### HELLO WORLD

Seštevanje:

>>> print 1 + 1

2

Množenje:

>>> print 1 \* 3

3

Potenciranje:

>>> print 2\*\*16

65536

"": Spremenljivke se v dvojnih narekovajih ne interpolirajo. Se pa zato ohranijo escape characterji, kot so \t (tab), \n (new line)

Izpiši text:

>>> print "Hello World"

Hello World

Uporaba escape characterjev \t (tab = 8 empty spacov) in 2x new line:

>>> print "Hello\tWorld\n\n"

Hello World

>>>

Izpis vrednosti spremenljivke:

>>> name = "Dragan"

>>> print name

Dragan

### VARIABLES

V pythonu, spremenljivki ni treba določiti tipa. Python bo tip samodejno določil glede na to ali je vrednost spremenljivke samo število (integer), če ima v vrednosti kakšen string (string), če je delimitiran z vejico/podpičjem/... (list) itd.

Spremenljivka bo imela enega izmed naslednjih tipov: (string,int,float,lists,tuples,dictionary).

Da bi ugotovili tip spremenljivke, ima python vgrajeno funkcijo type(), kjer v oklepaje vpišemo ime spremenljivke.

Funkcija type() je funkcija za ugotavljanje, v katerem bloku v memoryju je shranjena spremenljivka. V oklepaje vpišemo ime spremenljivke, ki nas zanima. Zakaj to rabimo? Če imamo npr. dodeljeno spremenljivko Price = 395.00. Nato dodelimo Price2 = Price. Ti dve spremenljivki bosta sedaj imeli dodeljen isti blok v memoryju, saj python ve, da je spremenljivka Price2 le alias spremenljivke Price. Tako Price2 ne zavzema dodatnega memoryja. Če bi kasneje vrednost spremenljivke Price spremenili, bi Price2 obdržal originalno vrednost Price in ne nove updejtane.

Imena spremenljivk so case sensitive.

***Primeri:***

Spremenljivka Product bo imela vrednost "LinuxCBT Scripting Edition". Najprej njeno vrednost izpišemo, nato z vgrajeno python funkcijo type preverimo tip spremenljivke. Tip je string:

>>> Product = "LinuxCBT Scripting Edition"

>>> print Product

LinuxCBT Scripting Edition

>>> type(Product)

<type 'str'>

Enako naredimo s spremenljivko Price, le da bo tu tip seveda integer;

>>> Price = 395

>>> print Price

395

>>> type(Price)

<type 'int'>

Vrednost spremenljivke Price damo v dvojne narekovaje. Katerakoli vrednost ima spremenljivka, če bo znotraj narekovajev, se tip spremeni v string:

>>> Price = "395"

>>> type(Price)

<type 'str'>

Vrednosti spremenljivke Price damo decimalno piko. Tip spremenljivke se spremeni v float. Nato spremenljivki Price2 dodelimo vrednost spremenljivke Price. Preverimo njuni lokaciji v memoryju s funkcijo id(), in sta seveda enaki. Nato spremenimo vrednost spremenljivke Price in spet preverimo njuni lokaciji v memoriju. Tokrat sta različni, saj je Price2 obdržal originalno vrednost.

>>> Price = 395.00

>>> type(Price)

<type 'float'>

>>> id(Price)

31335176

>>> Price2 = Price

>>> id(Price)

31335176

>>> id(Price2)

31335176

>>> Price = 495.00

>>> id(Price)

31335192

>>> id(Price2)

31335176

#!/usr/bin/python

# Author: Dragan

# Date: 4.11.2013

# Purpose: Basic helloworld script

# Dodelimo spremenljivko "message" z vrednostjo "Hello World!".

# Izpisemo dve prazni vrstici, vrednost spremenljivke "message" in zatem se dve

# prazni vrstici.

message = "Hello World!"

print "\n\n", message, "\n\n"

#END

### BASIC MATH

Vrstni red operacij, ki jih python izvaja, sledi osnovnim matematičnim pravilom PEMDAS (US - Parenthesies, Exponents, Multiplication, Division, Addition. Subtraction) ali BEMDAS (CAN - Brackets, Exponents, Multiplication, Division, Addition, Subtraction).

