PowerShell vs Python Reference

PowerShell Python

September 16, 2020

Below is a reference between PowerShell and Python language syntax. Most of these examples where adapted from W3 schools Python tutorials. Is there something we have wrong or is missing? Please contact us.

Syntax

Arrays

	PowerShell	Python
Defining	`@('Hello', 'World')`	`['Hello', 'World']`
Access Element	`$arr = @('Hello', 'World')` `$arr[0]` `# Hello`	`arr = ['Hello', 'World']` `arr[0]` `# 'Hello'`
Length	`$arr = @('Hello', 'World')` `$arr.Length`	`arr = ['Hello', 'World']` `len(arr)`
Adding	`$arr = @('Hello', 'World')` `$arr += "Dude"`	`arr = ['Hello', 'World']` `arr.append('Dude')`
Removing	`$arr = [System.Collections.ArrayList]@('Hello', 'World')` `$arr.RemoveAt($arr.Count - 1)`	`arr = ['Hello', 'World']` `arr.pop(1)`
Removing by value	`$arr = [System.Collections.ArrayList]@('Hello', 'World')` `$arr.Remove("Hello")`	`arr = ['Hello', 'World']` `arr.remove('Hello')`

Casting

	PowerShell	Python
Integers	`$i = [int]"10"`	`i = int("10")`
Floats	`$i = [float]"10.5"`	`i = float("10.5")`
Strings	`$i = [string]10`	`i = str(10)`

Classes

	PowerShell	Python
Definition	`class MyClass {` `    $x = 5` `}`	`class MyClass:` `  x = 5`
Create Object	`[MyClass]::new()`	`MyClass()`
Constructor	`class Person {` `    Person($Name, $Age) {` `        $this.Name = $Name` `        $this.Age = $Age` `    }` `    $Name = ''` `    $Age = 0` `}` `[Person]::new('John', 36)`	`class Person:` `  def __init__(self, name, age):` `    self.name = name` `    self.age = age` `p1 = Person("John", 36)`
Methods	`class Person {` `    Person($Name, $Age) {` `        $this.Name = $Name` `        $this.Age = $Age` `    }` `    [string]myfunc() {` `        return "Hello my name is $($this.Name)"` `    }` `    $Name = ''` `    $Age = 0` `}` `[Person]::new('John', 36)`	`class Person:` `  def __init__(self, name, age):` `    self.name = name` `    self.age = age` `  def myfunc(self):` `    print("Hello my name is " + self.name)` `p1 = Person("John", 36)` `p1.myfunc()`

Conditions

	PowerShell	Python
If \ Else	`$a = 33` `$b = 200` `if ($b -gt $a)` `{` `    Write-Host "b is greater than a"` `}` `elseif ($a -eq $b)` `{` `    Write-Host "a and b are equal"` `}` `else` `{` `  Write-Host "a is greater than b"` `}`	`a = 33` `b = 200` `if b > a:` `  print("b is greater than a")` `elif a == b:` `  print("a and b are equal")` `else:` `  print("a is greater than b")`

Comments

	PowerShell	Python
Single line	`# Hello, world!`	`# Hello, world!`
Multiline	`<#` `Hello, world!` `#>`	`"""` `Hello, world!` `"""`

Data Types

	PowerShell	Python
Get Type	`$var = 1` `$var | Get-Member` `#or` `$var.GetType()`	`var = 1` `type(var)`

Dictionaries

PowerShell | Python

Defining

```powershell
$thisdict = @{
  brand = "Ford"
  model = "Mustang"
  year = 1964
}
```

```python
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Accessing Elements

```powershell
$thisdict = @{
  brand = "Ford"
  model = "Mustang"
  year = 1964
}
$thisdict.brand
$thisdict['brand']
```

```python
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict['brand']
```

Updating Elements

```powershell
$thisdict = @{
  brand = "Ford"
  model = "Mustang"
  year = 1964
}
$thisdict.brand = 'Chevy'
```

```python
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict['brand'] = 'Chevy'
```

Enumerating Keys

```powershell
$thisdict = @{
  brand = "Ford"
  model = "Mustang"
  year = 1964
}
$thisdict.Keys | ForEach-Object {
    $_
}
```

```python
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
    print(x)
```

Enumerating Values

```powershell
$thisdict = @{
  brand = "Ford"
  model = "Mustang"
  year = 1964
}
$thisdict.Values | ForEach-Object {
    $_
}
```

```python
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.values():
    print(x)
```

Check if key exists

```powershell
$thisdict = @{
  brand = "Ford"
  model = "Mustang"
  year = 1964
}
if ($thisdict.ContainsKey("model"))
{
    Write-Host "Yes, 'model' is one of the keys in the thisdict dictionary"
}
```

```python
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

Adding items

```powershell
$thisdict = @{
  brand = "Ford"
  model = "Mustang"
  year = 1964
}
$thisdict.color = 'red'
```

```python
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
```

Functions

PowerShell | Python

Definition

```powershell
function my-function()
{
  Write-Host "Hello from a function"
}
my-function
```

```python
def my_function():
  print("Hello from a function")

my_function()
```

Arguments

```powershell
function my-function($fname, $lname)
{
    Write-Host "$fname $lname"
}


my-function -fname "Adam" -lname "Driscoll"
```

```python
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Adam", "Driscoll")
```

Variable Arguments

```powershell
function my-function()
{
    Write-Host "$($args[2])"
}

my-function "Bill" "Ted" "adam"
```

```python
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

Named Arguments

