Rafał Korzeniewski

Wstęp do języka Python (część 2)

W artykule poznamy nowe narzędzie – IPython, który jest ulepszoną powłoką interaktywną Pythona. Poznamy też kilka nowych, ważnych dla języka terminów, takich jak „obiekt”, „zmienna” czy „referencja”. Dowiemy się też, w jaki sposób korzystać z wbudowanej w Pythona dokumentacji.

IPYTHON

W artykule „Wstęp do języka Python” z pierwszego numeru *Juniora* poznaliśmy zarys historii języków programowania. Dowiedzieliśmy się też o pewnych cechach charakteryzujących języki programowania i zobaczyliśmy, gdzie w tym bogatym świecie możemy umieścić Pythona. Z kolei w artykule „Jak zainstalować i używać Pythona” poznaliśmy sposoby na to, w jaki sposób można napisać skrypt w Pythonie i go uruchomić. Wspominałem tam o REPL – interaktywnym środowisku Pythona. Ten skrót wziął się od angielskich słów *read-eval-print-loop*, czyli *czytaj-wykonuj-drukuj-zapętl*. Narzędzie to jest takim prostym, interaktywnym (czyli reagującym na nasze działania) środowiskiem programistycznym. Jest szczególnie przydatne i wygodne wtedy, kiedy chcemy poeksperymentować z krótkimi fragmentami kodu. Większość języków interpretowanych ma takiego własnego REPLa. Pojawiają się też dodatki, które go wzbogacają o nowe funkcjonalności i poprawiają komfort korzystania z takich narzędzi. My też skorzystamy z takiego narzędzia. W świecie Pythona dużą i zasłużoną sławę zyskał sobie IPython.

CIEKAWOSTKA

Zarówno w nauce, jak i w programowaniu czasem możemy znaleźć magię. W matematyce, fizyce czy informatyce mogą to być np. magiczne liczby. W IPythonie mamy natomiast magiczne polecenia. Są to przydatne programy, które znacznie ułatwiają niektóre zadania. Wywołuje się je poprzez specjalną notację. Na przykład polecenie `%lsmagic` wpisane w IPython zwróci nam listę wszystkich magicznych poleceń. Dodaj do nich „?”, a dowiesz się, jaką magię oferują.

Dziś zainstalujemy go i zaczniemy się z nim oswajać. By to zrobić, będziemy musieli skorzystać z jednego z narzędzi umożliwiających wydawanie komputerowi poleceń tekstowych. Na Mac czy Linux może być to program `terminal`. W przypadku Windows może to być program `Command Prompt` albo `Windows Power Shell`. Te programy powinno dać się znaleźć w zainstalowanych aplikacjach. Uruchom więc teraz jeden z nich i wpisz następującą komendę:

```
pip install ipython
```

Jeśli na twoim systemie Python jest poprawnie zainstalowany, to w oknie terminala powinno pojawić się coś takiego jak w Listingu 1. Trzy kropki oznaczają, że jakiś fragment został pominięty:

Listing 1. Instalacja IPythona przy użyciu polecenia narzędzia pip

```
$ pip install ipython
Collecting ipython
  Downloading ...
...
Installing collected packages: ipython
Successfully installed ipython-7.8.0
```

Pip to taki specjalny program, instalowany razem z Pythonem, który potrafi zainstalować różne zewnętrzne moduły pythonowe (opowiemy sobie o nim więcej przy innej okazji). Po takiej operacji IPython może zostać uruchomiony:

```
$ ipython
Python 3.7.2 (default, May  8 2019, 11:42:43)
Type 'copyright', 'credits' or 'license' for more
```

```
information
IPython 7.8.0 -- An enhanced Interactive Python. Type
'?' for help.

In [1]:
```

Jest trochę podobnie jak po uruchomieniu zwykłego REPL. Tutaj zamiast znaków zachęty w postaci >>> otrzymujemy ponumerowane wejścia (`In`) i, jak się potem przekonamy, także wyjścia (`Out`). Dodatkowo mogą być one kolorowane.

RATUNKU! POMOCY!

Często pisząc programy, ma się ochotę krzyknąć: „no niech ktoś mi pomoże!". Dlaczego to nie działa? Jak tego używać? W takich sytuacjach dobrze jest umieć skorzystać z pomocy, jaką oferuje nam sam Python w postaci dokumentacji swoich funkcji.

