PWSZ TARNÓW 2011

Język Python.™

Wykład PWSZ Tarnów 2011.

Łukasz Cichocki

Plan wykładu :

1. Krótka historia języka Python.
2. Dlaczego Python jest fajny.
3. Instalacja i używanie interpretera.
4. Podstawowe operacje na liczbach i łańcuchach znakowych.
5. Instrukcje warunkowe.
6. Struktury danych.
7. Moduły.
8. Technika Object Oriented Programming.
9. Obsługa wyjątków.
10. Przykłady praktyczne :

- Ingerencja w protokół **http** z użyciem Pythona.

1. Krótka historia języka Python.

Python to interpretowany, interaktywny, skryptowy język programowania stworzony przez Guido van Rossuma w 1990.

Python posiada w pełni dynamiczny system typów i automatyczne zarządzanie pamięcią, jest zatem podobny do takich języków, jak Tcl, Perl, Scheme czy Ruby.

Python realizuje jednocześnie kilka paradygmatów. Podobnie do C++ nie wymusza jednego stylu programowania, pozwalając na stosowanie różnych. Python posiada:

- Paradygmat programowanie obiektowego,
- Paradygmat programowanie strukturalnego
- Paradygmat programowanie funkcyjnego

Typy sprawdzane są dynamicznie, a do zarządzania pamięcią stosuje się architekturę garbage collection podobnie jak w języku Ruby czy też D.

2. Dlaczego Python jest fajny.

Prosta i czytelna składnia:

- Niewielka liczba słów kluczowych i operatorów
- Indentacja - Wymuszenie stosowania wcięć
- Dynamiczny system typów

Batteries included:

- Biblioteki operacji we/wy
- Obsługa wyrażeń regularnych
- HTTP, HTML, XML
- Interfejsy okienkowe (pyGTK, wxPython, Tkinter)

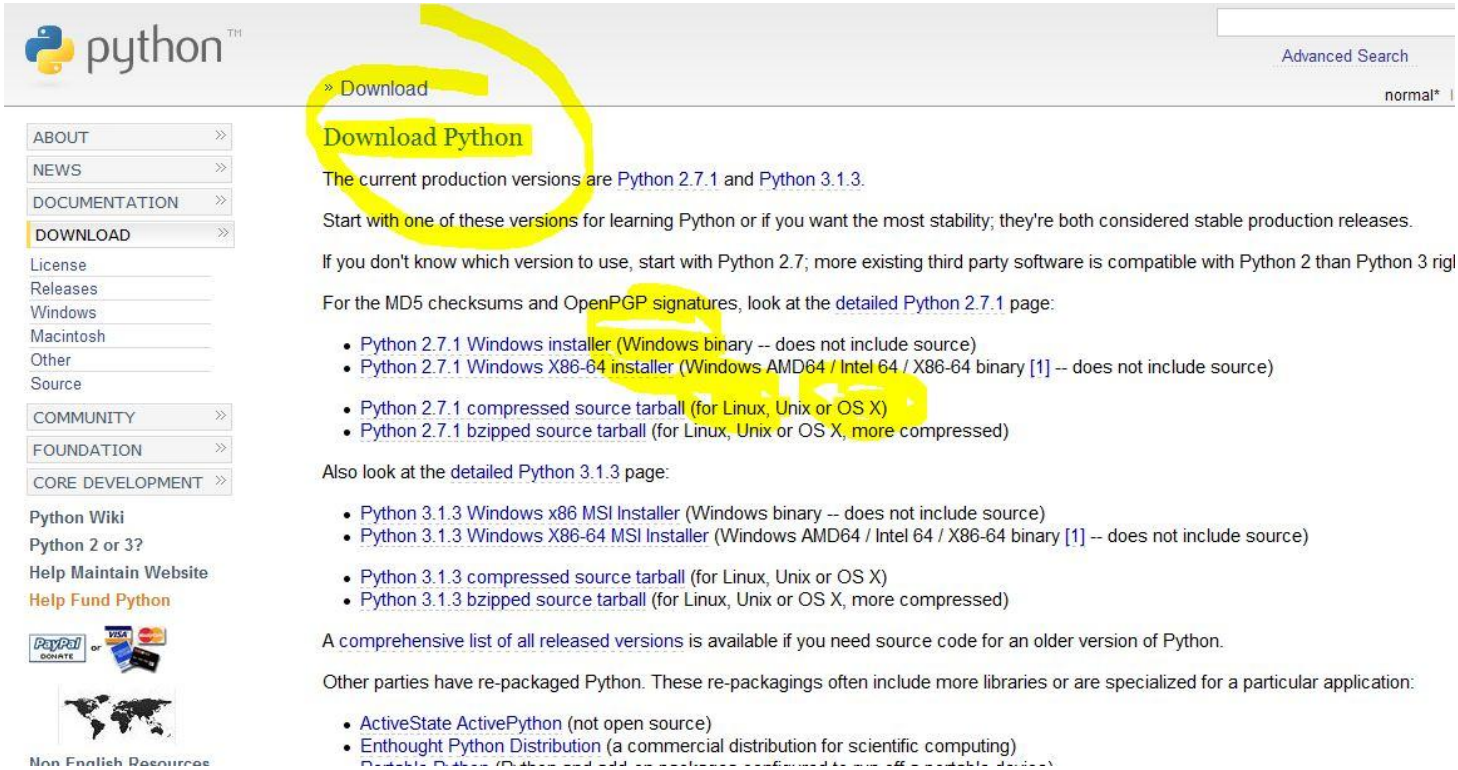
Zastosowania/projekty:

