Programowanie obiektowe - wykład 1

Roman Putanowicz

Katedra Technologii Informatycznych w Inżynierii

2 października 2019

W skrócie

- o przedmiocie
- o programowaniu obiektowym
- o Pythonie
- o narzędziach
- o kontekście dla projektów
- proste przykłady

Cel i efekty

Cele:

- Zapoznanie studentów z podstawowymi koncepcjami programowania obiektowego.
- Zapoznanie studentów z obiektowo zorientowanym językiem programowania wykorzystywanym do budowy i rozszerzania naukowych i inżynierskich środowisk obliczeniowych

Efekty:

- Znajomość klasyfikacji języków programowania. Znajomość charakterystyki obiektowych języków programowania.
- Znajomość składni języka Python w zakresie umożliwiającym programowanie obiektowe.
- Umiejętność analizy problemów z punktu widzenia analizy obiektowej. Umiejętność wyodrębniania obiektów i opisywania relacji między obiektami.
- Umiejętność tworzenia, uruchamiania i debugowania programów w języku Python.

Sposób pracy

Wymagania:

- Umiejętność programowania strukturalnego w dowolnym języku programowania.

Wykłady:

- Tematyka wykładów choć w zarysie naszkicowana, jest otwarta. Oczekuję propozycji zagadnień do poruszenia na wykładach czy też konkretnych pytań, na które Państwo chcielibyście usłyszeć odpowiedź.

Laboratoria:

- Laboratoria mają służyć przede wszystkim wyjaśnianiu konkretnych problemów z Państwa programami przy monitorze. Są okazją do pokazania Państwa **umiejętności i zaangażowania**. Podstawowy sposób pracy do **dyskusja**. Nad projektami możecie Państwo pracować w parach.

Ocenianie

Efekty		Zaangażowanie		
		małe	średnie	duże
	małe	2.0	3	3.5
	średnie	3.5	3.5	4
	duże	4	4.5	5

- Generalnie dzielę studentów na dwie kategorie: ci którzy chcą się czegoś nauczyć i pozostałych. Reszta to mniej istotne szczegóły.
- Trafność i obiektywność oceniania zależy w gruncie rzeczy od ilości informacji na temat efektów i zaangażowania w pracę, którą mi Państwo dostarczycie.
- **Zawsze można zawsze przyjść i podyskutować na temat konkretnej oceny.**

Tematy projektów

- Rzut ukośny w polu grawitacyjnym z oporami ruchu i bez, wersja 2D i 3D.
- Transformacje afiniczne siatek.
- Obliczanie charakterystyk geometrycznych figur płaskich i brył.
- Numeryczne całkowanie funkcji dwu zmiennych na siatkach.
- Generacja modelu geometrycznego obszaru ze sferycznymi pustkami (2D/3D).
- Analiza statyczna ramy płaskiej.

Literatura

Podstawowa

- Tony Gaddis, Python dla zupełnie początkujących, 2019, Helion S.A.
- Mark Lutz, Python, Wprowadzenie, 2011, Helion S.A.

Dodatkowa:

- Michał Jaworowski, Tarek Ziade, Profesjonalne programowanie w Pythonie. Poziom ekspert, 2017, Helion S.A.
- Gilles Dowek, Principles of Programming Languages, 2009, Springer *Dostępne w Bibliotece PK - zasoby elektroniczne*
- Kent D. Lee, Foundations of Programming Languages, 2014, Springer *Dostępne w Bibliotece PK - zasoby elektroniczne*
- Iain Graig, The Interpretation of Object Oriented Programming Languages, 2002, Springer *Dostępne w Bibliotece PK - zasoby elektroniczne*