Dokumentacja to jest taki tekst, przygotowany przez twórcę programu, który informuje innych programistów o tym, w jaki sposób mogą skorzystać z jego programu. Między innymi to właśnie jest w programowaniu takie fajne – nie trzeba za każdym razem pisać wszystkiego od nowa. My też dowiemy się w przyszłości, jak taką dokumentację napisać. Na razie jednak nauczymy się z niej korzystać. W Pythonie służy do tego specjalna funkcja `help`, do której przekazujemy jako argument nazwę funkcji, której dokumentację chcemy przeczytać:

Listing 2. Użycie funkcji help

```
In [1]: help(print)

Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.
stdout, flush=False)

    Prints the values to a stream, or to sys.stdout
by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to
the current sys.stdout.
    sep:   string inserted between values, default a
space.
    end:   string appended after the last value,
default a newline.
    flush: whether to forcibly flush the stream.
```

W IPython jest to jeszcze prostsze. Wystarczy, że po nazwie funkcji wpisany zostanie znak `?`:

Listing 3. Wywołanie dokumentacji funkcji print przy użyciu sposobu wbudowanego w IPython

```
In [2]: print?
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout,
flush=False)

Prints the values to a stream, or to sys.stdout by
default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the
current sys.stdout.
sep:   string inserted between values, default a
space.
end:   string appended after the last value, default
a newline.
flush: whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

Śmiało – przekonaj się o tym. Czytając ten artykuł, powracaj do interpretera w każdej chwili, w której zechcesz sprawdzić jakiś kod. Nie bój się też eksperymentować na własną rękę. Próbuj wykonywać różne operacje. Staraj się przewidzieć wynik i sprawdź, czy rzeczywiście jest tak jak myślałeś/myślałaś. To bardzo fajne ćwiczenia. Jest spora szansa, że w wyniku tej zabawy dostaniesz różne błędy. Błędy i wyjątki rzucane w Pythonie to kolejna porcja informacji o tym, co zrobiliśmy źle. Są bardzo pomocne. W tym momencie trzeba powiedzieć sobie jednak jasno, że jeśli myślisz poważnie o byciu programistą lub programistką, to musisz nauczyć się też angielskiego. Przynajmniej na tyle, by bez problemu czytać i rozumieć teksty w tym języku. Zarówno błędy, jak i dokumentacja pisane są na ogół w języku Szekspira. Same instrukcje języka czy wbudowane funkcje też są oparte o angielskie słowa. Dodatkowo większość programistów pisząc program, stara się także używać angielskich nazw. Przydatna jest też umiejętność zadawania pytań w tym języku. O właśnie – zadawanie pytań i poszukiwanie odpowiedzi na nie to jedna z najczęstszych czynności programisty – szczególnie tego początkującego. Jeśli nie znasz angielskiego na odpowiednim poziomie, to nic straconego. Na początku możesz użyć ogólnodostęp-

nych narzędzi do tłumaczeń. Z czasem na pewno nabierzesz biegłości. Gdy masz więc pytania i wątpliwości, to śmiało szukaj na nie odpowiedzi. Po pierwsze, znaleźć je możesz w oficjalnej dokumentacji Pythona (https://docs.python.org/3/). Choć trzeba przyznać, że to często dość wymagające źródło. Dużo pytań i wyjaśnień znajdziesz na platformie https://stackoverflow.com/ – to jedno z najważniejszych miejsc w sieci, w którym programiści rozmawiają o swoich problemach. Jest też trochę miejsc, w których pomoc możesz znaleźć w języku polskim. Np. na Facebooku dostępne są grupy dla początkujących i zaawansowanych Pythonistów. W Internecie jest też forum https://pl.python.org/forum/. Wielu doświadczonych Pythonistów znaleźć można także na kanale IRC. Ten ostatni to taka stara usługa, rodzaj czatu, bardzo popularnego w latach 90. ubiegłego wieku. Zanim jednak zaczniesz bombardować starszych kolegów pytaniami, postaraj się znaleźć odpowiedź na własną rękę – choćby korzystając z google. Dopiero kiedy spróbujesz kilka razy i nie będziesz sobie radzić, poproś o pomoc kogoś bardziej doświadczonego. To takie dobre zasady. Ale dość tych porad. Wracajmy do Pythona!

WARTOŚĆ, ZMIENNA, REFERENCJA, OBIEKT

Jednym z pierwszych programów, jakie piszemy w jakimkolwiek języku, jest wypisanie powitania. W Pythonie może to wyglądać tak jak w Listingu 4.

Listing 4. Wartość i przypisanie wartości do zmiennej w Pythonie

