Python les-materialen

# Formattering van String

Met het formattering van String kunt u items in een String invoegen in plaats van items aan elkaar te koppelen met komma’s of String-aaneenschakeling (concatenation). Overweeg als een snelle vergelijking:

player = 'Thomas'  
points = 33  
  
'Last night, '+player+' scored '+str(points)+' points.' # concatenation  
  
f'Last night, {player} scored {points} points.' # string formatting

Er zijn drie manieren om tekenreeksen (Strings) te formatteren. \* Bij de oudste methode worden tijdelijke aanduidingen gebruikt met het modulo %-teken. \* Een verbeterde techniek maakt gebruik van de .format() tekenreeksmethode. \* De nieuwste methode, geïntroduceerd met Python 3.6, gebruikt geformatteerde letterlijke tekenreeksen, genaamd *f-strings*.

Aangezien u waarschijnlijk alle drie de versies in de code van iemand anders zult tegenkomen, beschrijven we ze hier allemaal.

## Opmaak (formattering) met tijdelijke aanduidingen

U kunt %s gebruiken om strings in uw printstatements te injecteren. De modulo % wordt een “String-formattering-operator” genoemd.

print("I'm going to inject %s here." %'something')

I'm going to inject something here.

U kunt meerdere items doorgeven door ze in een tuple te plaatsen na de %-operator.

print("I'm going to inject %s text here, and %s text here." %('some','more'))

I'm going to inject some text here, and more text here.

U kunt ook variabelenamen doorgeven:

x, y = 'some', 'more'  
print("I'm going to inject %s text here, and %s text here."%(x,y))

I'm going to inject some text here, and more text here.

### Conversiemethoden voor het formatteren

Opgemerkt moet worden dat twee methoden %s en %r elk python-object converteren naar een string met behulp van twee afzonderlijke methoden: str() en repr(). We zullen later in de cursus meer over deze functies leren, maar u moet er rekening mee houden dat %r en repr() de *string-representatie* van het object leveren, inclusief aanhalingstekens en eventuele escape-tekens.

print('He said his name was %s.' %'Fred')  
print('He said his name was %r.' %'Fred')

He said his name was Fred.  
He said his name was 'Fred'.

Als een ander voorbeeld voegt \t een tab in een string in.

print('I once caught a fish %s.' %'this \tbig')  
print('I once caught a fish %r.' %'this \tbig')

I once caught a fish this big.  
I once caught a fish 'this \tbig'.

De %s-operator converteert alles wat het ziet in een string, inclusief gehele getallen en floats. De operator %d converteert getallen eerst naar gehele getallen, zonder afronding. Noteer het verschil hieronder:

print('I wrote %s programs today.' %3.75)  
print('I wrote %d programs today.' %3.75)

I wrote 3.75 programs today.  
I wrote 3 programs today.

### Opvulling en precisie van drijvende-kommagetallen

Drijvende-kommagetallen gebruiken het formaat %5.2f. Hier zou 5 het minimum aantal karakters zijn dat de string zou moeten bevatten; deze kunnen worden opgevuld met witruimte als het hele nummer niet zoveel cijfers heeft. Daarnaast staat .2f voor het aantal cijfers achter de komma. Laten we enkele voorbeelden bekijken:

print('Floating point numbers: %5.2f' %(13.144))

Floating point numbers: 13.14

print('Floating point numbers: %1.0f' %(13.144))

Floating point numbers: 13

print('Floating point numbers: %1.5f' %(13.144))

Floating point numbers: 13.14400

print('Floating point numbers: %10.2f' %(13.144))

Floating point numbers: 13.14

print('Floating point numbers: %25.2f' %(13.144))

Floating point numbers: 13.14

Ga voor meer informatie over tekenreeksopmaak met tijdelijke aanduidingen naar https://docs.python.org/3/library/stdtypes.html#old-string-formatting

### Meerdere formattering

Niets verbiedt het gebruik van meer dan één conversietool in dezelfde printopdracht:

print('First: %s, Second: %5.2f, Third: %r' %('hi!',3.1415,'bye!'))

First: hi!, Second: 3.14, Third: 'bye!'

