Tryb awaryjny w Python - poznajemy obsługę wyjatków

W programowaniu (w tym w Pythonie) poznaliśmy już najważniejsze z jego składowych instrukcji. Obserwujemy, jak pisane przez nas skrypty są w sposób uporządkowany wykonywane - linijka po linijce, funkcja po funkcji. Wszystko jest uporządkowane, jak jak np. ruch na drodze. Na każdym skrzyżowaniu dokładnie wiadomo, kto i w jakiej kolejności będzie je pokonywał. Jednak nie zawsze ten porządek jest zachowany. Czasem trafiają się sytuacje wyjątkowe, kiedy trzeba odłożyć wykonywanie poszczególnych instrukcji na bok i skupić się na ratowaniu całej sytuacji.

Okazuje się, że większość współczesnych języków programowania - a zatem i Python - posiada wbudowane mechanizmy z odmiennym - ratunkowym trybem pracy. Swoistym pasem dla karetek pogotowia. W tej lekcji opowiemy Państwu o mechanizmie obsługi sytuacji wyjątkowych w Python.

Błąd a wyjątek

Bardzo często w początkowym etapie nauki programowania mylone są dwa pojęcia - błędu i wyjątku. Stąd też pojawia się czasami sformułowanie "obsługa błędów". W informatyce posługujemy się następującym rozróżnieniem:

- *błędem* (Error) nazywać będziemy sytuację, w której doszło do nieprawidłowego działania programu i w efekcie nie da się kontynuować jego pracy;
- *wyjątkiem* (Exception) nazywać będziemy sytuację, w której doszło do nieprawidłowego działania programu, ale możliwe jest kontynuowanie jego działania po dokonaniu naprawy;

Przykładem błędu sytuacja, gdy na skutek awarii sprzętu część danych z działania programu została utracona. Przykładem wyjątku natomiast, gdy na skutek błędu wprowadzania z klawiatury użytkownik polecił pracować z nieistniejącym (lub nie pasującym do wzorca) plikiem. Drugą z sytuacji można by naprawić prosząc go ponowne podanie ścieżki pliku.

Składnia obsługi wyjątków w Python

Składnia dotycząca obsługi wyjątku jest bardzo podobna do tej dostępnej w C++, Javie oraz innych współczesnych języków. Składa się z reguły z dwóch bloków kodu

- blok przechwytywania - blok instrukcji które są zagrożone niepoprawnym wykonaniem, oraz
- blok obsługi - blok instrukcji opisujących instrukcje mające na celu przeprowadzenie naprawy tej sytuacji.
-

try:
  blok przechwytywania
except:
  blok obsługi
Zaprezentujmy to na możliwie najprostszym przykładzie. Chcemy dokonać konwersji tekstu na liczbę. Np. '1234' na liczbę 1234

In [1]:
```
tekst = "1234"
x = int(tekst)
print(f'Wczytaliśmy wartość {x}')
```
```
Wczytaliśmy wartość 1234
```
Jednak kiedy spróbujemy wszystko wykonać dla danych wprowadzonych bez znajomości działania naszego kodu

In [2]:
```
tekst = "sowa"
x = int(tekst)
print(f'Wczytaliśmy wartość {x}')
```
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-2-122968d176bb> in <module>
      1 tekst = "sowa"
----> 2 x = int(tekst)
      3 print(f'Wczytaliśmy wartość {x}')