```
In [1]: x = "Programista Junior"
In [2]: x
Out[2]: 'Programista Junior'
In [3]: print(x)
Programista Junior
In [4]: y = print(x)
Programista Junior
In [5]: y
In [6]:
In [7]: print(y)
None
```

Co tu się dzieje? W `In [1]` do nazwy `x` przypisujemy wartość `"Programista Junior"`. Od teraz będziemy ten `x` w zależności od kontekstu nazywać zmienną (ang. *variable*), nazwą (ang. *name*) albo referencją (ang. *reference*). Referencja to taka wartość, która zawiera informację o położeniu innej wartości. U nas ta referencja ma nazwę `x` i wskazuje na napis `"Programista Junior"`. W `In [2]` po prostu odwołujemy się do zmiennej `x`. Jak widzimy, tym razem pojawia się sekcja `Out[2]`. To oznacza, że poprzednie działanie zwraca jakąś wartość. W tym przypadku jest to po prostu napis. W `In [2]` wywołujemy funkcję `print`. Zauważmy, że tym razem nie ma sekcji `Out`. Oznacza to, że funkcja `print` niczego nie zwraca, choć, jak widzimy poniżej, pojawił się napis. By być precyzyjnym, powinienem jednak napisać, że funkcja `print` zwraca specjalny obiekt, który nazywa się `None`. To słowo oznacza po angielsku: żaden, nikt, nic. Możemy więc napisać, że takie funkcje zwracają nic. A czasem – dość przewrotnie – po polsku mówimy, że one niczego nie zwracają. W Pythonie takie „nic" opisuje się właśnie słówkiem `None`. Kryje się za nim specjalny obiekt, osobny typ, który bywa przydatny. Z czasem zobaczymy, w jakich sytuacjach. Zastanów się nad tym chwilę i poeksperymentuj. Wróćmy jeszcze do naszego kodu. Samo przypisanie wartości do nazwy odbywa się przy użyciu znaku =. Co to jest jednak to przypisanie? Wyobraź sobie szafkę, która ma wiele szufladek. Obok niej biurko, a na nim notatnik. Do szuflad będziemy chować różne rzeczy, a w notatniku na biurku będziemy notować, co gdzie włożyliśmy. W jedną z szuflad wsadzamy kartkę z napisem `"Programista Junior"`. Szuflady są ponumerowane. Teraz w notatniku notujemy, że pod nazwą `x` od teraz będzie kryła się zawartość tej szuflady, w której kryje się ta kartka z napisem. Python robi za nas coś podobnego. Możemy w nim nawet sprawdzić numer tej szuflady, w taki sposób jak w Listingu 5:

Listing 5. Wywołanie funkcji id zwraca nam adres obiektu

```
In [4]: id(x)

Out[4]: 140087084787512
```

Jak widać, została tu zwrócona jakaś duża liczba. Możemy ją potraktować właśnie jako numer naszej szuflady – czyli adres miejsca w pamięci. W szufladzie, jak pamiętamy, znajduje się kartka z napisem. W Pythonie moglibyśmy powiedzieć, że ta kartka jest obiektem (ang. *ob-*

ject), a wartością jest sam napis. I teraz za każdym razem, kiedy gdzieś w programie użyjemy nazwy x, to Python otwiera odpowiednią szufladę i przekazuje w to miejsce tę kartkę. Oczywiście to duże uproszczenie. W twojej głowie mogą teraz powstać pytania o to, co się dalej dzieje z tą kartką, a co jest w szufladzie, a co, jeśli wiele razy obok siebie użyjemy tego x? Czy mogę do szuflady włożyć wiele kartek naraz? Świetnie, jeśli tak jest. O takie pytania nam chodzi. Będziemy małymi krokami szukać na nie odpowiedzi. Wróćmy do tej nowej funkcji. Zobaczmy, co znajdziemy w jej dokumentacji:

Listing 6. Dokumentacja funkcji id

```
In [1]: id?
Signature: id(obj, /)
Docstring:
Return the identity of an object.

