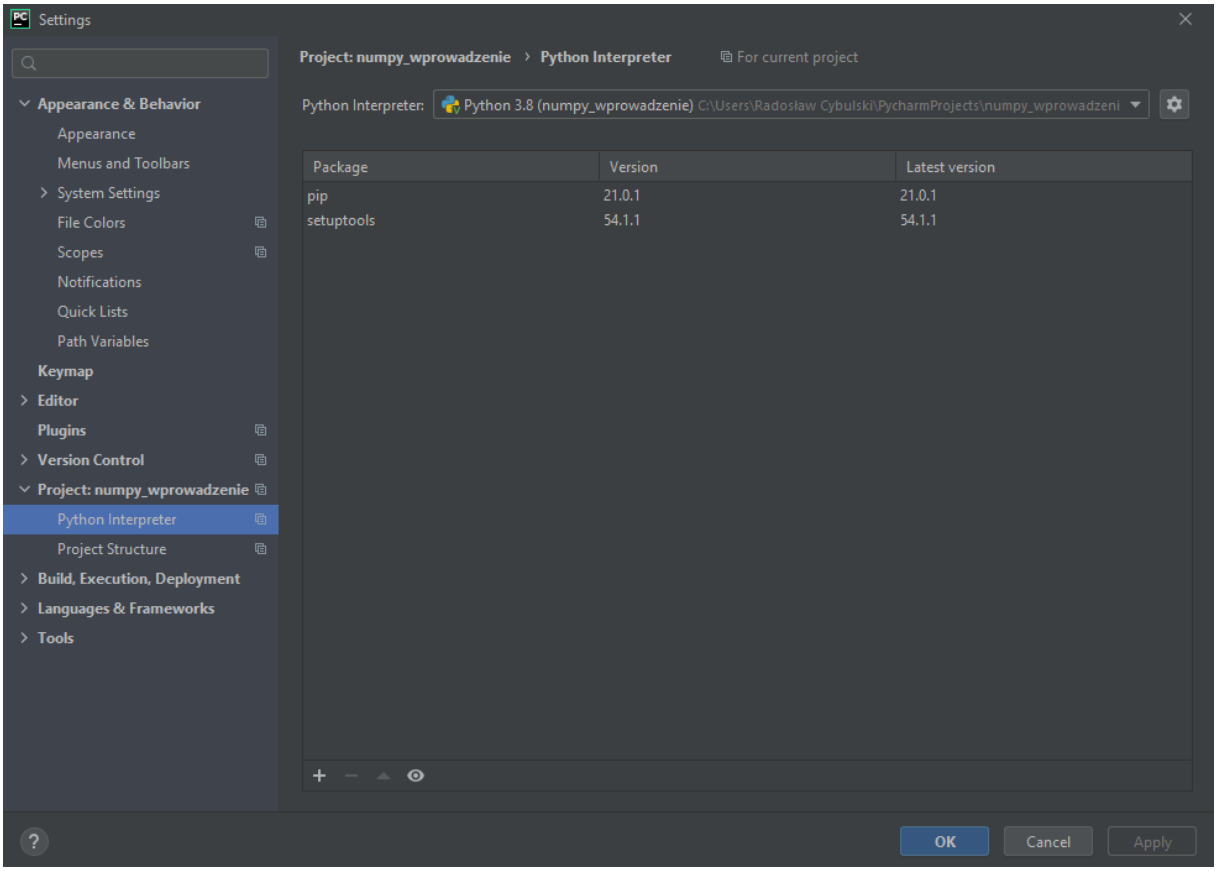