ValueError: invalid literal for int() with base 10: 'sowa'
```
otrzymamy komunikat o przerwaniu działania. Funkcja int nie umiała sobie poradzić z przekazym jej parametrem i zakończyła swoje działanie w stanie niepoprawnym (nie była w stanie wygenerować liczby na podstawie przekazanego tekstu). Przypuśćmy, że problem ten chcielibyśmy naprawić poprzez wstawienie wartości 11 w każdej sytuacji, gdy konwersja się nie powiedzie. Zostanie to zapisane tak
In [3]:
```
tekst = "sowa"
try: # rozpoczynamy kod zagrożony niepoprawnym przetworzeniem
    x = int(tekst)
except: # tu znajduje się kod naprawy sytuacji z błędem
    x = 1
print(f'Wczytaliśmy wartość {x}')
```
```
Wczytaliśmy wartość 1
```
W szczególności zauważmy, że nie pojawiła się żadna informacja o tym, że nastąpił jakiś błąd. Program mógł się dalej wykonywać zupełnie poprawnie.

Proces obsługi wyjątku

Proces obsługi wyjątku można sobie wyobrazić za pomocą naszej metafory z pasem dla karetek. Pamiętamy, że karetka jest pojazdem uprzywilejowanym jedynie w chwili gdy ma uruchomione sygnały dzwiękowe i świetlne - przez pozostały czas porusza się zgodnie z przepisami ruchu drogowego - zatem w szczególności nie może korzystać z pasa dla pojazdów uprzywilejowanych.

Zupełnie podobnie działają interpretery - mają zapasowy porządek przetwarzania instrukcji (dodatkowy pas), który jest wykorzystywany do radzenia sobie z wyjątkowymi sytuacjami. Omówimy to dokładnie pokazując odpowiedni przykład

In [4]:
```
lista = ['1234', '1000000', 'sowa', 'lis', '1001']
wyniki = []
try:
    for liczba in lista:
        wyniki.append(int(liczba)) #konwertuj liczbe i dodaj na liste
except:
    print('Wystapił błąd w przetwarzaniu')
print(f'Program kończy działanie. Lista ma postać {wyniki}')
```
```
Wystapił błąd w przetwarzaniu
Program kończy działanie. Lista ma postać [1234, 1000000]
```

Zauważmy, co dzieje się w programie. Mamy listę elementów, które mają zostać konwertowane do liczb. Wśród nich jest nasz literał 'sowa', który nie poddaje się ten konwersji.

1. Początkowo program działa poprawnie - elementy z listy '1234' oraz '1000000' zostają poprawnie skonwertowane i dodane do listy wynikowej.
2. Pojawia się literał 'sowa' i instrukcja int('sowa') nie zostaje poprawnie wykonana. W informatyce nazywamy ten moment zgłoszeniem (lub rzuceniem) wyjątku (**raise** , **throw**).
3. Uruchomiony zostaje tryb awaryjny (karetka włącza syrenę i zjeżdża na pas dla karetek).
4. Wszystkie instrukcje zapisane w Python od tego miejsca są pomijane (jedziemy innym pasem).
5. Docieramy do najbliższego bloku obsługi (sprawdzamy czy może on nam pomóc - aka czy ten szpital ma odpowiedni oddział, jeśli tak to wykonujemy instrukcje naprawy). Tu wypisywany jest komunikat ('wystąpił błąd w przetwarzaniu')
6. Powracamy do normalnego przetwarzania ale już na końcu po zakończeniu tego bloku.

Zauważmy, że na skutek przechodzenia na koniec bloku dwie rzeczy się nie stały się w naszym kodzie

- literał 'lis' nie spowodował rzucenia wyjątku. Komunikat jest w końcu tylko 1.
- Liczba '1001' nie została skonwertowana i dołączona do listy.

Zupełnie inaczej wykonuje się kod zapisany z przestawiem instrukcji pętli i bloków obsługi wyjątków

In [5]:
```
lista = ['1234', '1000000', 'sowa', 'lis','1001']
wyniki = []
for liczba in lista:
    try:
        wyniki.append(int(liczba)) #konwertuj liczbe i dodaj na liste
    except:
        print('Wystapił błąd w przetwarzaniu')
print(f'Program kończy działanie. Lista ma postać {wyniki}')
```
```
Wystapił błąd w przetwarzaniu
Wystapił błąd w przetwarzaniu
Program kończy działanie. Lista ma postać [1234, 1000000, 1001]
```

Filtrowanie wyjątków

Okazuje się, że porównanie z różnymi oddziałami szpitali ma swój metaforyczny odpowiednik w składni Pythona. Okazuje się bowiem, że można utworzyć wiele bloków obsługi dla potencjalnie różnych rodzajów awarii, które mogę się trafić w bloku przechwytywania. Określić, jaki blok obsługi jest odpowiedni, można na podstawie danych które mamy o problemie - czyli typie zmiennej która została wygenerowana do zgłoszenia tej sytuacji.

Zobaczmy na przykładzie

In [6]:
```
lista = ['1234', 3j+7, 'sowa', 'lis','1001']
wyniki = []
for index in range(7):
    try:
        wyniki.append(int(lista[index])) #konwertuj liczbe i dodaj na liste
    except TypeError:
        print(f'Zły typ - otrzymano {type(lista[index])}')
    except ValueError:
        print(f'Podano złą wartość {lista[index]}')
    except IndexError:
        print(f'Próba siegniecia poza zakres tablicy na index {index}')
print(f'Program kończy działanie. Lista ma postać {wyniki}')
```
```
Zły typ - otrzymano <class 'complex'>
Podano złą wartość sowa
Podano złą wartość lis
Próba siegniecia poza zakres tablicy na index 5
Próba siegniecia poza zakres tablicy na index 6
Program kończy działanie. Lista ma postać [1234, 1001]
```
Widzimy w powyższym przykładzie, że w zależności od typu problemu mogą być uruchamiane różne bloki naprawcze. Natomiast jeśli odpowiedni blok naprawczy się nie pojawi

In [7]:
```
lista = ['1234', 3j+7, 'sowa', 'lis','1001']
wyniki = []
for index in range(7):
    try:
        wyniki.append(int(lista[index])) #konwertuj liczbe i dodaj na liste
    except TypeError:
        print(f'Zły typ - otrzymano {type(lista[index])}')
    except ValueError:
        print(f'Podano złą wartość {lista[index]}')
print(f'Program kończy działanie. Lista ma postać {wyniki}')
Zły typ - otrzymano <class 'complex'>
Podano złą wartość sowa
Podano złą wartość lis
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-7-af97c92972d3> in <module>
      3 for index in range(7):
      4     try:
----> 5         wyniki.append(int(lista[index])) #konwertuj liczbe i dodaj
na liste
      6     except TypeError:
      7         print(f'Zły typ - otrzymano {type(lista[index])}')