Paradygmaty programowania

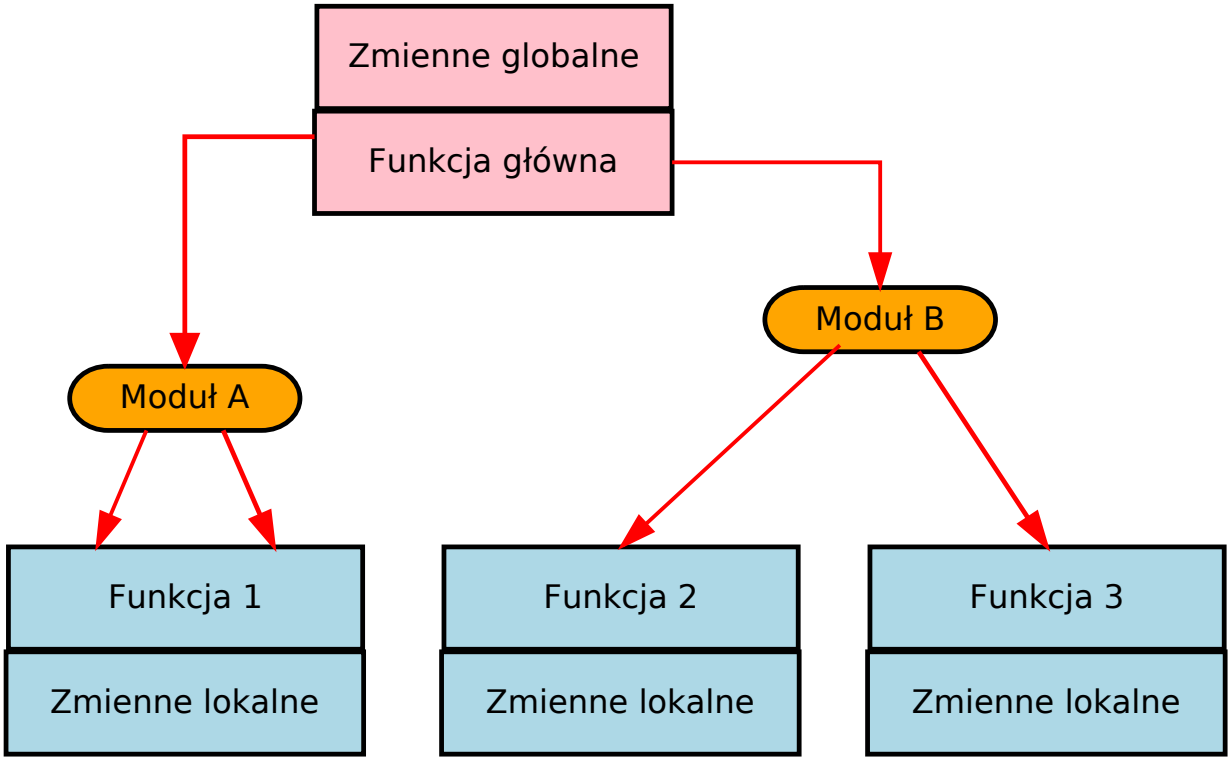
Programowanie:

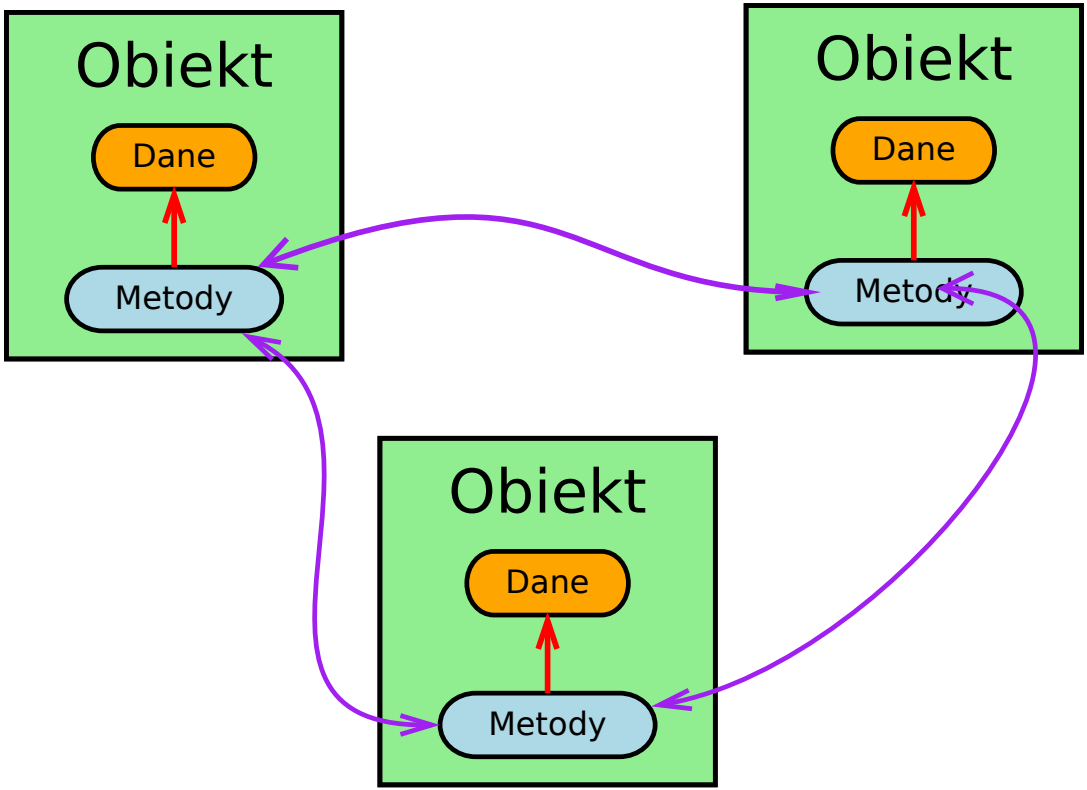
- **Imperatywne** : C, C++, Java, Kotlin, PHP, Python, Ruby, Wolfram Language
- **Strukturalne** : C, C++, Java, Kotlin, Pascal, PHP, Python, Wolfram Language
- **Proceduralne** : C, C++, Java, Kotlin, Pascal, PHP, Python, Wolfram Language
- **Funkcyjne** : C++, Clojure, Elixir, Erlang, F#, Haskell, Java (od wersji 8), Kotlin, Lisp, Python, JavaScript
- **Zdarzeniowe** : JavaScript, ActionScript, Visual Basic, Elm
- **Obiektowe** : Common Lisp, C++, C#, Eiffel, Java, Kotlin, PHP, Python, Ruby, Scala, JavaScript
- **Deklaratywne** : SQL, regular expressions, Prolog, OWL, SPARQL, Prolog