- Narzędzia systemowe (RedHat)
- NASA
- Blender
- MySQL Workbench
- Google

3. Instalacja i używanie interpretera.

Możliwość użytkowania pod wieloma platformami operacyjnymi:

- Linux
- Unix
- Windows (ActivePython.)
- MAC OS X

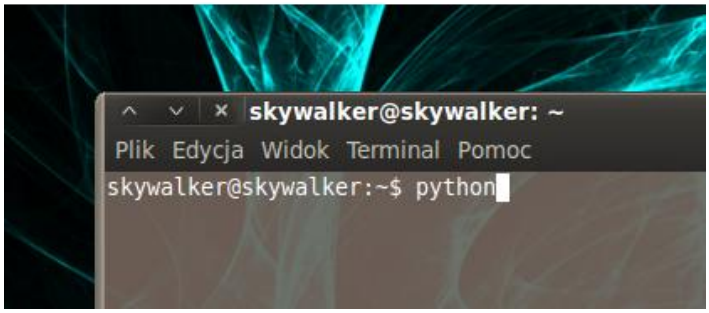


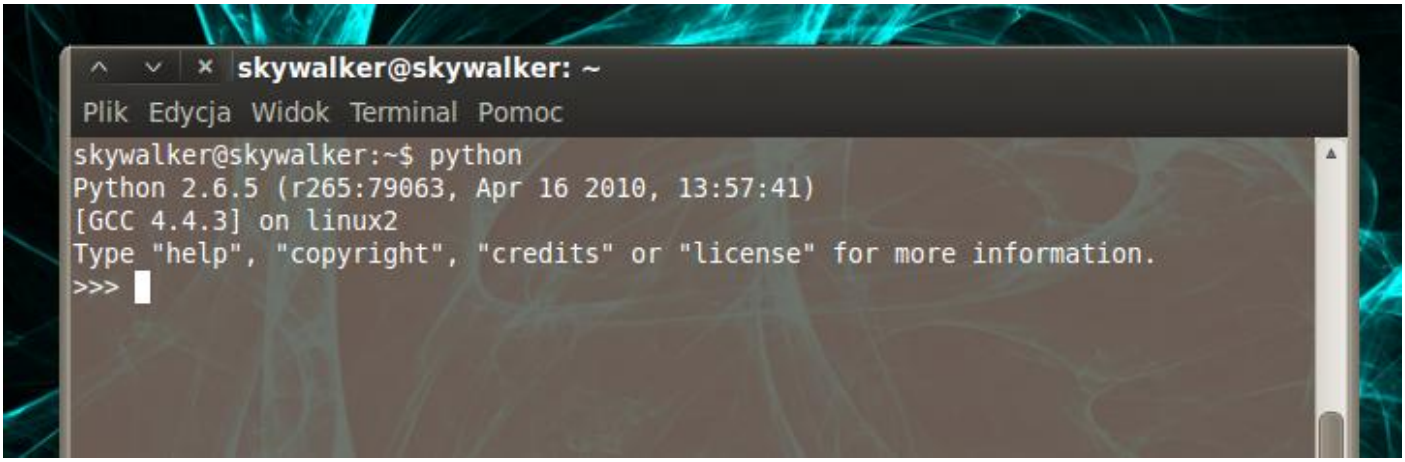
Tryb interaktywny:

- kalkulator
- polecenia w locie

Uruchamiamy poleceniem:

```
$ python
```





Możemy teraz wydawać polecenia, które zostaną wykonane od razu.

```
>>> print "Czesc Swiecie !"
```

Drugim sposobem jest zapisanie treści programu w pliku o rozszerzeniu .py

A następnie odtworzeniu go za pomocą polecenia:

```
>>> python nazwa_pliku
```

Łańcuchy znaków – Stringi.

Łańcuchy możemy przypisywać zmiennym i wyświetlać je za pomocą instrukcji PRINT.

Należy pamiętać o wielkości liter !

Zauważmy też że linie kodu nie są zakańczane terminatorem „ ; ” jak w przypadku innych języków pokroju C, C++.

```
>>> a = ”Witaj w PWSZ”
>>> print a
```

Łączenie łancuchów:

```
>>> a = "Witaj w PWSZ"
>>> b = "w Tarnowie!!!"
>>> print a, b
```

Możliwe użycie funktora „ + ” dla dodania oraz „ * ” dla pomnożenia ilości.

Należy zauważyć że „ , ” pełni rolę automatycznej spacji, bez niego łańcuchy zlepiane są ciągiem.

```
>>> a = "Witaj w PWSZ"
>>> b = "w Tarnowie!!!"
>>> print a, b*3
```

Możemy także w prosty sposób zamienić tekst na liczbę (o ile ma to sens) za pomocą predykatu eval()

```
>>> x = "2"
>>> print x*4
2222
>>> x = eval(x)    # obliczenie wartości
>>> print x*2
4
```

Wyświetlanie łańcuchów:

- ”””” .... ”””” pole tekstowe uznające formatowanie użyte w trakcie wprowadzania
- ”...” – pole tekstowe w obrębie jednej linijki
- ‘...‘– tożsame jw.
- \n – znak nowej lini
- + - łączenie łańcuchów
- * - powielanie ilości
- eval()– wartości z łańcuchów

Do pobrania wartości, którą zechcemy przypisać zmiennej, należy użyć funkcji input() :

```
>>> a = input("podaj a:")
podaj a:4
>>> print a
4
```

SLICES – Plasterki.

Ciekawym elementem Pythona są tzw. plasterki (ang. slices).

Łańcuch znaków zostaje pocięty na " plasterki" , oznaczane nawiasami kwadratowymi.

I tak łańcuch [0] oznacza pierwszy znak łańcucha, łańcuch [1] drugi.

```
>>> s = "tarnów"
>>> print s[0]
t
>>> print s[1]
a
```

Python idzie nieco dalej ogólniejsze

"krojenie" polega na wskazaniu dolnego indeksu i górnego (w ten sposób dostaniemy się do fragmentu od wskazanego dolnego indeksu do górnego, ale bez niego).

```
>>> print s[2:]
rnów

>>> print s[:2]
ta

>>> print s[1:5]
arn

>>> print s[:-1]
tarnó
```

Liczby – opcje kalkulacyjne.

Operacje na funktorach :

- „ + ” dodawanie elementów
- „ * „ mnożenie elementów
- „ / „ dzielenei elementów
- „round()” zaokrąlgenie
- „pow(x,y)” potęgowanie 1 metoda.
- „x ** y” potęgowanie 2 metoda.
- x += nienadmiarowe dodawanie

Liczby zespolone są również wbudowane, części urojone zapisywane są z przyrostkiem „j” lub „J”. Liczby zespolone z niezerową częścią rzeczywistą zapisywane są jako „(*real*+*imag*j)” lub mogą być´ stworzone za pomocą, funkcji „complex(*real*, *imag*)”.

```
>>> 1j * 1J

(-1+0j)

>>> 1j * complex(0,1)

(-1+0j)

>>> 3+1j*3

(3+3j)

>>> (3+1j)*3

(9+3j)

>>> (1+2j)/(1+1j)

(1.5+0.5j)
```

Listy.