```powershell
function my-function($child3, $child2, $child1)
{
    Write-Host "The youngest child is $child3"
}

my-function -child1 "Emil" -child2 "Tobias" -child3 "Linus"
```

```python
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

Default Values

```powershell
function my-function
{
    param(
        $country = "Norway"
    )

    Write-Host "I am from $country"
}
```

```python
def my_function(country = "Norway"):
  print("I am from " + country)
```

Return Values

```powershell
function my-function($x)
{
    5 * $x
}
```

```python
def my_function(x):
  return 5 * x
```

Lambdas

PowerShell | Python

Lambda

```powershell
$x = { param($a) $a + 10 }
& $x 5
```

```python
x = lambda a : a + 10
print(x(5))
```

Loops

PowerShell | Python

For

```powershell
$fruits = @("apple", "banana", "cherry")
foreach($x in $fruits)
{
    Write-Host $x
```

```python
fruits = ["apple", "banana", "cherry"]
for x in fruits:
```

	`}`	`print(x)`

While	`$i = 1`
	`while ($i -lt 6)`
	`{`
	`    Write-Host $i`
	`    $i++`
	`}`
		`i = 1`
		`while i < 6:`
		`  print(i)`
		`  i += 1`

Break	`$i = 1`
	`while ($i -lt 6)`
	`{`
	`    Write-Host $i`
	`    if ($i -eq 3)`
	`    {`
	`        break`
	`    }`
	`    $i++`
	`}`
		`i = 1`
		`while i < 6:`
		`   print(i)`
		`   if i == 3:`
		`     break`
		`  i += 1`

Continue	`$i = 1`
	`while ($i -lt 6)`
	`{`
	`    Write-Host $i`
	`    if ($i -eq 3)`
	`    {`
	`        continue`
	`    }`
	`    $i++`
	`}`
		`i = 1`
		`while i < 6:`
		`   print(i)`
		`   if i == 3:`
		`     continue`
		`  i += 1`

Operators

	PowerShell	Python
Addition	`$var = 1 + 1`	`var = 1 + 1`
Subtraction	`$var = 1 - 1`	`var = 1 - 1`
Multiplication	`$var = 1 * 1`	`var = 1 * 1`
Division	`$var = 1 / 1`	`var = 1 / 1`
Modulus	`$var = 1 % 1`	`var = 1 % 1`
Floor	`[Math]::Floor(10 / 3)`	`10 // 3`
Exponent	`[Math]::Pow(10, 3)`	`10 ** 3`

Packages

	PowerShell	Python
Install	`Install-Module PowerShellProtect`	`pip install camelcase`
Import	`Import-Module PowerShellProtect`	`import camelcase`
List	`Get-Module -ListAvailable`	`pip list`

Strings

	PowerShell	Python
String	`"Hello"`	`"Hello"` `'Hello'`

Multiline	`"Hello`
	`World`
	`"`
		`"""Hello`
		`World"""`

Select Character	`$str = 'Hello'`
	`$str[0]`
	`# H`
		`str = 'Hello'`
		`str[0]`
		`# 'H'`

Length	`$str = 'Hello'`
	`$str.Length`
		`str = 'Hello'`
		`len(str)`

Remove whitespace at front and back	`$str = ' Hello '`
	`$str.Trim()`
	`# Hello`
		`str = ' Hello '`
		`str.strip()`
		`# 'Hello'`

To Lowercase	`$str = 'HELLO'`
	`$str.ToLower()`
	`# hello`
		`str = 'HELLO'`
		`str.lower()`
		`# 'hello'`

To Uppercase	`$str = 'hello'`
	`$str.ToUpper()`
	`# HELLO`
		`str = 'hello'`
		`str.upper()`
		`# 'HELLO'`

Replace	`$str = 'Hello'`
	`$str.Replace('H', 'Y')`
	`# Yello`
		`str = 'Hello'`
		`str.replace('H', 'Y')`
		`# 'Yello'`

Split	`'Hello, World' -split ','`
	`# @('Hello', ' World')`
		`str = 'Hello, World'`
		`str.split(',')`
		`# ['Hello', ' World']`

Join	`$array = @("Hello", "World")`
	`$array -join ", "`
	`[String]::Join(', ', $array)`
		`list = ["Hello", "World"]`
		`", ".join(list)`

Formatting	`$price = 49`
	`$txt = "The price is {0} dollars"`
	`$txt -f $price`
		`price = 49`
		`txt = "The price is {} dollars"`
		`print(txt.format(price))`

Formatting by Index	`$price = 49`
	`$txt = "The price is {0} dollars"`
	`$txt -f $price`
		`price = 49`
		`txt = "The price is {0} dollars"`
		`print(txt.format(price))`

Formatting Strings	`$price = 49`
	`"The price is $price dollars"`
		`price = 49`
		`f"The price is {price} dollars"`

Try \ Catch

	PowerShell	Python

	`try {`
	`    Write-Host $x`
	`} catch {`
	`    Write-Host "An exception ocurred"`
	`}`
		`try:`
		`  print(x)`
		`except:`
		`  print("An exception occurred")`

Variables

	PowerShell	Python
	`$var = "Hello"`	`var = "Hello"`

Global	`$global:var = "Hello"`	`global var`
		`var = "Hello"`

Subscribe

Keep up to date with Ironman Software

Email Address *

Subscribe