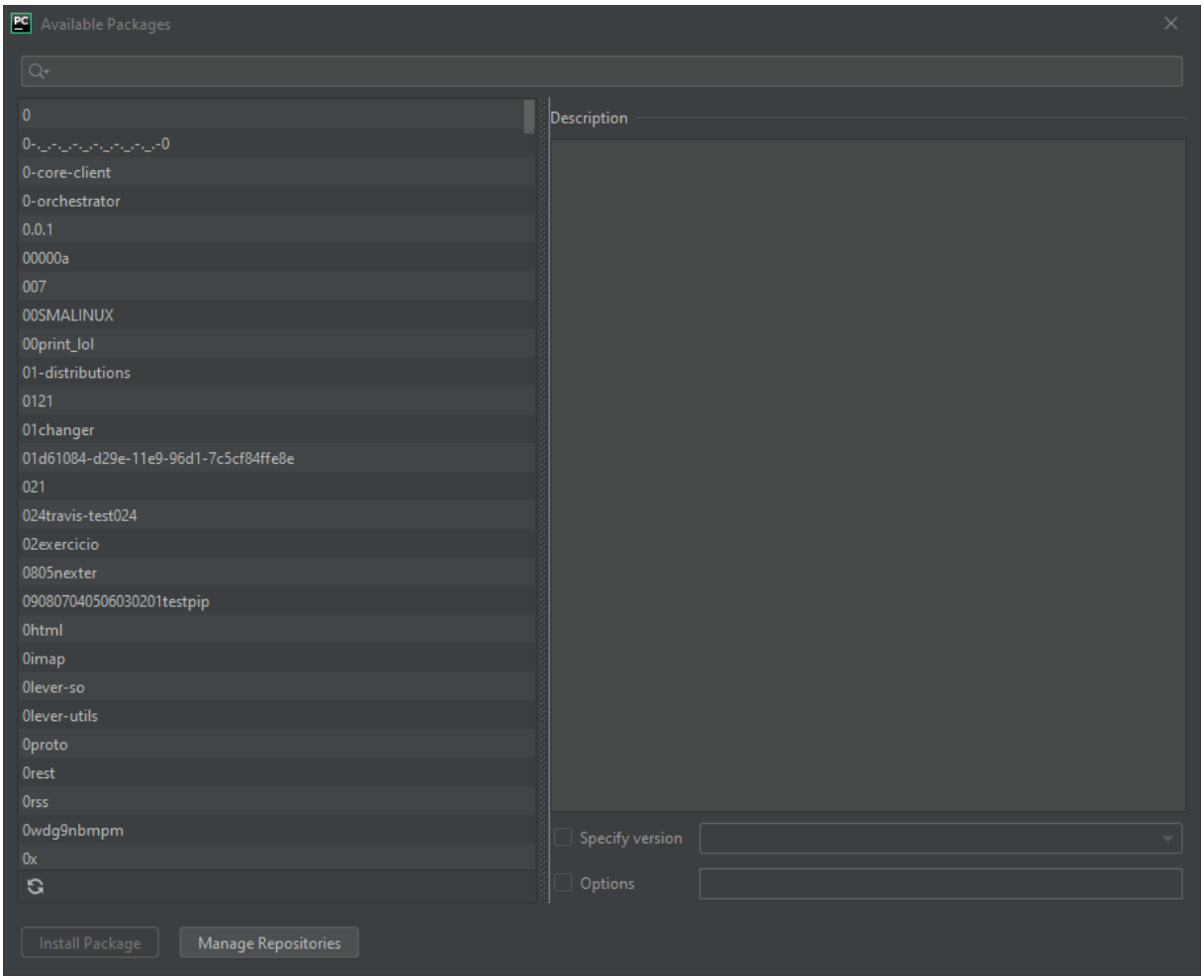
Instalacja biblioteki numpy