Więcej szczegółów :

`https://en.wikipedia.org/wiki/Comparison_of_programming_paradigms`

Programowanie strukturalne




Obiekt
Dane
Metody
Obiekt
Dane
Metody
Obiekt
Dane
Metody

Jakie znaczenia ma wybór języka

- Który język jest “najlepszy”?
- Domain Specific Languages (DSL)
- “Hot language” : Python

Można zadać pytanie jaki jest wpływ wyboru języka programowania na efektywność prowadzenia badań naukowych (lub dydaktyki)? Pytanie wydaje się proste ale odpowiedź jest bardzo skomplikowana.

“Konkurs pięknosci” języków

TIOBE Index : `https://www.tiobe.com/tiobe-index/`

Sep 2019	Sep 2018	Change	Language	Ratings	Change
1	1		Java	16.661%	-0.78%
2	2		C	15.205%	-0.24%
3	3		Python	9.874%	+2.22%
4	4		C++	5.635%	-1.76%
5	6	change	C#	3.399%	+0.10%
18	13	change	MATLAB	1.062%	-0.21%
34			Fortran	0.359%	
35			Ada	0.346%	
36			Julia	0.338%	
37			Kotlin	0.337%	

Latami: Fortran (1957/1977), C(1972), C++ (1980), Matlab(1984), Wolfram (1988), Python (1991), Fortran90 (1992), R (1993), Java (1995), C# (2001), Scala (2003), Kotlin (2011), Julia (2012)

Domain Specific Languages (DSL)

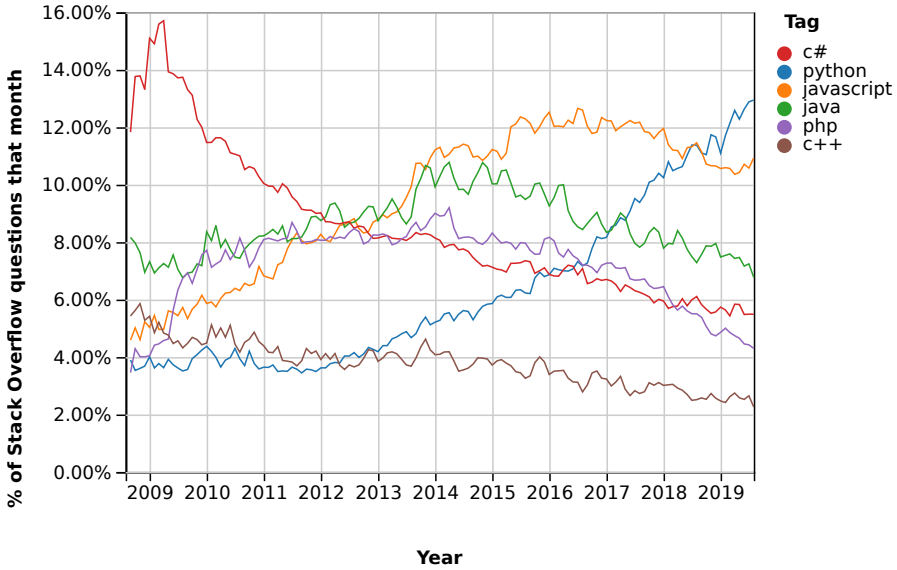
- Języki do modelowania geometrycznego : e.g. GMSH
- Języki dla MES: FreeFem, Fiat(FENics), Unified Form Language(FENIcs), GetFEM high level assembly
- Języki dla operacji na tensorach: packages for Mathematica, Maple, Maxima, Matlab, R, Yoric,
- Języki do programowania grafiki: Asymptote, tikz(LaTeX)
- Języki programowania wizualnego : Dynamo, Grashooper

Evolucje języków

- Fortran: 77, 95, 2008
- C++: 1998, C++03, C++11, C++14, C++17, C++20 (wkrótce)
- Python: Python2, Python3 (od 2008)
- Matlab: 2008 - znaczące rozszerzenia obiektowe

“Hot” language: Python

From: Stack Overflow Trends



Now, the Developer Survey 2019 reveals that Python has “edged out Java” and is the second most loved language. Stack Overflow refers to Python as the “fastest-growing major programming language”.