Listy przypominają nieco tablice w innych językach. Charakteryzują się umieszczeniem elementów w nawiasach kwadratowych.

```
>>> lista = ["pierwsza", "druga", "trzecia"]
>>> print lista
['pierwsza', 'druga', 'trzecia']
```

Na listach możemy dokonywać standardowych operacji: dodawać nowe elementy, usuwać stare, sortować, wyłapywać fragmenty.

Jeśli wydamy polecenie dir(nazwa typu), otrzymamy listing nazw operacji, które można przeprowadzać na danym typie:

```
>>> dir(lista)
['append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>> lista.append("Dodatek")
>>> print lista
['pierwsza', 'druga', 'trzecia', 'Dodatek']
```

>>> lista.sort() - sortuje listę w kolejności alfabetycznej bądź liczbowej.

Co ciekawe, w Pythonie mamy dostęp do prostej " dokumentacji" używanych funkcji przy pomocy atrybutu .__doc__. Np.:

```
>>> print lista.append.__doc__
L.append(object) -- append object to end
```

Tuple(zbiory, krotki) i slowniki.

Zbiór to typ danych bardzo przypominający listę, wizualnie różni się brakiem nawiasów.

```
>>> kolejka = 'Agata', 'Ania', 'Ula'
>>> print zbior
('Agata', 'Ania', 'Ula')
>>> print zbior[1]
Ania
```

W przeciwieństwie do list, wartości zbiorów są niezmienne.

Słowniki zaś przypominają tablice asocjacyjne: z każdą wartością jest skojarzony jakiś klucz. Kolejność w słowniku nie ma żadnego znaczenia, dlatego zwykłe indeksowanie nie zadziała.

```
>>> slownik = { 'Agata':96, 'Ula':69}
>>> slownik['Agata']
96
```

Funkcje słowników:

- Keys() służy do wyświetlania kluczy
- values() do wyświetlania wartości
- clear() do czyszczenia słownika
- copy() do kopii elementów
- get() do pobrania elementów
- update() do zamiany elementów

Do poszczególnych par możemy się dostać w ten sposób:

```
>>> print slownik.items()[1]
('Ula', 69)
```

Instrukcje warunkowe

if (jeśli)

```
>>> if 2+2==4:
...     print "2+2=4!"
...
2+2=4!
```

Należy zwrócić uwagę na dwukropek kończący warunek oraz wcięcie poprzedzające instrukcję.

elif ("jeśli natomiast...")

```
>>> if 2+2==5:
...     print "Blad- niedrukowany"
... elif 2+2==4:
...     print "Poprawny druk"
```

elif można z powodzeniem zastosować do stworzenia konstrukcji „switch-else” znanej np. z C.

Reszta instrukcji iteracyjnych :

- for ("dla każdego x z przedziału...")
- while ("dopóki")

Dla instrukcji for używa się funkcji zasięgu :

range():

```
>>> for i in range (1,4):
...     print "Numer iteracji: ", i
...
Numer iteracji:  1
Numer iteracji:  2
Numer iteracji:  3
```

Dodatkowe operatory i instrukcje sterujące:

- break
- continue
- pass
- ==
- <=
- !=
- <>

Funkcje.

Tworzenie funkcji w Pythonie jest bardzo proste. Konstrukcje rozpoczynamy od słowa „ def ”, po którym następuje nazwa funkcji z nawiasami, w których opcjonalnie mogą się znajdować argumenty.

```
>>> def dodaj(a,b):
...     return a+b
...
>>> dodaj(1,2)
3
```

Nie musimy martwić się o sposób przekazywania argumentów cały proces odbywa się automatycznie.