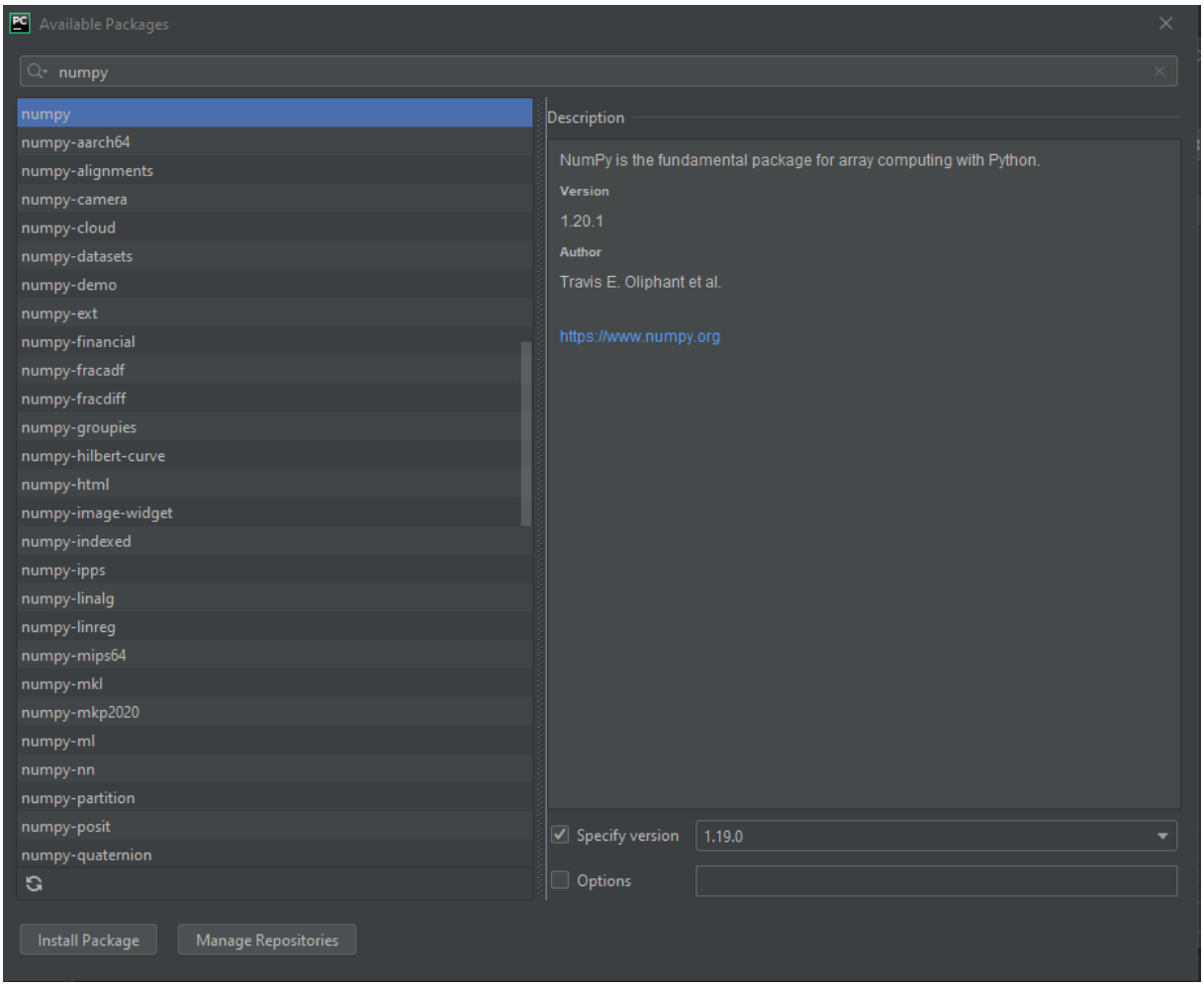
W nowo utworzonym projekcie przechodzimy do File/Settings/Project: project_name i wybieramy Python Interpreter