`https://jaxenter.com/stack-overflow-dev-survey-2019-157815.html`

Python: podstawowe informacje

https://en.wikipedia.org/wiki/Python_(programming_language)
Python - interpretowany język wysokiego poziomu, ogólnego przeznaczenia. Zaprojektowany przez Guido van Rossuma i po raz pierwszy opublikowany w 1991. Umożliwia programowanie z wykorzystaniem różnych paradygmatów programowania, w szczególności programowanie obiektowe. W roku 2008 została opublikowana wersja 3.0 Pythona, która jest niekompatybilna z wersją 2. Wersja 2.7 jest ostatnią z wersji 2 i nie będzie dalej rozwijana.

```
1       def hello_world():
2           print('Hello world')
3
4       if __name__ == '__main__':
5           hello_world()
6
```

Implementacje Pythona

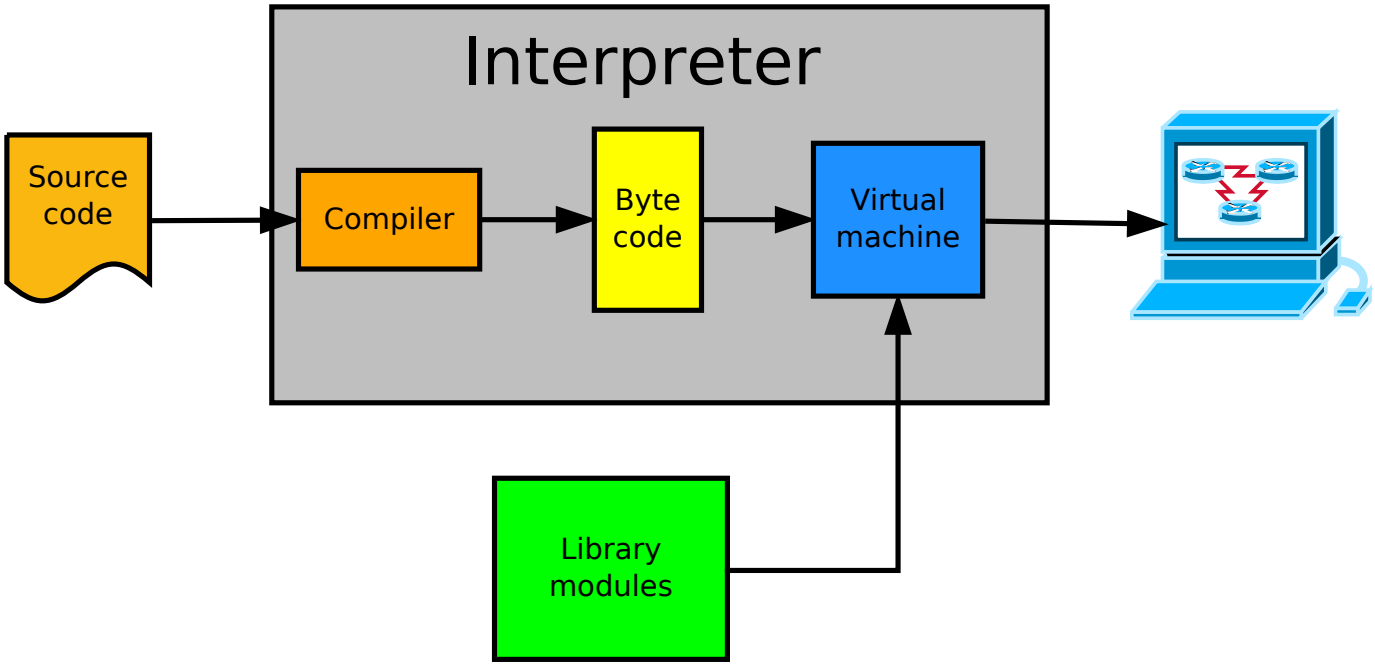
- CPython - podstawowa implementacja w języku C
- Stackless Python - implementacja w C (unika korzystania ze stosu systemowego)
- Jython - implementacja w Javie
- IronPython - implementacja dla środowiska .NET Framework
- PyPy - implementacja Pythona w Pythonie

Organizacja kodu: pakiety i moduły

- Dla prostych programów cały kod w jednym skrypcie.
- Złożone programy podzielone są na pakiety i moduły:
  - moduł - plik *.py zawierający kod
  - pakiet - folder zawierający minimum plik __init__.py oraz jakieś moduły lub inne pakiety.

Korzystanie z zawartości modułów i pakietów wymaga ich "importowania", które jest realizowane za pomocą instrukcji `import`.

Sposób wykonywanie programów Pythona



Rysunek na podstawie `https://indianpythonista.wordpress.com/2018/01/04/how-python-runs/`

“Prosty” program

```
1         # -*- coding: utf-8 -*-
2   """
3   Calculate distance between two points with coordinates
4   read from file my_points.txt
5   """
6   import math
7
8   def read_points(filename):
9       with open('my_points.txt') as input_file:
10          points = []
11          for line in input_file:
12              points.append(list(map(float, line.split())
        ))
13          return points
14
15  def segment_length(pt1, pt2):
16      length = 0
17      for c1, c2 in zip(pt1, pt2):
18          length += (c1-c2)**2
19      return math.sqrt(length)
20
21  if __name__ == '__main__':
22      pts = read_points('my_points.txt')
23      if len(pts) == 2:
24          distance = segment_length(pts[0], pts[1])
25          print(f"Distance between points : {distance}")
26      else:
27          print(f"Error: expected 2 points got {len(pts)}
        ")
28
```

Dystrybucje Pythona

- Oficjalna dystrybucja CPython
- Anaconda (Continuum Analytics)
- Canopy (Enthought)
- ActivePython (ActiveState)