Deljenje (multiplication) ima enako težo kot deljenje (division). Seštevanje (addition) ima enako težo kot odštevanje (subtraction). To pomeni, da če so napisani v enaki vrstici, jih bo python izvedel od leve proti desni.

Lahko pa vrstni red operacij prilagodimo našim potrebam.

***Primeri:***

>>> print 2\*\*8

256

>>> print 2\*\*8 \* 4 / 4 + 44 - 44

256

>>> print 2\*\*8

256

>>> print 2\*\*8 \* 4

1024

>>> print 2\*\*8 \* 4 / 4

256

>>> print 2\*\*8 \* 4 / 4 + 44

300

>>> print 2\*\*8 \* 4 / 4 + 44 - 44

256

Deljenje in množenje v isti vrstici:

>>> print 100 / 2 \* 3

150

Spremenimo vrstni red operacij z oklepajem (parenthesies). Če so vsa števila v računu cela (integer) potem bo python vrnil rezultat kot integer:

>>> print 100 / (2 \* 3)

16

Enak račun kot zgoraj, le da smo številu 100 dodali decimalno piko (100.0). V tem primeru bo python rezultat vrnil kot float:

>>> print 100.0 / (2 \* 3)

16.6666666667

Še en podoben primer. 100 ni v celoti deljivo s 16. Rezultat je 6, ampak je še ostanek:

>>> print 100 / 16

6

Izpis ostanka z znakom %. Rezultat tega računa nam pove, da je 100 deljivo s 16 x-krat, pri tem pa bo ostanek 4.

>>> print 100 % 16

4

Izpis rezultata in ostanka v eni vrstici:

>>> print "100 / 16 is:", 100 / 16, " Remainder", 100 % 16

100 / 16 is: 6 Remainder 4

### STDIN INPUT

V Linux sistemu so komande vpisane v command line preko tipkovnice upoštevane kot standard input. Tu bomo prikazali, kako te komande zberemo. Python ima za te namene vgrajeni dve funkciji: input in raw\_input. Funkcijo input je najbolje uporabljati takrat, ko potrebujemo od inputa vrednost integer. Če od inputa potrebujemo string, potem uporabimo funkcijo raw\_input. Funkcijo raw\_input lahko uporabimo za zbiranje kateregakoli uporabnikovega inputa, vključno z integerji, vendar bodo ti označeni kot string. Ko pa bo potrebno na njih izvajati operacije na teh integerjih označenih kot"string", bodo operacije vseeno uspele.

***Primeri:***

Izpis dveh stringov:

>>> print "first", "second"

first second

Konkatenacija dveh stringov. Ker string pač ni število, ga python ne more sešteti, zato bo znak + imel funckijo konkatenacije, torej leplenja:

>>> print "first" + "second"

firstsecond

Množenje stringa. če string zmnožimo, se bo preprosto konkatenirano izpisal x-krat:

>>> print "hello"\*3

hellohellohello

Enak primer z dodanim new line characterjem:

>>> print "hello\n"\*3

hello

hello

hello

>>>

Uporabnika vprašamo "What is your message?". Odgovor, ki ga bo vtipkal, shranimo v spremenljivko message. Če ne, bi se vrednost samo izpisala ter nato izgubila. Ker bo odgovor uporabnika string, je uporabljena funkcija raw\_input. Uporabnik vpiše svoj odgovor kot "Hello World!". Nato izpišemo vrednost spremenljivke message:

>>> message = raw\_input("What is your message? ")

What is your message? Hello World!

>>> print message

Hello World!

Enaka fora kot zgoraj, le da je tu pričakovan odgovor uporabnika integer, zato funkcija input. Če bi uporabnik vpisal kakšen string, bi python vrnil error.

>>> var1 = input("Please specify an integer: ")

Please specify an integer: 100

>>> print var1

100

#!/usr/bin/python

# Author: Dragan

# Date: 5.11.2013

# Purpose: Illustrate input functions

# Preprosta skripta, ki bo izracunala koliko casa bomo ziveli, glede na trenutno

# starost, in starost, ki jo pricakuje uporabnik. Vejica za vsakim "print name"

# zato, ker zelimo, da sta uporabnikovo ime in vprasanje/odgovor v isti vrstici.

# Po defaultu je brez vejice na koncu print stavka, new line character.

print "------- Life Expectency v0.99 --------"

name = raw\_input("What is your name? ")

print name,

age = input("what is your age? ")

print name,

expect = input("to what age do you expect to live? ")

print name,

print "based on our calculations, you will live for", expect - age, "more years$

print "Thank you for using our software!"