Następnie klikamy przycisk plusa w celu dodania do projektu wybranej biblioteki. Po naciśnięciu plusa otworzy się następujące okno.



Wpisujemy numpy w wyszukiwarkę, zaznaczamy Specify version i wybieramy wersję 1.19.0. Widok jak na zdjęciu poniżej.



Klikamy na install package i czekamy na zakończenie procesu instalacji.

Tworzenie tablic w numpy

Tablice biblioteki Numpy to kolekcje, które mogą przechowywać dane jednorodne, czyli dane tego samego typu. Taki stan rzeczy powoduje, że w kwestii przechowywania danych nie są tak uniwersalne jak listy, ale z racji tego, że znając typ danych, który będzie przechowywany można łatwo obliczyć jaki będzie rozmiar tablicy w pamięci. Dzięki temu Numpy może wykonywać operacje na całych wektorach wartości a nie na pojedynczych elementach jak w przypadku list. Biblioteka Numpy w znakomitej części jest napisana w języku C co zapewnia bardzo wysoką wydajność większości operacji. Deklaracja tablicy korzystającej z podobnego mechanizmu działania jak funkcja range():

```
import numpy as np
a = np.arange(2)

print(a)
```

Po wypisaniu zmiennej otrzymamy informację postaci [0 1].

Faktyczną nazwą klasy dla tablicy Numpy jest ndarray co stanowi skrót od n-dimensional array czyli tablicę n-wymiarową.

Przykład 1.

```
import numpy as np
#inicjalizacja tablicy
a = np.array([0, 1])
print(a)
#lub drugi sposób
a = np.arange(2)
print(a)
#wypisanie typu zmiennej tablicy (nie jej elementów) - ndarray
print(type(a))
#sprawdzenie typu danych tablicy
print(a.dtype)
#inicjalizacja tablicy z konkertnym typem danych
a = np.arange(2, dtype ='int64')
print(a.dtype)
#zapisywanie kopii tablicy jako tablicy z innym typem
b = a.astype('float')
print(b)
print(b.dtype)
#wypisanie rozmiaru tablicy
print(b.shape)
# można też sprawdzić ilość wymiarów tablicy
print(a.ndim)
#stworzenie tablicy wielowymiarowej może wyglądać tak
#parametrem przekazywanym do funkcji array jest obiekt, który
zostanie skonwertowany na tablicę
#może to być Pythonowa lista
m = np.array([np.arange(2), np.arange(2)])
print(m.shape)
#ponownie typem jest ndarray
print(type(m))
```

Pełna lista typów danych, które możemy umieścić w tablicach Numpy znajduje się pod adresem https://numpy.org/doc/