Zen of Python in Easter Egg

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Narzędzia

Absolutne minimum: interpreter Pythona + edytor

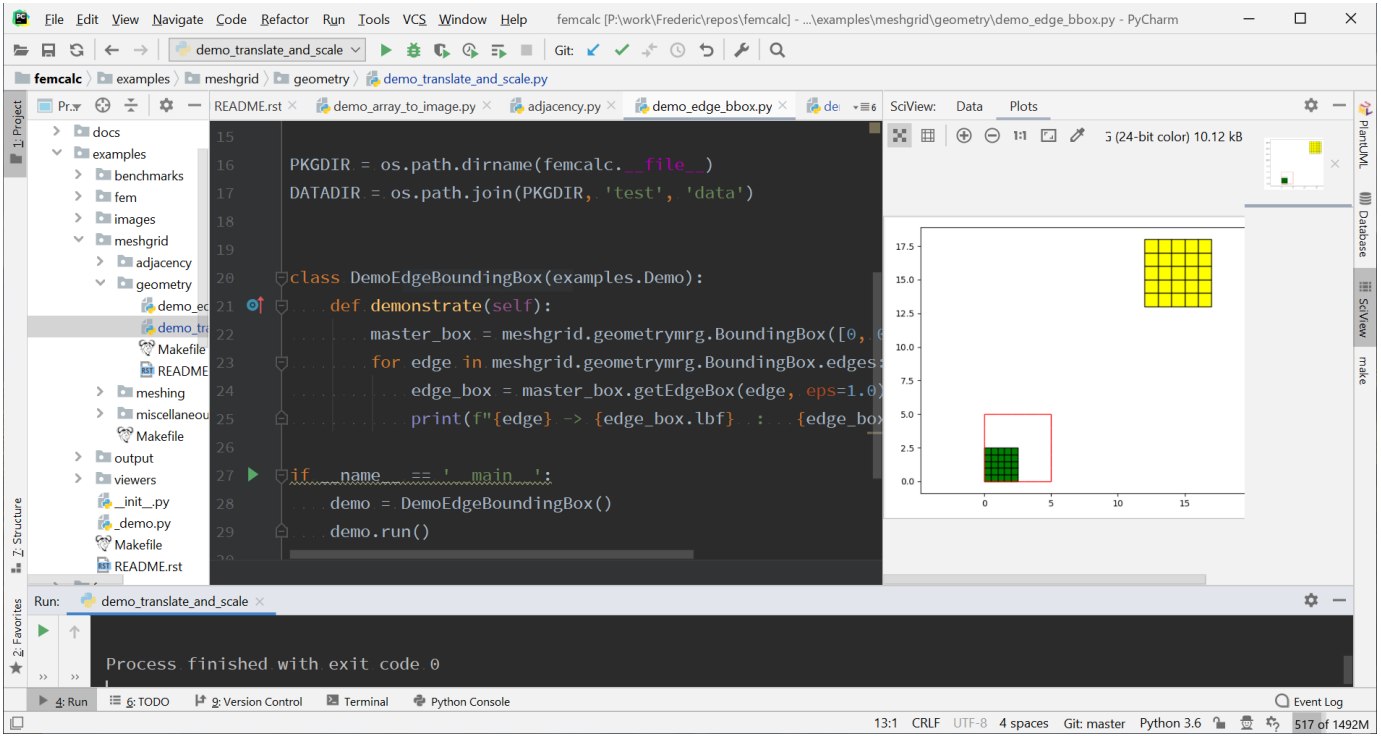
Efektywne programowanie wymaga dodatkowych narzędzi:

- „inteligentne” zintegrowane środowisko tworzenia programów (ang. IDE - Integrated Development Environment), np. **PyCharm**
- system kontroli wersji: np. **Git, Subversion** + webowy serwis, np. **GitHub, BitBucket**
- narzędzia do tworzenia dokumentacji, np. **Sphinx, pandoc, LaTeX**
- powłoka: np. **Bash, PowerShell**

Dodatkowo potrzebujemy narzędzi specyficznych dla obszaru zastosowań:

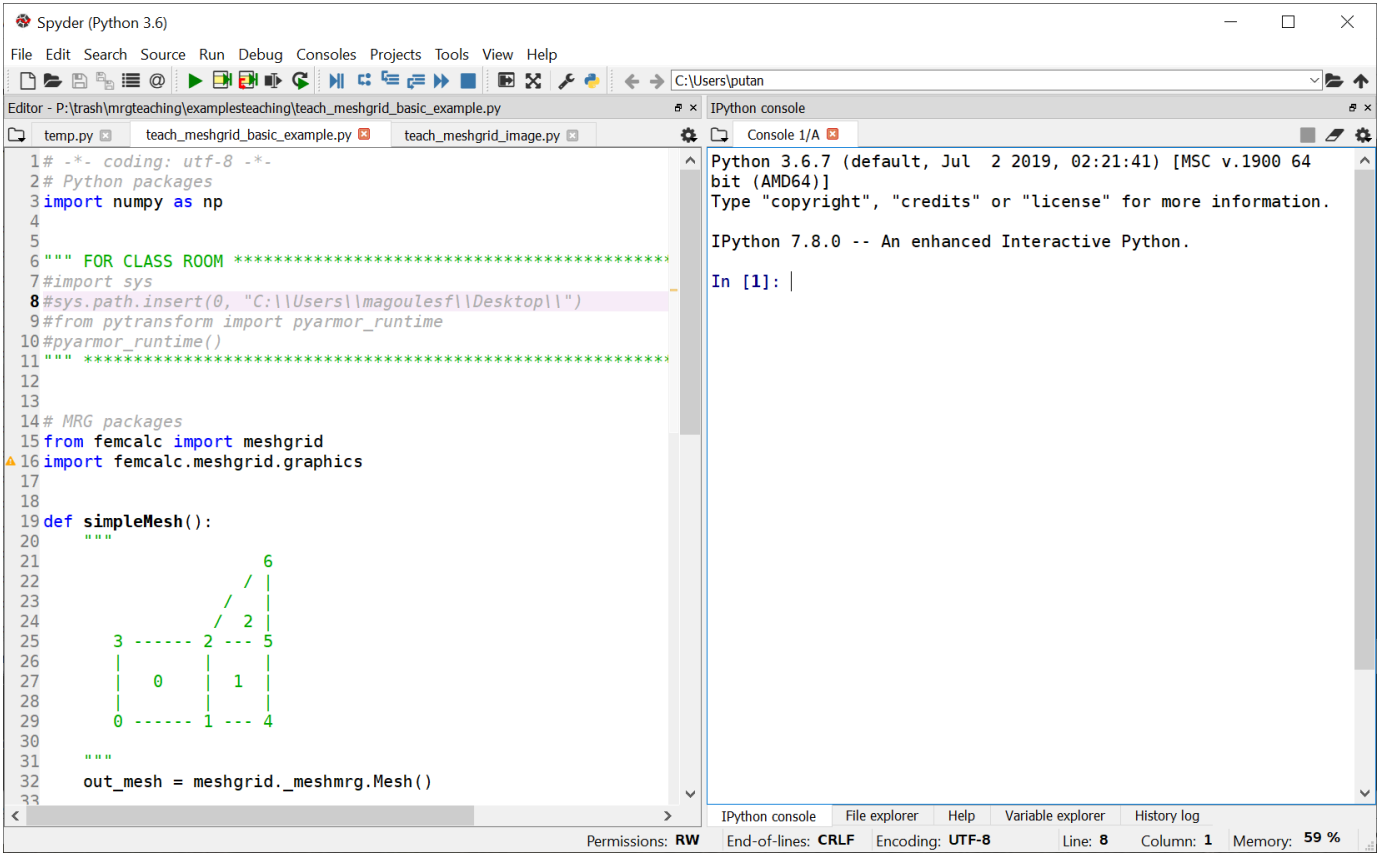
- modelowanie geometryczne i generacja siatek
- wizualizacja danych

PyCharm



https://www.jetbrains.com/pycharm/

Spyder



https://www.spyder-ide.org/

Git

System kontroli wersji (ang. revision control system) to narzędzie bez którego nie zapewne nie powstałby żaden poważny projekt oprogramowania Git (`https://git-scm.com/`) to rozproszony system kontroli wersji, obecnie jedno z najpopularniejszych narzędzi w swojej kategorii.

Minimalny zestaw poleceń Git'a: `clone`, `add`, `commit`, `status`, `diff`, `pull`, `push`

Git'a najlepiej nauczyć się obsługiwać z poziomu wiersza poleceń a potem dopiero używać jakiegoś GUI.

Instalowanie pakietów

Jeżeli korzystamy z dystrybucji Anaconda:

```
conda install nazwa_pakietu
```

Jeżeli korzystamy z `pip`:

```
python -m pip install nazwa_pakietu
pip install nazwa_pakietu
```

Generalnie zaleca się nie mieszać `conda` z `pip`. Jeżeli już to należy zainstalować wszystkie potrzebne pakiety używając `conda` a dopiero gdy czegoś brakuje można użyć `pip`. Zaleca się też korzystać z dobrodziejstw wirtualnych środowisk (`virtualenv` - temat na poźniej).