Wywołanie funkcji nie zmienia tego obszaru pamięci (on nadal istnieje), tworzony jest jedynie nowy obszar pamięci z innymi danymi, zmienna " lista" na chwilę wskazuje na nowy obszar, po czym wracając z funkcji przywraca oryginalne wskazanie.

```
>>> def wypelnij_liste(lista):
...     lista[0] = "Łukasz"
...
>>> lista = ["ala","ma","kotka"]
>>> wypelnij_liste(lista)
>>> print lista
```

Moduły.

Moduły zawierają wiele dodatkowych funkcji. Wraz z Pythonem dostarczanych jest wiele modułów, oferujących ogromne choć rzadko doceniane możliwości. Ładuje się je przy pomocy polecenia import.

Przykladowe moduły:

String - łańcuchy znaków

```
>>> import string
```

zobaczmy, co w sobie kryje:

```
>>> dir(string)
['__builtins__', '__doc__', '__file__', '__name__',
'_idmap', '_idmapL', '_lower', '_re', '_safe_env',
'_swapcase', '_upper', 'atof', 'atof_error',
'atoi', 'atoi_error', 'atol', 'atol_error',
'capitalize', 'capwords', 'center', 'count',
'digits', 'expandtabs', 'find', 'hexdigits',
'index', 'index_error', 'join', 'joinfields',
'letters', 'ljust', 'lower', 'lowercase', 'lstrip',
'maketrans', 'octdigits', 'replace', 'rfind',
'rindex', 'rjust', 'rstrip', 'split',
'splitfields', 'strip', 'swapcase', 'translate',
'upper', 'uppercase', 'whitespace', 'zfill']
```

Istnieją dwie możliwości: albo zaimportujemy moduł poleceniem import nazwa_modułu, i wtedy musimy wywoływać funkcję w postaci: nazwa_modułu.nazwa_funkcji, albo zaimportujemy je do użycia w sposób przezroczysty, za pomocą polecenia from nazwa_modułu import nazwa_funkcji, np.:

```
>>> from string
```

Możemy też od razu zaimportować wszystkie:

```
>>> from string import *
```

Funkcja : Upper – służy do zamiany rozmiaru wszystkich znaków na duże.

```
>>> a = "Literki Rosną"
>>> print upper(a)
LITERKI ROSNĄ
```

Funkcja: **Capitalize()** – Powiększa tylko pierwsza literę zdania.

```
>>> print capitalize(a)
Literki rosną
```

Funkcja **: Capwords()** – powiększa pierwsze litery każdego słowa.

```
>>> print capwords(a)
Literki Rosną
```

Moduł : re i regex – jest modułem który pozwala na stosowanie wyrażeń regularnych.

Wyrażenia regularne pozwalają na odnalezienie znaków pasujących do podanego wzorca.

```
>>> import re
```

Funkcja: Findall –znajduje wszystkie przypadki wystąpienia znaku.

```
>>> re.findall("a", 'Agata Kowalska')
['a', 'a', 'a', 'a']
```

znak 'A' na początku linii:

```
>>> re.findall("^A", 'Agata Kowalska')
['A']
```

ciąg 'ka' na końcu linii:

```
>>> re.findall("ka$", 'Agata Kowalska')
['ka']
```

Object Oriented Programming.

Klasy i obiekty tworzone są podobnie jak w innych językach wysokiego poziomu.

```
# definiujemy klasę Kwadrat:
class Kwadrat:
  bok = 0
# do tego metodę podającą wartość boku
  def podaj_bok(self):
    return self.bok
```

Na powyższym przykładzie na początku definiujemy klasę kwadrat, w kolejnym kroku utworzona została zmienna bok oraz przypisywana jest jej wartość początkowa, wywoływana standardowo w przypadku nie podania żadnej innej wartości, następnie definiowana jest metoda klasy podaj_bok.

Jak widać technika obiektowa w języku Python nie przyjmuje żadnych innowacji.

Utworzenie instancji, czyli nowego obiektu na kanwie klasy, odbywa się:

```
>>> obiekt = Kwadrat()
```

Dziedziczenie

Odbywa się standardowo, na zasadzie utworzenia nowej instancji o metodach dziedziczonych przez rodzica, jak na poniższym przykładzie, instancja Sześcian, posiadająca wszystkie właściwości klasy Kwadrat.

```
class Szescian(Kwadrat):
```

Wyjątki.