## Formattering met .format() -methode

Een betere manier om objecten op te maken in uw strings voor printstatements is met de string .format() methode. De syntaxis is:

'String here {} then also {}'.format('something1','something2')

Bijvoorbeeld:

print('This is a string with an {}'.format('insert'))

This is a string with an insert

### De .format() methode heeft verschillende voordelen ten opzichte van de %s placeholder methode:

#### 1. Ingevoegde objecten kunnen worden opgeroepen op indexpositie:

print('The {2} {1} {0}'.format('fox','brown','quick'))

The quick brown fox

#### 2. Ingevoegde objecten kunnen trefwoorden worden toegewezen:

print('First Object: {a}, Second Object: {b}, Third Object: {c}'.format(a=1,b='Two',c=12.3))

First Object: 1, Second Object: Two, Third Object: 12.3

#### 3. Ingevoegde objecten kunnen opnieuw worden gebruikt, waardoor duplicatie wordt voorkomen:

print('A %s saved is a %s earned.' %('penny','penny'))  
# vs.  
print('A {p} saved is a {p} earned.'.format(p='penny'))

A penny saved is a penny earned.  
A penny saved is a penny earned.

### Uitlijning, opvulling en precisie met .format()

Binnen de accolades kunt u veldlengtes, links/rechts uitlijningen, afrondingsparameters en meer toewijzen

print('{0:8} | {1:9}'.format('Fruit', 'Quantity'))  
print('{0:8} | {1:9}'.format('Apples', 3.))  
print('{0:8} | {1:9}'.format('Oranges', 10))

Fruit | Quantity   
Apples | 3.0  
Oranges | 10

Standaard lijnt .format() tekst links uit, cijfers rechts. U kunt een optionele <,^ of > doorgeven om een linker-, midden- of rechteruitlijning in te stellen:

print('{0:<8} | {1:^8} | {2:>8}'.format('Left','Center','Right'))  
print('{0:<8} | {1:^8} | {2:>8}'.format(11,22,33))

Left | Center | Right  
11 | 22 | 33

U kunt de uitlijningsoperator vooraf laten gaan door een opvulteken (padding)

print('{0:=<8} | {1:-^8} | {2:.>8}'.format('Left','Center','Right'))  
print('{0:=<8} | {1:-^8} | {2:.>8}'.format(11,22,33))

Left==== | -Center- | ...Right  
11====== | ---22--- | ......33

Veldbreedtes en zweefnauwkeurigheid worden op dezelfde manier behandeld als tijdelijke aanduidingen. De volgende twee afdrukinstructies zijn equivalent:

print('This is my ten-character, two-decimal number:%10.2f' %13.579)  
print('This is my ten-character, two-decimal number:{0:10.2f}'.format(13.579))

This is my ten-character, two-decimal number: 13.58  
This is my ten-character, two-decimal number: 13.58

Merk op dat er 5 spaties achter de dubbele punt staan, en 5 karakters in beslag genomen door 13.58, voor een totaal van tien karakters.

Ga voor meer informatie over de string .format()-methode naar https://docs.python.org/3/library/string.html#formatstrings

## Geformatteerde String Literals (f-strings)

Geïntroduceerd in Python 3.6, bieden f-strings verschillende voordelen ten opzichte van de oudere .format() string-methode die hierboven is beschreven. Ten eerste kun je variabelen van buitenaf direct in de string opnemen in plaats van ze als argumenten door te geven via .format(var).

name = 'Fred'  
  
print(f"He said his name is {name}.")

He said his name is Fred.

Pass !r to get the string representation:

print(f"He said his name is {name!r}")

He said his name is 'Fred'

#### Float-formattering volgt "resultaat: {value:{width}.{precision}}"

Waar je met de .format() methode {value:10.4f} ziet, met f-strings kan dit {value:{10}.{6}} worden

num = 23.45678  
print("My 10 character, four decimal number is:{0:10.4f}".format(num))  
print(f"My 10 character, four decimal number is:{num:{10}.{6}}")

My 10 character, four decimal number is: 23.4568  
My 10 character, four decimal number is: 23.4568

Merk op dat met f-strings *precisie* verwijst naar het totale aantal cijfers, niet alleen die achter de komma. Dit sluit nauwer aan bij wetenschappelijke notatie en statistische analyse. Helaas worden f-snaren niet rechts van het decimaalteken weergegeven, zelfs als de precisie het toelaat:

num = 23.45  
print("My 10 character, four decimal number is:{0:10.4f}".format(num))  
print(f"My 10 character, four decimal number is:{num:{10}.{6}}")

My 10 character, four decimal number is: 23.4500  
My 10 character, four decimal number is: 23.45

Als dit belangrijk wordt, kun je altijd de syntaxis van de methode .format() gebruiken in een f-string:

num = 23.45  
print("My 10 character, four decimal number is:{0:10.4f}".format(num))  
print(f"My 10 character, four decimal number is:{num:10.4f}")

My 10 character, four decimal number is: 23.4500  
My 10 character, four decimal number is: 23.4500

Ga voor meer informatie over geformatteerde de formatteringen van String naar https://docs.python.org/3/reference/lexical\_analysis.html#f-strings

Dat is de basis van String-formattering!