Dokumentowanie kodu

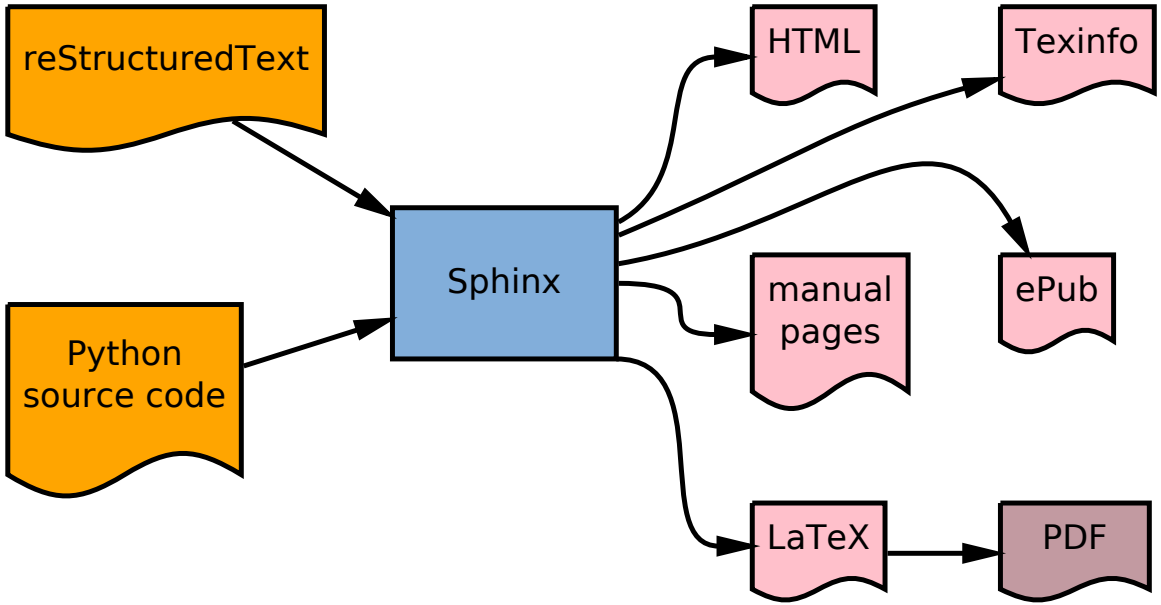
- https://numpydoc.readthedocs.io/en/latest/format.html
- `https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_numpy.html`

Każdy pakiet, moduł, klasa, funkcja powinny być opatrzone notką dokumentacyjną (ang. docstring) (poniżej w konwencji pakietu `Numpy`)

```
1  def translate(points, displacement):
2      """Translate points.
3
4      Parameters
5      ----------
6      points :  array
7          Array o points (one point per row).
8      displacement : vector
9          The translation vector
10
11     Returns
12     -------
13     array
14         Array of translated points.
15     """
16     return points + displacement
```

Sphinx

Spinx `http://www.sphinx-doc.org` - to system tworzenia dokumentów w różnych formatach ze źródeł w formacie reStructuredText. Pozwala na automatyzację tworzenia wysokiej jakości dokumentacji bezpośrednio z notek informacyjnych i elementów składni Pythona.



reStructuredText

reStructuredText to język znaczników przeznaczony do szybkiego tworzenia dokumentacji technicznej. Jest wykorzystywany w:

- Docutils
- Sphinx
- Trac

W połączeniu z LaTeX'em jest wykorzystywany do tworzenia zaawansowanej dokumentacji technicznej (LaTeX - zwłaszcza do zaawansowanego składania tekstów, np. matematycznych)

```
1. Chapter one
==============

  * the first list element
  * the second **important** element
  * the third with some formula :math:`x^2`
```

LaTeX(wymowa latech lub lejtech) to język do formatowania dokumentów tekstowych, oraz związane z nim oprogramowanie pozwalające na renderowanie tych dokumentów w formatach dla tradycyjnego druku lub elektronicznego rozpowszechniania.

Jaki jest związek z LaTeX'a z programowaniem obiektowy i Pythonem? Pierwsza odpowiedź to tworzenie dokumentacji. Druga, to że LaTeX podobnie jak Python to świetne narzędzie a lepiej jest znać dwa świetne narzędzia niż jedno. Wreszcie praca z Pythonem i LaTeX'em jest powszechnie zalecana jako środek zapobiegawczy atrofii umysłowej.

Edytory: Kile, Texmaker, TeXworks, TeXnicCenter
Dystrybucje (dla Windows): MikTeX‘

Grafika w Pythonie

Python nie ma żadnych mechanizmów grafiki wbudowanych w język ale dostarcza znakomite pakiety to programowania grafiki:

- Matplotlib - `https://matplotlib.org/`
- Pillow - fork of PIL (Python Imaging Library) for Python 3
- scikit-image - `https://scikit-image.org/`

Matplotlib będzie naszym podstawowym narzędziem do wizualizacji danych i algorytmów. **UWAGA:** Matplotlib dostarcza dwu interfejsów a) obiektowego i b) wzorowanego na funkcjach Matlaba. Warto do początku zacząć używać interfejsu obiektoweg.