#END

### STRINGS 1

Primeri:

Najprej spremenljivki message dodelimo string vrednost "new string". Spremenljivko nato izpišemo:

>>> message = "new string"

>>> print message

new string

Funkcija len(message) nam pove dolžino stringa v spremenljivki message, torej vse znake in prazne prostore:

>>> len(message)

10

Število v oglatih oklepajih nam pove, kateri znak string vrednosti spremenljivke message želimo izpisati. 0 je seveda prvi znak, torej n ("new string"):

>>> print message[0]

n

Ne moremo pa na tak način spremeniti znaka v stringu. Stringov ne moremo spreminjati:

>>> message[0] = m

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'm' is not defined

Še en način izpisa celotnega stringa:

>>> print message[:]

new string

Izpis samo dela stringa, od prvega do šestega znaka. Zadnji znak (torej šesti) se ne izpiše:

>>> print message[0:5]

new s

For loop, ki gre čez string. Argument "letter" bo dobil vsak loop vrednost enega characterja v string vrednosti spremenljivke message in vsakič to vrednost izpisal. Ker ima print po defaultu na koncu vedno new line character, je rezultat navpično izpisan string. Pozor! Pri for loopu, ko pritisnemo enter za "for letter in message:", se nam prikažejo tri pikice. Takoj za temi tremi pikicami moramo narediti vsaj en presledek (ali tab), drugače python vrne error.

>>> for letter in message:

... print letter

...

n

e

w

s

t

r

i

n

g

Primerjava dveh stringov. Spremenljivki message2 smo dodali enako vrednost "new string" kot spremenljivki message. Zato tak rezultat primerjave stringov:

>>> if message == message2:

... print "They match!"

...

They match!

### STRINGS 2

Python ima mnogo modulov. Tu bomo importali enega za manipulacijo stringov imenovanega "string". Dokumentacijo za ta modul lahko najdemo na pythonovi spletni strani. Modul importamo s komando "import string". String je ime modula.

Upper je ime prve izmed metod tega modula, ki jih bomo uporabili na naši spremenljivki message. Ta metoda bo izpisala vrednost spremenljivke message (vrednost je "new string") z velikimi začetnicami. Vrednost spremenljivke se NE spremeni če to vrednost ne dodelimo spremenljivki message.

Ostale metode, ki jih bomo še uporabili so string.lower, string.capitalize, string.capwords, string.split in string.join.

Zakaj so te metode pomembne za sistemsko administriranje? Če pomisliš, kako so zgrajeni log fajli v linuxu. Načeloma so pač vrstice eventov ki so se logirali. Ena vrstica je v bistvu string ločen z whitespaci. Če bi uporabili python za pregled logov (python ima default delimiter whitespace), bi lahko celotno vrstico razdelili v posamezne elemente lista. In to je zelo pomembno. Npr v apache logu je en element datum, drugi soucre IP, tretji bi bil source port, četrti destination IP, peti destination port itd.

Module je najbolje importati na začetku skripte. Če metode uporabimo brez importanja modula, python vrne error.

***Primeri:***

String je objekt (modul). Upper je metoda (funkcija) objekta string, ki sprejema argumente. Argument, ki smo ji ga mi poslali je message.

>>> string.upper(message)

'NEW STRING'

String vrednost spremenljivke message smo spremenili v velike začetnice:

>>> message = string.upper(message)

>>> print message

NEW STRING

String vrednost spremenljivke message smo spremenili nazaj v male začetnice:

>>> message = string.lower(message)

>>> print message

new string

Prvi besedi (prvemu stringu) v vrednosti spremenljivke message daj veliko začetnico. Delimiter mora biti whitespace. Če ni, potem moramo delimiter določiti:

>>> message = string.capitalize(message)

>>> print message

New string

Vsakemu stringu v vrednosti spremenljivke message daj veliko začetnico. Delimiter whitespace:

>>> message = string.capwords(message)

>>> print message

New String

Vrednost spremenljivke message, ki je "New String" bomo ločili in iz nje naredili list, v katerem bosta vrednosti "New" in "String". Vrednosti nismo še shranili, le izpisana je:

>>> string.split(message)

['New', 'String']

Tu je ta vrednost sedaj shranjena. Spremenljivka message ima sedaj dva elementa, element 0 (New) in element 1 (String):

>>> message = string.split(message)

>>> print message

['New', 'String']

Izvedemo obratno operacijo in list vrednost spremenljivke message združimo nazaj v string:

>>> message = string.join(message)

>>> print message

New String

### STRINGS 3

Zgornji primeri zapisani v skripti.

#!/usr/bin/python

# Author: Dragan

# Date: 7.11.2013

# Purpose: Illustrate string manipulation functions

# Importamo modul strings.

import string

# Spremenljivkama message in message2 dodelimo vrednost "new string".

message = "new string"

message2 = "new string"

# Izpisemo vrednost spremenljivke message.

print message

# Izpise dolzino vrednosti spremenljivke message (len.message).

print message, "contains", len(message), "characters"

# Izpise prvi znak vrednosti spremenljivke message (message[0]).

print "The first character in", message, "is:", message[0]

# Izpisemo elemente od 0 do 4, brez 4.

print "Example of slicing", message, message[0:4]

# Pri for zanki je pomembno da vsebino for zanke zamaknemo za vsaj 1 whitespace

# ali se bolje tab v desno. Drugace python ne dela.

for letter in message:

print letter

# Tudi tu zamaknemo vsebio funkcije v desno. Nujno.

if message == message2:

print "They match!"

# Vrednosti spremenljivke message spremenimo v velike zacetnice, in nato se v

# male zacetnice.

message = string.upper(message)

print message

message = string.lower(message)

print "After reassignment ", message

# Izpisemo vrednost spremenljivke message tako, da ima prva beseda

# veliko zacetnico.

print string.capitalize(message)

# Izpisemo vse besede tako, da imajo veliko zacetnico.

print string.capwords(message)

# Vse besede spremenljivke message (whitespece delimiter) izpisemo tako, da

# besede razdelimo v locene elemente lista.

print string.split(message)

# Elemente lista zdruzimo. string.join je nasprotje od string.split.

print string.join(message)

#END

## LISTS 1

List nam omogoča shranjevanje stringov, integerjev, floatov... znotraj ločenih elementov, do katerih lahko dostopamo. Po defaultu je delimitiran z vejico.

Primeri:

Definiramo list "numlist":

numlist = [1,2,3,4,5]

Izpišemo list:

>>> print numlist

[1, 2, 3, 4, 5]

Zamenjamo vrstni red številk v listu. Uporabimo metodo reverse:

>>> numlist.reverse()

>>> print numlist

[5, 4, 3, 2, 1]

Ustvarimo še list numlist2 s številkami od 6-10. Ta numlist2 nato appendamo numlistu. Append pomeni dati celoten numlist2 v en element numlista. Ustvari se nested array/list.

>>> numlist2 = [6,7,8,9,10]

>>> numlist.append(numlist2)

>>> print numlist

[1, 2, 3, 4, 5, [6, 7, 8, 9, 10]]

Izpišemo prvi element lista:

>>> print numlist[0]

1

Izpišemo šesti element lista:

>>> print numlist[5]

[6, 7, 8, 9, 10]

Izpišemo prvi element gnezdenega (nested) elementa:

>>> print numlist[5][0]

6

Izbrišemo element lista. Po defaultu se bo izbrisal zadnji element (LIFO - Last in first out, ali poznan pod imenom "stack". Preveri kaj je stack, nisem čisto siguren. V vsakem primeru se tu obravnava numlist kot objekt in pop kot metoda):

>>> numlist.pop()

10

Listu numlist dodamo numlist2. Za razliko od .append funkcije, ki je vsebino numlist2 dala v en element numlista, pa se tukaj dejansko dodajo elementi za vsako vrednost numlista2:

>>> numlist.extend(numlist2)

>>> print numlist

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Izbrišemo prvi element (FIFO - first in first out, ali poznano pod imenom queuing):