" Wyjątki" to skrót myślowy od " sytuacje wyjątkowe" , a mniej eufemistycznie błąd w wykonywaniu (a nie prekompilowaniu) programu. Wyjątki to mechanizm nie tyle zapobieganiu groźnym sytuacjom, ale sposób ich zgrabnego opanowania bez palpitacji serca u użytkownika.

```
try:
  ...  # kod programu
except:
  ...  # kod wykonywany w przypadku
```

Python wykonuje kod zawarty między klauzulami "try-except" i jeśli zdarzy się tu błąd wykonywanie przeskoczy od razu do bloku " except" . Dlaczego jest to przechwycenie definitywne ewentualny błąd nie wycieka poza blok " except" , jest tam przechwytywany i eliminowany.

```
x=1
y=0
try:
  z = x / y
except:
  print "Niepoprawne dane"
```

try-finally

- sekcja " finally" będzie wykonana zawsze. Jeśli błąd wystąpi, to zaraz po jego wystąpieniu, jeśli nie po ostatniej linii " try", zostanie wykonana pierwsza linia " finally"
- sekcja " finally" przekazuje ewentualne błędy dalej propagacja nie zostaje zatrzymana

```
x = 10
try:
  z = x / y
finally:
  print "koniec"
```

Przykłady praktyczne :

Wykorzystanie protokołu *http* przy użyciu Pythona.

W poniższym przykładzie wykorzystamy bibliotekę

„urllib” która umożliwi Nam bezpośrednią ingerencję w protokół dowolnej strony www.

Zaczniemy od importowania.

```
>>> from urllib import *
```

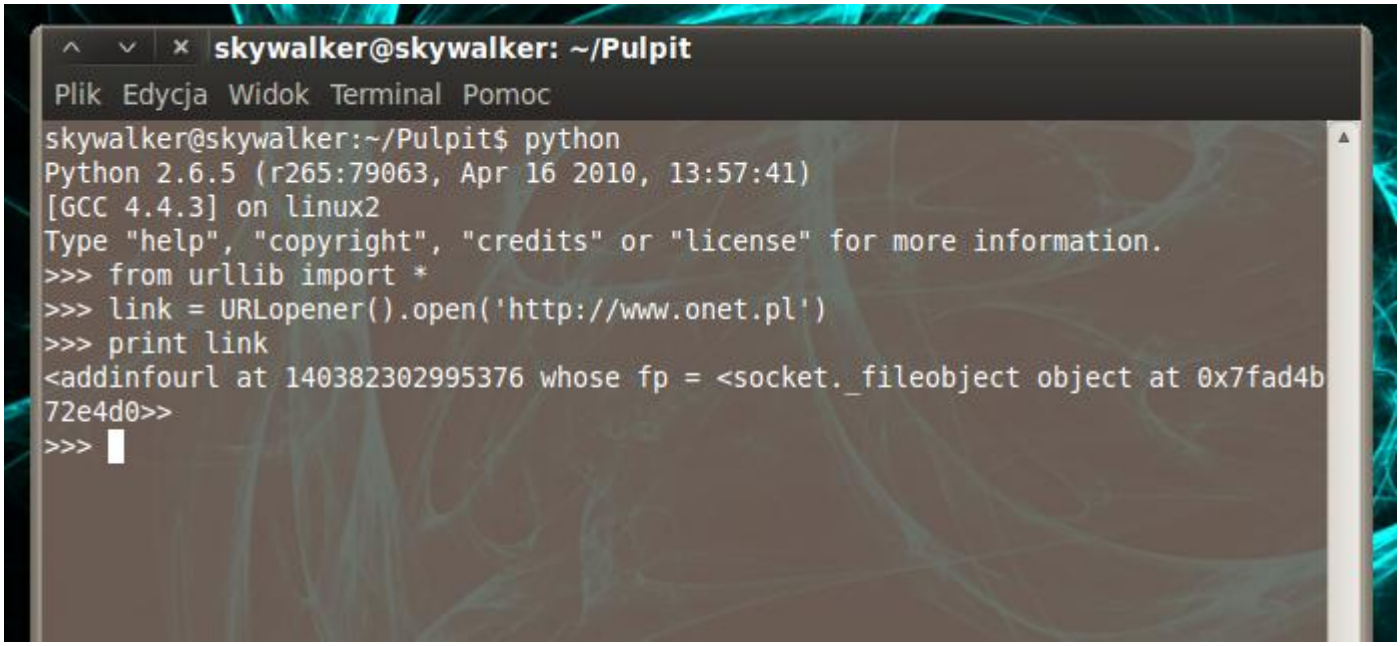
Obiekt 'link' wskazuje na metodę open() klasy URLopener, głównej klasy zdefiniowanej w urllib.py

```
>>> link =
URLopener().open('http://www.onet.pl'
```