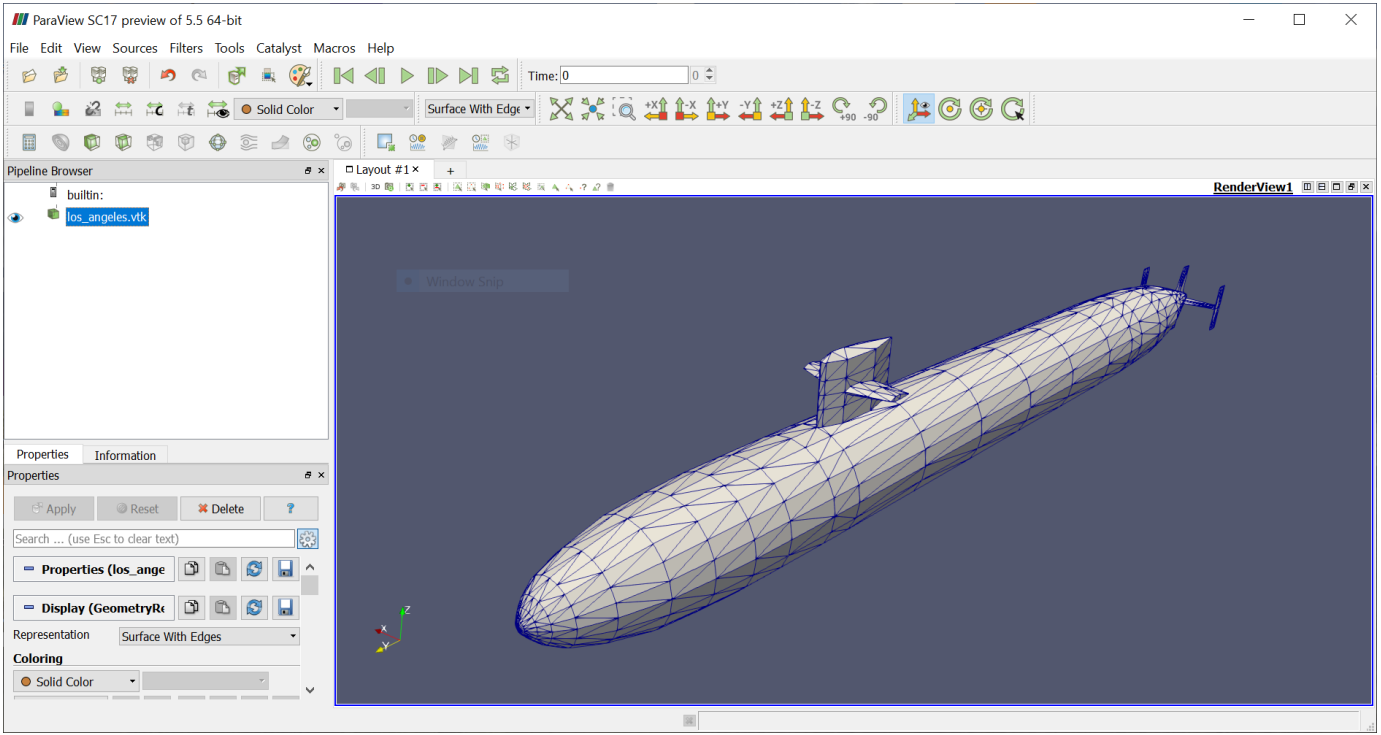
Obilczenia numeryczne

Jeżeli mamy jakiś problem z zakresu numeryki (generalnie jakiś problem) to z 98% pewnością został on już rozwiązany w Pythonie. Trzeba tylko znaleźć odpowiedni moduł. W przypadku numeryki szukamy w pakietach:

- `NumPy` - `https://numpy.org/`
- `SciPy` - `https://www.scipy.org/`
- `Sympy` - `https://www.sympy.org`

Jak zawsze pomaga nam Stack Overflow `https://stackoverflow.com` czyli społeczność programistów.

ParaView



`https://www.paraview.org/`

ParaView można kontrolować i rozbudowywać poprzez skrypty Pythona.

O co chodzi z tymi kontekstami?

Programowanie w zasadzie nie jest niczym innym jak tworzeniem precyzyjnego opisu rzeczy i procesów. Najłatwiej opisywać coś co się dobrze zna i rozumie, a jednocześnie coś co już jest dalece sformalizowane. Dodatkowo dobrze wybrać takie zagadnienia, które w miarę prosto jest przedstawić graficznie - ułatwi to ocenę czy program działa dobrze i jednocześnie samo zadanie będzie atrakcyjniejsze.

- Rozwiązywanie problemów początkowych dla układów równań różniczkowych zwyczajnych
- Operacje na siatkach obliczeniowych
- Całkowanie numeryczne
- Metoda Elementów Skończonych (konstrukcje ramowe)

Kontekst - ODE (Ordinary Differential Equations)

- Modelowy problem - rzut ukośny w polu grawitacyjnym z uwzględnieniem oporów ruchu
- Wyprowadzenie równań ruchu
- Metody całkowania ODE

Kontekst - siatki obliczeniowe

- Siatki strukturalne i niestrukturalne
- Opis geometrii i "topologii" siatki
- Transformacje afiniczne siatek
- Operacje kombinatoryczne na siatkach

Kontekst - całkowanie numeryczne

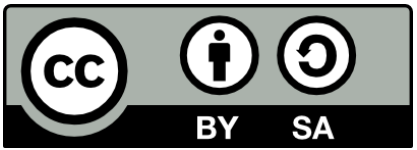
- Koncepcja elementu wzorcowego
- Opis geometrii elementu za pomocą funkcji kształtu
- Konstrukcja kwadratur w 2D

Kontekst - FEM - element ramowy

- Macierze elementowe
- Opis obciążeń i podparcia konstrukcji
- Algorytm agregacji równań algebraicznych MES

Dziękuję za uwagę

Copyrights



Roman.Putanowicz@L5.pk.edu.pl