>>> numlist.pop(0)

1

Dodamo element v list. Uporabimo metodo .insert, ki je sestavljena iz dveh argumentov. Iz indexa in vrednosti. Index pove, kam bomo vrednost v list vmestili:

>>> print numlist

[2, 3, 4, 6, 7, 8, 9]

>>> numlist.insert(0,1)

>>> print numlist

[1, 2, 3, 4, 6, 7, 8, 9]

### LIST 3

Python vsebuje funkcijo range(), ki izpiše range integerjev od 0 do vrednosti, ki jo damo v oklepaje (izključno z vrednostjo v oklepajih).

Primeri:

Izpišemo vrednosti od 0 do 9 v obliki lista:

>>> range(10)

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Spremenljivki numlist3 dodelimo vrednosti od 0 - 10:

>>> numlist3 = range(11)

>>> print numlist3

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Drug način uporabe ranga. Določimo kje naj se vrednosti začnejo in kje končajo (zadnja številka ni všteta):

>>> range(1,10)

[1, 2, 3, 4, 5, 6, 7, 8, 9]

Tretji način range funkcije. določimo začetno in končno vrednost ter korake, po koliko naj se vrednosti večajo:

>>> range(0,100,2)

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98]

Naredimo stringlist, torej list v katerem je več stringov. Python vrednost interpretira kot string, če je vrednost v narekovajih:

>>> stringlist = ["LinuxCBT", "Scripting", "Edition"]

>>> print stringlist

['LinuxCBT', 'Scripting', 'Edition']

Stringe v listu postavimo v obratni vrstni red:

>>> stringlist.reverse()

>>> print stringlist

['Edition', 'Scripting', 'LinuxCBT']

Stringlistu appendamo stringlist2. Enako kot pri številkah, append postavi vsebino appendanega lista v en element lista kateremu appendamo:

>>> print stringlist

['Edition', 'Scripting', 'LinuxCBT', ['Python', 'Bash']]

Izpis enega izmed gnezdenih elementov:

>>> print stringlist[3][1]

Bash

Zbrišemo zadnji element lista:

>>> stringlist.pop()

['Python', 'Bash']

>>> print stringlist

['Edition', 'Scripting', 'LinuxCBT']

Združimo dva lista:

>>> stringlist.extend(stringlist2)

>>> print stringlist

['Edition', 'Scripting', 'LinuxCBT', 'Python', 'Bash']

Vstavimo element "Perl" v list pod index številko 3 (element je lahko tudi drug list):

>>> stringlist.insert(3,"Perl")

>>> print stringlist

['Edition', 'Scripting', 'LinuxCBT', 'Perl', 'Python', 'Bash']

### LISTS 4

#!/usr/bin/python

# Author: Dragan

# Date: 14.11.2013

# Purpose: Illustrate lists of strings

# Importamo modul string

import string

# Vrne vrednosti od 0-9.

print range(10)

# Vrne vrednosti od 1-10.

print(1,11)

# Naredimo dva lista, ki vsebujeta stringe.

stringlist = ["LinuxCBT","Scripting","Edition"]

stringlist2 = ["Python","BASH"]

# Izpisemo list stringlist.

print stringlist

# Stringle v listu postavimo v obranem vrstnem redu in list izpisemo..

stringlist.reverse()

print stringlist

# Stringlistu appendamo (v bistvu nestamo) stringlist2.

stringlist.append(stringlist2)

print stringlist

# Izpisemo tocno dolocen elemen. Tu izpisemo prvi element iz tretjega elementa.

print stringlist[3][0]

# Izpisemo celoten nested element (ki se nahaja seveda na tretjem mestu).

print stringlist[3][:]

# Izbrisemo zadnji element lista.

stringlist.pop()

# Zdruzimo dva lista.

stringlist.extend(stringlist2)

print stringlist

# Vstavimo string v list, pod pozicijo 3 z vsebino "Perl".

stringlist.insert(3,"Perl")

print stringlist

# ---------Processing of ficticious Apache web server log file

# V spremenljivko logfile damo string, ki izgleda kot vsebina log fajla.

logfile = "20131114 192.168.1.100 1100 192.168.1.104 80 404 index.php"

print logfile

# Preverimo za foro tip spremenljivke.

print type(logfile)