Przykład 2

```
import numpy as np
#możemy w łatwy sposób stworzyć macierz danego rozmiaru
wypełnioną zerami lub jedynkami
zera = np.zeros((5,5))
jedynki = np.ones((4,4))
print(zera)
print(jedynki)
#warto sprawdzić jaki jest domyślny typ danych takich tablic
```

```
print(zera.dtype)
print(jedynki.dtype)
#można również stworzyć "pustą" macierz o podanych wymiarach,
która pusta wcale nie jest
#wartości umieszczane są losowe, najpierw podawana jest ilość
wierszy tablicy, potem ilość kolumn
pusta = np.empty((2,2))
print(pusta)
#do elementów tablicy możemy odwołać się tak jak do elementów
np. listy czyli podając indeksy
poz_1 = pusta[1,1]
poz_2 = pusta[0,1]
print(poz_1)
print(poz_2)
#tworzenie macierzy 2x2 wraz z uzupełnieniem
macierz = np.array([[1,2],[3,4]])
print(macierz)
#funkcja arange potrafi takrze tworzyć ciągi liczb
zmiennoprzecinkowych
liczby = np.arange(1,2,0.1)
print(liczby)
#podobnie działa funkcja linspace, które działanie jest
równoważne tej samej funkcji w MATLAB-ie
liczby_lin = np.linspace(1,2,5)
print(liczby_lin)
#a teraz możemy utworzyć dwie macierze, najpierw wartości
interowane są w kolumnie a następnie w wierszu
z = np.indices((5,3))
print(z)
#wielowymiarowe macierze możemy również generować funkcją
mgrid
x, y = np.mgrid[0:5, 0:5]
print(x)
print(y)
#podobnie jak w MATLAB-ie możemy tworzyć macierze diagonalne
mat_diag = np.diag([a for a in range(5)])
print(mat_diag)
#w powyższym przykładzie stworzony wektor wartości zostanie
umieszczony na głównej przekątnej macierzy
#możemy podać drugi parametr funkcji diag, który określa
indeks przekątnej względem głównej przekątnej
#która zostanie wypełniona wartościami podanego wektora
mat_diag_k = np.diag([a for a in range(5)],-1)
print(mat_diag_k)
#Numpy jest wstanie stworzyć tablicę jednowymiarową z
dowolnego obiektu iterowalnego (iterable)
z = np.fromiter(range(5), dtype='int32')
print(z)
#ciekawą funkcją Numpy jest funkcja frombuffer, dzięki której
możemy stworzyć np. tablicę znaków
marcin = b'Marcin'
```

```
# mar = np.frombuffer(marcin,dtype='S1')
# print(mar)
# mar_2 = np.frombuffer(marcin,dtype='S2')
# print(mar_2)
#powyższa funkcja ma jednak pewną wadę dla pythona 3.x, która
powodóje, że trzeba jawnie określić
#iż ciąg znaków przekazujemy jako ciąg bajtów co osiągamy po
przez podanie litery 'b' przed wartością
#zmiennej tekstowej. Można podobne efekty osiągnąć inaczej
marcin = 'Marcin'
mar_3 = np.array(list(marcin))
mar_4 = np.array(list(marcin), dtype='S1')
mar_5 = np.array(list(b'Marcin'))
mar_6 = np.fromiter(marcin,dtype='S1')
mar_7 = np.fromiter(marcin,dtype='U1')
print(mar_3)
print(mar_4)
print(mar_5)
print(mar_6)
print(mar_7)
#talice w Numpy możemy w prosty sposób do siebie dodawać,
odejmować, mnożyć, dzielić
mat = np.ones((2,2))
mat_1 = np.ones((2,2))
mat = mat + mat_1
print(mat)
print(mat - mat_1)
print(mat*mat_1)
print(mat/mat_1)
```

Indeksowanie i cięcia tablic

Cięcie i indeksowanie danych w tablicy Numpy jest możliwe do wykonania na bardzo wiele sposobów. Poniżej przykłady niektórych z nich.

```
import numpy as np
#cięcie (slicing) tablicy numpy można wykonać za pomocą
wartości z funkcji slice lub range
a = np.arange(10)
print(a)
s = slice(2,7,2)
print(a[s])
s = range(2,7,2)
print(a[s])
#możemy ciąć tablice również w sposób znany z cięcia list
(efekt jak wyżej)
print(a[2:7:2])
#lub tak
print(a[1:])
print(a[2:5])
#w podobny sposób postępujemy w przypadku tablic
```

```
wielowymiarowych
mat = np.arange(25)
#teraz zmieniamy kształt tablicy jednowymiarowej na macierz
5x5
mat = mat.reshape((5,5))
print(mat)
print(mat[1:]) #od drugiego wiersza
print(mat[:,1]) #druga kolumna jako wektor
print(mat[...,1]) #to samo z wykorzystaniem ellipsis (...)
print(mat[:,1:2]) #druga kolumna jako ndarray
print(mat[:,-1:]) #ostatnia kolumna
print(mat[2:6, 1:3]) # 2 i 3 kolumna dla 3,4,5 wierszy
print(mat[:, range(2,6,2)]) # 3 i 5 kolumna
print('')
#bardziej zaawansowane, lecz trudniejsze do zrozumienia cięcia
można osiągnąć wg. poniższego przykładu
#y będzie tablicą zawierającą wierzchołki macierzy x
x = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])
print(x)
rows = np.array([[0, 0], [3, 3]])
cols = np.array([[0, 2], [0, 2]])
y = x[rows,cols]
print(y)
```