This is guaranteed to be unique among simultaneously
existing objects.
(CPython uses the object's memory address.)
Type:      builtin_function_or_method
```

Pojawia się tu pojęcie `object`. Termin ten bywa różnie definiowany w różnych językach programowania. Na nasze potrzeby możemy przyjąć, że w Pythonie obiektem jest wszystko to, co można przypisać do zmiennej. Jak się przekonamy później, obiekty mogą mieć atrybuty i metody. Metody to różnego rodzaju działania. Atrybuty to z kolei różne cechy obiektu. Obiektem może być np. `pies`. Jego atrybutem może być `waga`, a metodą np. polecenie: `daj_głos`. Więcej opowiemy sobie o tym w artykule poświęconym programowaniu obiektowemu, już teraz jednak przyda się nam pewna elementarna wiedza o tych zagadnieniach, tym bardziej że w Pythonie w zasadzie wszystko jest obiektem. W naszym wcześniejszym przykładzie to kartka jest obiektem. Z kolei napis na niej to wartość tego obiektu. Jak widzieliśmy, nasz obiekt powstawał w momencie przypisania do zmiennej, ale, co ciekawe, może on istnieć także bez tej nazwy, bez przypisania do zmiennej – bez referencji do niego. Możemy stworzyć kartkę z napisem i nie odnotować jej w naszym notatniku. Referencja nie powstała, ale kartka ma się dobrze. Takich kartek jednak nie chcemy przechowywać zbyt długo. Gdyby było ich bardzo dużo, mogłyby utrudnić nam pracę. Warto więc co jakiś czas je posprzątać. Jak by to wyglądało w Pythonie? Podajmy w funkcji `id` wartość i zobaczmy, że także dla niej zostanie zwrócony adres obiektu.

Listing 7. Wywołanie funkcji id dla napisu, który nie jest przypisany do nazwy, także zwraca adres obiektu

```
In [5]: id("Programista Junior")
Out[5]: 140247295987904
```

Zauważ, że tym razem jest to już nieco inna liczba, mimo że napis jest ten sam. Nie zawsze jednak ten adres będzie się różnił. Jako ciekawostkę potraktuj fakt, że jeśli nieznacznie zmodyfikujemy nasz napis, to otrzymamy nieco inne zachowanie. Zobacz:

Listing 8. Czasem obiekty są umieszczane w cache

```
In [13]: x = "Programista_Junior"
In [14]: y = "Programista_Junior"
In [15]: id(x)
Out[15]: 140247296531384
In [16]: id("Programista_Junior")
Out[16]: 140247296531384
```

Tym razem adresy naszych obiektów jest takie same. Python w celach optymalizacyjnych w niektórych przypadkach odkłada takie obiekty w pamięci podręcznej – jest to tzw. *cache*. Pozwala to oszczędzać pamięć. Zamiast przechowywać wiele identycznych kartek w różnych szufladach – trzymamy jedną i tam, gdzie trzeba, przekazujemy tylko jej adres. Dzięki temu pozostałe są wolne. W przypadku napisów stanie się tak wtedy, gdy napis składa się z samych znaków alfanumerycznych (litery i cyfry) i znaków podkreślenia. Nie jest to jednak mechanizm, na którym powinniśmy polegać, ponieważ te metody optymalizacji mogą się zmieniać wraz z nowymi wydaniami Pythona. Potraktuj to więc jako ciekawostkę. Wiąże się z tym jednak pewne zagadnienie, które może być dla nas ważne w przyszłości.

Kiedy w Pythonie chcemy porównać dwie wartości, używamy operatora porównania ==. O samych operatorach powiemy sobie jeszcze trochę później. Na razie wystarczy, że zapamiętasz, że gdy chcesz porównać ze sobą jakieś dwie wartości, sprawdzić, czy są to np. takie

same liczby czy napisy, to używasz właśnie tego podwójnego znaku ==. Działa to tak jak w Listingu 9.

Listing 9. Porównanie dwóch zmiennych

```
In [17]: x == y
Out[17]: True
```

Słowo `True` poznaliśmy już w poprzednim artykule. Oznacza ono prawdę. Jego przeciwieństwem jest słówko `False`. W tym przypadku nasze porównanie powiedziało, że to, co jest po lewej i prawej stronie operatora ==, jest takie samo. Tak naprawdę chodzi tu o to, że pewne wartości są tu takie same. Jeśli naszym obiektem są liczby, to porównywać będziemy ze sobą wartości tych liczb. Jeśli napisy, to sprawdzimy wtedy, czy te napisy są identyczne. Ale ten operator nie odpowiada na pytanie, czy są to te same obiekty. Operator == nie sprawdza więc tego, czy odwołujemy się do tej samej szuflady – a wiemy już, że czasem wartości mogą leżeć w różnych szufladach naszej pamięci, mimo że wydają się identyczne. Kiedy chcemy sprawdzić, czy nasze zmienne (referencje) wskazują na te same obiekty, używamy innego operatora – jest nim słowo `is`.