IndexError: list index out of range
```

przetwarzanie w trybie wyjątku nie zostaje zawieszone i powoduje zakończenie programu z komunikatem o błędzie.

Popularne typy wyjątków

Warto wyjaśnić, że rodzajów błędów (typów danych reprezentujących błąd) może być w zasadzie dowolnie wiele. Jednak zwłaszcza na początku pracy z językiem warto wyszczególnić kilka z nich - choćby dlatego, że możemy uzyskać informację o ich wyniknięciu z naszego kodu

Typ wyjątku	Opis
IOError	Błąd odczytu pliku lub podobnego zasobu
IndexError	Błąd próby pobrania nie istniejącego elementu z kolekcji
KeyError	Brak danego klucza w kolekcji

Typ wyjątku	Opis
NameError	Brak zdefiniowania zmiennej o danej nazwie
SyntaxError	Niepoprawnie użyta składnia języka
TypeError	Próba użycia niepasującego typu
ValueError	Próba użycia niepoprawnej wartości
ZeroDivisionError	Próba dokonania dzielenia przez 0

Zgłaszanie wyjątków

Aby przykłady, które mamy tu powyżej, działały - musiało zostać spełnione pewne wymaganie. Otóż funkcja int() - a dokładnie jej autorzy - musieli przewidzieć, że może dojść do takiej sytuacji gdy ktoś podaje na jej wejście niepoprawny literał. Problem nie leży w ich kodzie - gdyby tak było, to by go naprawili. Problem leży w tym, że ktoś źle korzysta z przygotowanego przez nich kodu. Jak zatem dać temu komuś znać, że źle robi? Najprościej właśnie poprzez rzucenie wyjątku. W tej części zaprezentujemy jak można zgłosić wyjątek w wybranej linijce kodu i rozpocząć całe przetwarzanie.

In [8]:
```
def funkcja_z_wyjatkiem(x):
    if type(x) is int:
        return 1
    elif type(x) is float:
        return 2
    else:
        raise ValueError('x ma niepoprawny typ')

