

```
>>> 1 + 2.0
3.0
>>> 2 * 3.0
6.0
>>> 3.0 ** 2
9.0
```

Python hər hansı əməliyyatda float istifadə olunursa, nəticəni avtomatik float olaraq verir, hətta nəticə tam ədəd olsa belə.

Types of Operators in Python



Underscores in Numbers

Uzun ədədləri yazarkən, ədədləri daha oxunaqlı etmək üçün alt xətlər (_) istifadə edərək qruplaşdıra bilərsiniz:

```
>>> universe_age = 14_000_000_000
```

Alt xətlərlə təyin edilmiş ədədi çap etdikdə, Python yalnız **rəqəmləri** göstərir:

```
>>> print(universe_age)
14000000000
```

Python bu tip ədədləri saxlayarkən alt xətləri nəzərə almır. Hətta rəqəmləri üçlükdə qruplaşdırmasanız belə, ədədin dəyəri dəyişməz qalır. Python üçün `1000`, `1_000` və `10_00` eyni ədəddir.

Multiple Assignment

Birdən çox dəyişənə **tək sətirdə** dəyər təyin edə bilərsiniz. Bu üsul proqramınızı qısaldır və oxunmasını asanlaşdırır; ən çox ədədləri ilkin dəyər ilə təyin edərkən istifadə olunur. Məsələn, x, y və z dəyişənlərini sıfıra təyin etmək üçün:

```
>>> x, y, z = 0, 0, 0
```

Dəyişən adlarını vergüllə ayırmalısınız və eyni qayda ilə dəyərləri də ayırmalısınız; Python hər dəyəri müvafiq dəyişənə təyin edəcək. Dəyərlərin sayı dəyişənlərin sayı ilə uyğun olduğu müddətcə Python onları düzgün təyin edəcək.

Constants

Konstant dəyişən kimi bir dəyişəndir ki, proqram boyunca dəyəri dəyişməz qalır. Python-da daxili konstant tipi yoxdur, amma Python proqramçıları **bütün hərfləri böyük** edərək dəyişənin konstant olduğunu göstərirlər və heç vaxt dəyişdirilməməlidir:

```
MAX_CONNECTIONS = 5000
```

Kodunuzda bir dəyişəni konstant kimi istifadə etmək istədiyiniz zaman, dəyişənin adını **bütün böyük hərflərlə** yazın.

Comments

Comments əksər proqramlaşdırma dillərində çox faydalı bir xüsusiyyətdir. İndiyə qədər proqramlarınızda yazdığınız hər şey **Python kodu**dır. Proqramlarınız daha uzun və mürəkkəb olduqda, həll etdiyiniz problemin ümumi yanaşmasını izah edən qeydləri proqramın daxilində əlavə etməlisiniz. Python-da **hash işarəsi (#)** şərh olduğunu göstərir. Kodunuzda hash işarəsindən sonra yazılan hər şey Python interpreter-i tərəfindən nəzərə alınmır.

```
# Say hello to everyone.
print("Hello Python people!")
```

Python birinci sətri nəzərə almayacaq və ikinci sətri icra edəcək:

```
Hello Python people!
```

Commentleri yazmağın əsas səbəbi, kodunuzun nə etməli olduğunu və onu necə işlək vəziyyətə gətirdiyinizi izah etməkdir. Layihə üzərində işləyərkən, bütün hissələrin necə birləşdiyini başa düşürsünüz. Amma

bir müddət sonra layihəyə qayıtdığınızda, detalları unuda bilərsiniz. Kodunuza bir müddət baxıb, hissələrin necə işləməli olduğunu öyrənə bilərsiniz, amma yaxşı şərhlər yazmaq **ümumi yanaşmanı sadə və aydın şəkildə izah edərək vaxtınıza qənaət edər**. Peşəkar proqramçı olmaq və ya digər proqramçılarla əməkdaşlıq etmək istəyirsinizsə, **mənalı şərhlər** yazmalısınız. Bu gün əksər proqram təminatı birgə işləmə yolu ilə hazırlanır; ya bir şirkətdə bir qrup işçi tərəfindən, ya da açıq mənbəli layihədə bir qrup insan tərəfindən. Təcrübəli proqramçılar kodda şərhlərin olmasını gözləyirlər, buna görə proqramlarınıza **indi təsviri şərhlər əlavə etməyə başlamaq** ən yaxşı üsuldur. Kodunuzda **aydın və qısa şərhlər yazmaq**, yeni proqramçı kimi formalaşdıra biləcəyiniz ən faydalı vərdişlərdən biridir. Şərh yazmalı olub-olmadığınızı müəyyənləşdirərkən özünüzə sual verin: bir işi işlək vəziyyətə gətirmək üçün bir neçə yanaşmanı düşünməli oldunuzmu? Əgər belədirsə, həll yolunuz haqqında şərh yazın. Sonradan əlavə şərhləri silmək, az şərhlə yazılmış proqram üçün geri qayıdıb şərhlər yazmaqdan daha asandır.

”””

This is a multi-line comment.

”””