# Pretvorimo string vsebino spremenljivke logfile v list, in pretvorjeno

# vrednost shranimo v novo spremenljivko logfile2. Originalna je nedotaknjena.

logfile2 = string.split(logfile)

print logfile2

# Slice logfile2 to extract important fields.

# Odrezemo del lista od 0-5 pozicije.

logfile3 = logfile2[0:5]

print logfile3

# Join logfile3 into a flat string

logfile4 = string.join(logfile3)

print logfile4

# Preverimo, ce se je res pretvoril v string.

print type(logfile4)

#END

### DICTIONARIES

Preden gremo k dictionarijem, povejmo, kaj je razlika med List in Tuple. Elemente lista lahko spreminjamo, elementov tupla pa ne (reče se temu "immutable". Je read only.

Dictionary vsebuje key/value pair. Key je ključna unikatna beseda, tako kot v slovarju. Value pač vrednost te besede (razlaga v sovarju).

Ena stvar, ki jo ne štekam dobro: če ima v dictionariju key več kot eno vrednost, moramo uporabiti immutable type kot je tuple.

Primeri:

Naredili smo list s tremi različnimi stringi, vsebino izpisali, preverili tip spremenljivke in element na drugi poziciji zamenjali z novim stringom:

>>> products = ["LinuxCBT", "Scripting Edition", "RedHat Edition"]

>>> print products

['LinuxCBT', 'Scripting Edition', 'RedHat Edition']

>>> type(products)

<type 'list'>

>>> products[2] = "Debian Edition"

>>> print products

['LinuxCBT', 'Scripting Edition', 'Debian Edition']

Naredimo tuple "products2" in ga primerjamo z listom "products":

>>> products2 = ("LinuxCBT", "Scripting Edition", "RedHat Edition")

>>> print products

['LinuxCBT', 'Scripting Edition', 'Debian Edition']

>>> print products2

('LinuxCBT', 'Scripting Edition', 'RedHat Edition')

>>> type(products2)

<type 'tuple'>

Poskusimo spremeniti vsebino tupla:

>>> products2[2] = "Debian Edition"

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'tuple' object does not support item assignment

Naredimo dictionary "LinuxCBT". Leva stran od dvopičja je Key in desna je value. Naš dictionary ima dva key/value para:

>>> LinuxCBT = {'Scripting': 395, 'RedHat': 595}

>>> LinuxCBT

{'RedHat': 595, 'Scripting': 395}

Izpišemo value za določen key:

>>> LinuxCBT['Scripting']

395

### Dodelimo nov key/value par:

>>> LinuxCBT['Debian'] = 395

>>> LinuxCBT

{'RedHat': 595, 'Scripting': 395, 'Debian': 395}

Spremenimo value določenega keya:

>>> LinuxCBT['Debian'] = 495

>>> LinuxCBT

{'RedHat': 595, 'Scripting': 395, 'Debian': 495}

Ker je dictionary objekt, lahko uporabimo njemu namenjene metode. Tu je metoda, ki izpiše samo keye:

>>> LinuxCBT.keys()

['RedHat', 'Scripting', 'Debian']

Tu uporabimo metodo, ki izpiše samo values:

>>> LinuxCBT.values()

[595, 395, 495]

Izbrišemo določen key/value par:

>>> del LinuxCBT['RedHat']

>>> LinuxCBT

{'Scripting': 395, 'Debian': 495}

Loop čez vse key/value pare v dictionaryju z vgrajeno metodo iteritems. k,v (key,value) sta spremenljivki:

>>> for k,v in LinuxCBT.iteritems():

... print k,v

...

Scripting 395

Debian 495

Dictionariju dodamo key "Suite" , ki bo imel več vrednosti. Te vrednosti poberemo iz lista "suiteprice":

>>> suiteprice = [395,595]

>>> print suiteprice

[395, 595]

>>> LinuxCBT['Suite'] = suiteprice

>>> LinuxCBT

{'Suite': [395, 595], 'Scripting': 395, 'Debian': 495}

Tako zgleda zdaj novi dictionary skorzi iteritems metodo:

>>> for q,r in LinuxCBT.iteritems():

... print q,r

...