```
In [18]: x is y
Out[18]: True
```

Zwrócona wartość `True` oznacza, że `x` i `y` wskazują na ten sam obiekt. Czytając dosłownie, moglibyśmy powiedzieć, że x jest y-kiem. Ale chodzi tu o to, że to, na co wskazuje `x`, jest tym samym, na co wskazuje `y`. Może się zdarzyć też inna sytuacja. Wartości będą takie same, ale `x` i `y` będą wskazywać na dwa różne obiekty – czyli na dwie różne szuflady. Wróćmy do przykładu, w którym w napisie użyliśmy spacji (Listing 10):

Listing 10. Różne obiekty, ale te same wartości

```
In [19]: x = "Programista Junior"
In [20]: y = "Programista Junior"
In [21]: x == y
Out[21]: True
In [22]: x is y
Out[22]: False
```

W Pythonie operator `is` używany jest bardzo często wtedy, kiedy chcemy porównać coś z wartościami logicznymi `True/False/None`. Robimy tak dlatego, że te wartości logiczne są zawsze tymi samymi obiektami. To znaczy, że każdy z nich ma swoją własną szufladę i nie są tworzone nowe szufladki, które by je zawierały. Takie obiekty nazywamy też czasem singletonami. W zapisie chodzi o to, by zamiast pisać: `jakas_wartosc == True` pisać raczej tak: `jakas_wartosc is True`. Czasem może się to okazać istotne. Prześledź uważnie przykład z Listingu 11. Wyjaśnienie tego, co się tam dzieje, znajdziesz w kolejnym artykule.

Listing 11. Przykład różnych wyników zastosowania operatorów == oraz is

```
In [1]: x = True
In [2]: x is True
Out[2]: True
In [3]: x == True
Out[3]: True

In [4]: x = 1
In [5]: x is Truexx
Out[5]: False
In [6]: x == True
Out[6]: True

In [7]: x = 2
In [8]: x is True
Out[8]: False
In [9]: x == True
Out[9]: False
```

Do wartości logicznych wrócimy jeszcze w kolejnym artykule, kiedy będziemy omawiać typy. W tej chwili postaraj się jednak zapamiętać te podstawowe fakty o wartościach, zmiennych, nazwach, obiektach i referencjach. Poeksperymentuj z funkcją `id` dla różnych napisów. Ale też i dla liczb czy wartości logicznych.

Poznaliśmy dziś też funkcję `help` i zobaczyliśmy, jak ją wywołać dla jakiejś innej funkcji. Skoro `help` to też funkcja, to zobacz, co się stanie, jeśli spróbujesz wywołać pomoc dla niej samej? Czasem ta dokumentacja jest długa. By przejść dalej, wystarczy wcisnąć klawisz Enter. Z pomocy można wyjść, wciskając klawisz `q`. Spróbuj poniższych wywołań:

Listing 12. Szczególne przypadki wywołania funkcji help

```
In [1]: help(help)
...
In [2]: help()
...
In [3]: ?
...
```

Python, podobnie jak inne języki programowania, składa się z pewnych słów kluczowych. Kiedy tworzymy jakieś nowe zmienne, musimy pamiętać o tym, że nie mogą się one nazywać tak jak te słowa kluczowe. Warto więc wiedzieć, jakie są to słowa. Możemy to sprawdzić w pomocy:

Listing 13. Rozbudowana pomoc oferowana przez funkcję help()

```
In [5]: help()

Welcome to Python 3.6's help utility!
...

help> keywords

Here is a list of the Python keywords.  Enter any
keyword to get more help.

False       def         if          raise
None        del         import      return
True        elif        in          try
and         else        is          while
as          except      lambda      with
assert      finally     nonlocal    yield
break       for         not
class       from        or
continue    global      pass
```

Jeśli spróbujesz stworzyć zmienną z któregoś z tych słów, to zostanie zwrócony błąd:

Listing 14. Próba użycia słowa kluczowego jako nazwy zmiennej zakończy się niepowodzeniem

