(C) 2024 OSTC GmbH (http://www.ostc.de)


```
================================================================================
ÃM-^Dhnlichkeiten und Unterschiede von Python und Perl
(C) 2013-2021 T.Birnthaler OSTC GmbH
================================================================================

* In Perl geschweifte Klammern {...} pro (eingeschachtelten) Block notwendig.
  In Python per EinrÃ¼ckung erledigt (die man auch in Perl macht ;-).

* In Perl bei print "\n" fÃ¼r Zeilenvorschub anzugeben (eigentlich Normalfall).
  In Python mit "sep=''" Zeilenvorschub unterdrÃ¼cken (selten notwendig).

* In Perl ";" nach jeder Anweisung notwendig.
  In Python automatisch per Zeilenende.

* In Perl Variablen-PrÃ¤fix $/@/%/&/* je nach Variablentyp notwendig.
  In Python gibt es keine PrÃ¤fixe (der Datentyp ist beim Wert vermerkt).

* In Perl stehen Funktionsparameter gesammelt als Liste in "@_",
  ihre Benennung muss selbst durchfÃ¼hren per my ($a1, $a2, ...) = @_;
  In Python muss man die Funktionsparameter benennen,
  (und Defaultwerte sind dazu angebbar).
  --> Viel freundlichere und klarere Schnittstelle!

* In Perl ist der Namensraum "__main__" zu voll.
  In Python sind viele Funktionen in Bibliotheken ausgelagert
  oder mit einem Datentyp assoziiert und damit in getrenntem Namensraum.

* Fehlerbehandlung in Perl umstÃ¤ndlich per "eval" gelÃ¶st.
  In Python per "try" ... "except" Teil der Sprache.

* In Perl sind die vielen einbuchstabigen Spezialvariablen $_, @_, %_, $?, $!,
  $#, $@, ... schwer zu merken. Ebenso ist ihre implizite Verwendung zusammen
  mit vielen Funktionen schwer zu merken und inkonsistent.

* In Perl ist aus der Aufrufsyntax nicht ableitbar, ob eine Funktion eine
  Variable prinzipiell verÃ¤ndern kann oder nicht.
  In Python:  erg = FUNK(var, ...)  --> kann Variable NICHT verÃ¤ndern
                    var.FUNK(...)   --> kann Variable verÃ¤ndern (muss aber nicht)

* In Perl wirkt die bjektorientierte Programmierung "aufgesetzt" ("bless").
  In Python ist sie ein (syntaktischer) Traum.

* Automatische Konvertierung String <-> Zahl in Perl ist zwar superbequem,
  aber auch sehr fehleranfÃ¤llig. Python erwartet, dass man selber konvertiert.

* Modulsystem in Python einfach + leistungsfÃ¤hig, Modul automatisch Namensraum.
  In Perl sind Module und NamensrÃ¤ume (Package) inhaltlich getrennt.

* In Python Syntax fÃ¼r Zugriff auf Elemente einer Sequenz (String, Liste, Tupel)
  sowie Mappings (Dictionary, Set) weitgehend gleich.
  In Perl PrÃ¤fix + Klammernsorte passend auszuwÃ¤hlen (fehleranfÃ¤llige Syntax).

* In Perl muss man zwischen Werten und Referenzen auf Werte unterscheiden.
  In Python gibt es nur Referenzen auf Werte.

* In Python ist alles ein Objekt und damit gleichartig behandelbar.
  In Perl ist das nicht so (z.B. elementare Datentypen sind keine Objekte).

Positiv fÃ¼r Perl:

* In Perl ist die Nutzung RegulÃ¤rer AusdrÃ¼cke wesentlich einfacher, da sie
  Teil der Basissprache sind. Die Operatoren "=~" und "!~" und der Datentyp
  "RegulÃ¤rer Ausdruck" /.../ sowie die Operationen "m=match", "s=substitute"
  und "t=translate" gibt es in Python nicht direkt in der Sprache.
  Dort ist eine Bibliothek "re" dafÃ¼r zustÃ¤ndig,
  deren Schnittstelle (etwas) umstÃ¤ndlicher ist als die von Perl.

+-----------------------------------------------+-----------------------------------------------
-----------+-----------------------------------------------------------------------------+
| Feature                                       | Perl
           | Python                                                                      |
+-----------------------------------------------+-----------------------------------------------
-----------+-----------------------------------------------------------------------------+
| Start                                         | 1987
           | 1991                                                                        |
| Creator                                       | Larry Wall
           | Guido van Rossum                                                            |
| Feature                                       | Linguist, computer scientist, System administr
ator       | BVD (Benevolent Dictator)                                                   |
| Motto                                         | TIMTOWTDI
           | Batteries included                                                          |
| Origin of name                                | ? (Pearl --> Perl)
```