Suite [395, 595]

Scripting 395

Debian 495

### CONDITIONALS 1

Tako kot v ostalih jezikih, se if komanda uporablja za testiranje če je pogoj izpolnjen.

<, <=, >, >=, ==, ! ali <>

Primeri:

Dodelimo spremenljivki min in max naenkrat. Izpišemo njuno vrednost in nato izvedemo if stavek. V prvi vrstici if stavka daj dvopičje in nato v naslednji pred pisanjem daj na začetko whitespace ali še bolje tab:

>>> min, max = 8,9

>>> print min

8

>>> print max

9

>>> if min < max:

... print min, "is less then", max

...

8 is less then 9

Pri else stavku ne dajaj whitespacow!:

>>> if min < max:

... print min, "is less then", max

... else:

... print "no luck"

...

8 is less then 9

#!/usr/bin/python

# Author: Dragan

# Date: 15.11.2013

# Purpose: Illustrate input functions

# Preprosta skripta, ki bo izracunala koliko casa bomo ziveli, glede na trenutno

# starost, in starost, ki jo pricakuje uporabnik. Vejica za vsakim "print name"

# zato, ker zelimo, da sta uporabnikovo ime in vprasanje/odgovor v isti vrstici.

# Po defaultu je brez vejice na koncu print stavka, new line character.

print "------- Life Expectency v0.99 --------"

name = raw\_input("What is your name? ")

print name,

age = input("what is your age? ")

print name,

expect = input("to what age do you expect to live? ")

print name,

# Extending life1.py to incorporate conditional testing.

# Izracunamo koliko let mu se preostane.

timeleft = expect - age

# Vprasamo uporabnika naj ugane koliko let mi se preostane.

answer = input("How much time have you got left? ")

# Ce uporabnik ni uganil let, potem izpisemo Sorry in skripto zakljucimo.

# Ce je, potem se izvrsi else statement in skripta se zakljuci.

if answer != timeleft:

print "Sorry", name, "That is incorrect!"

else:

print "Bingo, you will live for", timeleft, "years"

#END

### CONDITIONALS 2

Skripte včasih potrebujejo različno število parametrov, da bi se uspešno izvršile. V pythonu (in še drugih jezikih) obstaja spremenljivka imenovana argv, ki je del modula sys (import sys). sys.argv je list, sestavljen iz stringov in/ali integerjev, odvisno pač od parametrov, ki smo jih poslali skripti. V prvem elementu tega lista je vedno ime skripte, ne glede na to ali smo poslali skripti kakšen argument (parameter) ali ne. V naslednjih elementih lista pa so shranjeni parametri, ki smo jih skripti poslali.

#!/usr/bin/python

# Author: Dragan

# Date: 16.11.2013

# Purpose: Expound on conditional branching

# Importamo sys modul, da spremenljivka argv deluje.

import sys

# To bo nase stevilo minimalno zahtevano stevilo parametrov - 1.

threshold = 5

# Dolzino lista sys.argv damo v spremenljivko.

arglen = len(sys.argv)

# Ce je dolzina lista sys.argv manjsa od stevila parametrov, ki smo jih poslali.

# Default dolzina je vedno 1, ker sys.argv vedno vsebuje v listu ime skripte,

# kot en element lista.

# Povemo koliko argumentov skripta potrebuje in koliko smo jih ze vnesli.

if arglen < threshold:

print "Script requires at least", threshold - 1, "arguments!"

print "You've entered ", arglen - 1, "arguments."

# Ce je stevilo argumentov med 5 in 8, povemo koliko smo jih vnesli in katerem

# delu conditional stavka smo.

elif arglen >= 5 and arglen <= 8:

print "You've entered", arglen - 1, "arguments."

print "You are now in branch 2"

# Ce je stevilo parametrov med 8 in 10 izpisi to stevilo, izpisi branch in

# s for zanko izpisi vse parametre.

elif arglen >= 8 and arglen <= 10:

print "You've entered", arglen - 1, "arguments."

print "You are now in branch 3"

for i in sys.argv:

print i

# Ce je stevilo argumentov prekoraceno. Pozor na dvopicje!

else:

print sys.argv[0], "Accepts only 9 arguments"

#END