```
In [6]: class = 1
  File "<ipython-input-6-0251f39d4826>", line 1
    class = 1
          ^
SyntaxError: invalid syntax
```

Podobnie zakończy się próba wywołania dla nich pomocy: `help(class)` - bezpieczniej będzie użyć tu tego słowa w postaci napisu: `help("class")`. Można też dotrzeć do opisu słów kluczowych właśnie poprzez tę rozbudowaną pomoc dostępną po wywołaniu funkcji `help()`. Postaraj się zrobić to samodzielnie.

Zauważ, że w Listingu 13 nie ma na liście funkcji `print`, `help` czy `id`. Te ostatnie są tak zwanymi funkcjami wbudowanymi. Pełną ich listę (dla Pythona 3.7) znajdziesz w Tabeli 1.

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Tabela 1. Funkcje wbudowane w Pythona. Możesz też o nich przeczytać w dokumentacji Pythona: https://docs.python.org/3/library/functions.html

W przypadku funkcji wbudowanych możliwe jest użycie ich nazw jako nazwy zmiennej. Należy jednak pamiętać o tym, że jeśli to zrobimy, to stracimy możliwość użycia takiej funkcji. Mówimy wtedy, że przesłaniamy daną funkcję czy obiekty. W Pythonie funkcje to też obiekty. A więc ich nazwy to referencje do tych obiektów. Prześledźmy to w Listingu 15.

Listing 15. Przykład przesłonięcia wbudowanej funkcji poprzez zastosowanie złej nazwy dla zmiennej

```
In [1]: print("Działa")
Działa
In [2]: print = 10
In [3]: print("Działa?")
---------------------------------
TypeError  Traceback (most recent call last)
<ipython-input-3-24d45c998241> in <module>
----> 1 print("Działa?")

TypeError: 'int' object is not callable

In [4]: print
Out[4]: 10
```

W `In [2]` powiedzieliśmy Pythonowi, że od teraz nazwa `print` wskazuje na obiekt, który jest liczbą `10`. Zmieniliśmy po prostu szufladki, na które wskazuje `print`. Możemy też zrobić odwrotnie, co może być przyczyną różnych kłopotów. Prześledź to w Listingu 16.

Listing 16. Od teraz x wskazuje na ten sam obiekt, co nazwa print. Może więc być tak samo używany

```
In [1]: x=print
In [2]: x("to działa")
to działa

In [3]: y=10
In [4]: print=y
In [5]: print('czy to zadziala?')
---------------------------------
TypeError  Traceback (most recent call last)
<ipython-input-5-297252317ec0> in <module>
----> 1 print('czy to zadziala?')

TypeError: 'int' object is not callable

In [6]: print = __builtins__.print
In [7]: print("Uff znowu działa")
Uff znowu działa
```

WARTO WIEDZIEĆ

Świat Pythona jest znacznie większy niż mogłoby się wydawać na pierwszy rzut oka. Pierwsza styczność z językiem na ogół dotyczy tzw. standardowej implementacji języka, określanej też jako CPython. Nie jest ona jednak jedyna. Inne popularne implementacje to IronPython, Jython, PyPy. Hmm... ale co to jest implementacja? Postaraj się samemu znaleźć odpowiedź na to pytanie.

Na koniec zapamiętaj jeszcze, że nazwy zmiennych powinny składać się z liter, cyfr i znaku podkreślenia. Nie mogą natomiast zaczynać się od cyfry. Zobacz, co się stanie, gdy spróbujesz obejść tę zasadę.

Gdy skończysz, możesz zamknąć IPythona. By to zrobić, wpisz po prostu `exit()` lub naciśnij jednocześnie Ctrl+d (w przypadku Windows Ctrl + z, a potem Enter lub F6 i potem Enter).

A teraz pora na chwilę przerwy. Oderwij się na trochę od czytania, od komputera. Poświęć trochę czasu na ćwiczenia fizyczne. Daj odpocząć swoim oczom. Jeśli możesz, wybierz się z rodzicami na spacer, a jeśli nie, to popatrz sobie chociaż przez okno w dal. To ćwiczenia akomodacyjne. Poświęć na takie ćwiczenia oczu choć parę minut dziennie. Najlepiej rób to kilka razy w ciągu dnia. Zobaczysz – kiedyś mi za tę radę podziękujesz.

Rafał Korzeniewski

Z wykształcenia muzyk-puzonista i fizyk. Z zamiłowania i zawodu Pythonista. Trener Pythona w ALX, współorganizator PyWaw (http://pywaw.org) – warszawskiego meetupu poświęconego Pythonowi. W wolnych chwilach uczy się gry na nowych instrumentach, udziela się społecznie i dużo czyta.

KORZENIEWSKI@GMAIL.COM