Po wprowadzeniu polecenia :

```
>>> print link
```

Otrzymujemy komunikat świadczący o powodzeniu:

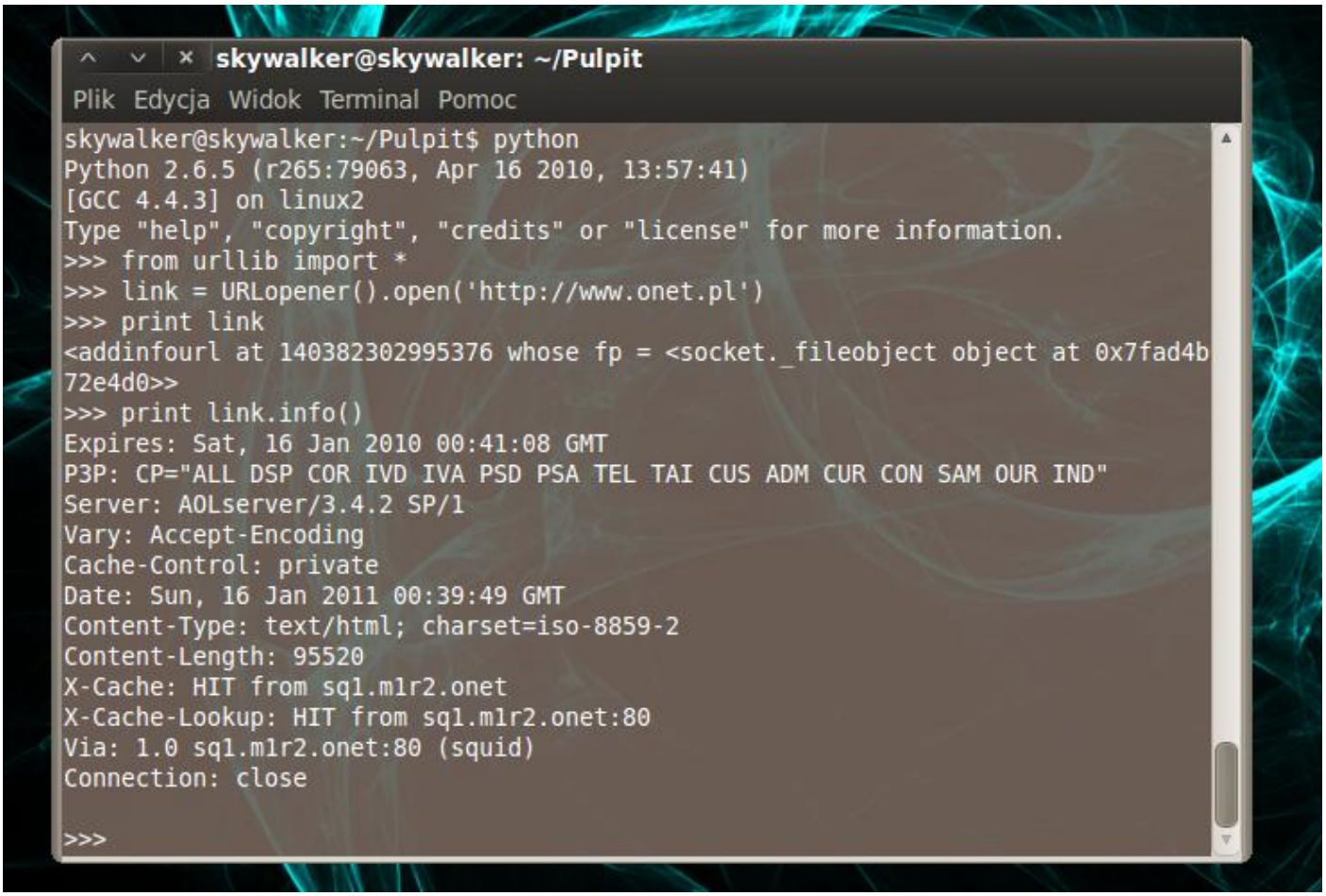


Strona jest już osiągalna, możemy dostać się do treści za pomocą funkcji readlines.

```
>>>print link.readlines()
```

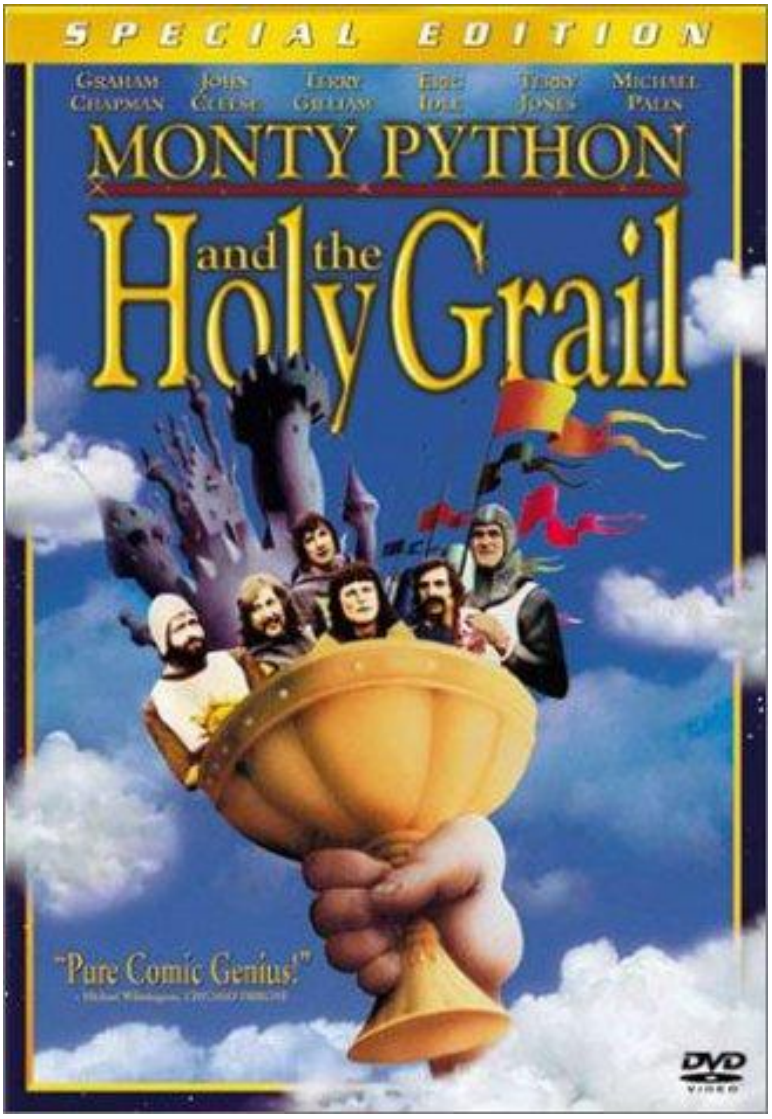
Otrzymamy w ten sposob pokaźną ilość informacji.Przy pomocy polecenia print link.info() wyłuskamy najistotniejsze informacje.

```
>>> print link.info()
```



W ten sposób uzyskaliśmy informacje o rodzaju używanego serwera, sposobie cachowania, dacie ostatniej aktualizacji, długości wątków, oraz typie bazy danych. Za pomocą tego narzędzia możemy podejrzeć witrynę dowolnego rodzaju.

CO TO ZA FILM :)
?



DZIĘKUJĘ ZA UWAGĘ.