	Monty Python	
High level language (abstracting from hardware)	YES	
Scripting language (not compiled to machine code)	YES	
Compiled to byte code interpreted very fast	YES	
Look and feel	Garbage (hard to write/read)	
	Like english text (easy to write/read)	
Free and open source (FOSS)	YES	
Embeddable into other SW (e.g. Abaqus, Apache)	YES	
Embeddable into prog. languages (C, C++, ...)	YES	
Portable	YES	
Object Orientation	No objects internally	
	Fully even internally (everything is an object)	
	Simulated	
Real		
	Added very late (between V4 and V5)	
	Built in from the start	
Extensible (via C, C++, ...)	YES	
Library of ready to use Modules/Packages	YES	
Library location	CPAN (comprehensive perl archive network)	
	PyPI (package index)	
Version	perl -v --version -V	
	python -V	
Extension	*.pl	
	*.py	
Shee-Bang line	#!/usr/bin/perl	
	#!/usr/bin/python	
First usage purpose	One-liners	
	Real programs	
Installed automatically under Linux	YES	
	YES	
Warnings	To be activated manually	
	Active by default	
Help/Documentation	perldoc (external)	
	help("CMD") (internal + external)	
Comments # one-line-long	YES	
	YES	
Statement delimiter ";"	needed always	
	Optional (between two needed)	
Literal constants int, float, string		
Escape character expansion	"..." qq/.../	
	"..." '...'	
No Escape character expansion	'...' q/.../	
	r"..." r'...' b"..." b'...' ??	
Floating point: double = 64 Bit = 15-16 digits	YES	
	YES	
Complex number	---	
	1 + 5j (real + imaginary)	
Multiline string	Here-document <<EOF; "EOF"; 'EOF';	
	Triple quotes """TEXT '''TEXT	
	TEXT TEXT	TEXT TEXT TEXT
		${VAR} ${VAR} ---
	{VAR} ---	
		`CMD` `CMD` ---
	?? ---	
		EOF EOF EOF
	""" '''	
Sigil (Identifier-Prefix)	$ @ % & * (--> 5 different name spaces)	
	NO (--> only 1 name space)	
String concatenation	. _	
	+	
String formatting	printf("%s %d %f", $v1, $v2, $v3)	
	print("%s %d %f" % (v1,v2,v3)) PY2!	

	print("{0} {1} {2}".format(v1,v2,v3))	PY3!
	'{name} wrote {book}'.format(name='Swaroop', book='A Byte of Python')	PY3!
Variable embedding		"this ${name} has ${cnt} values"
	"this {n} has {c} values".format(n=name, c=cnt)	
Command substitution		`CMD ...` qx/.../
	??	
Value conversion		automatically in the background
	explicitly: int("123") --> 123 str(123) --> "123"	
		(hopefully correct)
	(deliberately correct)	
Variable name form		[A-Za-z_][A-Za-z0-9_]* (no Sigil)
	Sigil $@%&* + [A-Za-z_][A-Za-z0-9_]*	
Case sensitive (upper/lower case counts)		YES
	YES	
Data Types		scalar (string, int, double, reference, undef)
	Just objects (anything is an object, even numbers and strings)	
		array, hash, function, regex, typeglob
	str, int, float, complex, list, tuple, dict, ...	
Variable type undeclared (derived from data)		
Block structure		nesting of {...}
	Indenting (space + tab)	
		(indentation doesn't matter, but should be correct)
Operators (same but...)		
Comparison		two sets: numeric + string
	Just one set (for all object types!)	
//		nearly the same as ||
	Division without rest (PY3)	
Logical		&& || ! and or not xor
	and or not	
Concatenation		.
	+	
String replication		x
	*	
Check if element member of a list/array		---
	in, not in	
Check if element of certain type/class/identity		ref --
	is, is not, type, instanceof	
Boolean Type		NO (everything has value true=1/false="")
	YES (True = 1, False = 0)	
		5 false values 0/0.0 ""/'' "0" () undef, REST true
	10 false values None False 0 0.0 0.0j "" '' () [] {}, REST true	
Shortcut arithmetic operator and assignment		YES
	YES	
Operator precedence		perldoc perlop
	http://docs.python.org/3/reference/expressions.html#summary	
Precedence top --> bottom		high to low
	low to high	
		(lot of differences, but main parts have same precedence, e.g. arithmetic operators + - * / % **)
Operator associativity		perldoc perlop
	?? (not documented)	
while/for loop		no else part available
	else part possible	
		last
	break	
		next
	continue	
		redo
	---	
switch...case...default		given when default
	NO	
Range operator		1..9
	range(1,10)	
Different step than 1		--- (not possible)
	range(1,10,5)	
Range countable up+down,positive+negative		NO
	YES	
Operator "," in print		produces no space
	produces 1 space	
Function		sub NAME { ... }
	def NAME()	
		no () behind function name

() behind function name	
no argument names (to do manually if needed)	arguments named automatically
no check of argument number	number of arguments checked
YES (by @_)	Optional parameters
YES	
YES (by @_)	Any number of parameters
YES	
YES (by %param = @_)	Named parameters
YES	
impossible	Local variables via argument list
YES	
automatically if not pushed out by "my"	Transfer global variable into function
Manually by tearing it in via "global"	
POD (plain old documentation) + perldoc	Documentation
docstring + pydoc (or help)	
use MODULE;	Module loading
import MODULE	
*.pm	Module file extension
*.py	
??	Byte code extension
.pyc/.pyo (same directory or __pycache__	
PERL5LIB $ENV{PERL5LIB} %INC @INC	Module loading
PYTHONPATH sys.path dir()	

Feature	
Perl	Python

TODO:
* Plural --> Singular
* Autovivication in Perl aber nicht in Python
* Dynamisches Array in Perl, aber nicht in Python (Indexzugriff ausserhalb)