dane = [3.14, 3, 'sowa']

for dana in dane:
    print(funkcja_z_wyjatkiem(dana))
```
```
2
1
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-8-39cc1ed0cc5d> in <module>
     10
     11 for dana in dane:
---> 12     print(funkcja_z_wyjatkiem(dana))

<ipython-input-8-39cc1ed0cc5d> in funkcja_z_wyjatkiem(x)
      5         return 2
      6     else:
----> 7         raise ValueError('x ma niepoprawny typ')
      8
      9 dane = [3.14, 3, 'sowa']

ValueError: x ma niepoprawny typ
```

Kluczowe okazuje się tu zgłoszenie błędu, czyli instrukcja **raise**. Tak dokładnie to w linijce tej tworzony jest nowy obiekt w typie ValueError. Jednak do pełnego zrozumienia niuansów tej składni potrzebny będzie kolejny temat Państwa szkolenia - programowanie obiektowe.

Obsługa plików w Python

Do pracy z plikami w Python kluczowe jest opanowanie działania funkcji open

uchwyt_do_pliku = open(nazwa_pliku,tryb_pracy)

- nazwa_pliku oczywiście oznacza ścieżkę do pliku w systemie operacyjnym,
- tryb pracy może być odrobinę enigmatycznym. Jest to wyrażony za pomocą string zestaw nastawień dla danego pliku. W szczególności
  - "r" oznacza uprawnienia do odczytu, wyjątek oznacza brak pliku
  - "a" oznacza uprawnienia na dopisywanie do pliku (append), jeśli plik nie istnieje to zostanie utworzony
  - "w" oznacza uprawnienia do zapisu (nadpisania) pliku, jeśli plik nie istnieje to zostanie utworzony
  - "x" oznacza utworzenie pliku. Jeśli plik istnieje - rzucony zostanie wyjątek
  - "t" oznacza plik otwarty jako tekstowy (obszary pamięci interpretowane jako znaki alfabetu)
  - "b" oznacza plik otwarty jako binarny (obszary pamięci interpretowane jako kolejne liczby) domyślny tryb_pracy to "rt"
- zwracany jest uchwyt do pliku, co można interpretować jak kursor czytający dany plik.

należy również pamiętać, że pliki otwarte za pomocą **open** - powinny zostać na końcu zamknięte operacją **close**.

Do przeczytania elementów pliku można np. użyć funkcji czytających całą linijkę tekstu **readline**

In [9]:

```
plik = open('iliada.txt', 'rt')
```

```
for i in range(10):
    print(plik.readline(), end='')
plik.close()
```

```
Book I

Sing, O goddess, the anger of Achilles son of Peleus, that brought countles
s ills upon the Achaeans. Many a brave soul did it send hurrying down to Ha
des, and many a hero did it yield a prey to dogs and vultures, for so were
the counsels of Jove fulfilled from the day on which the son of Atreus, kin
g of men, and great Achilles, first fell out with one another.

And which of the gods was it that set them on to quarrel? It was the son of
Jove and Leto; for he was angry with the king and sent a pestilence upon th
e host to plague the people, because the son of Atreus had dishonoured Chry
ses his priest. Now Chryses had come to the ships of the Achaeans to free h
is daughter, and had brought with him a great ransom: moreover he bore in h
is hand the sceptre of Apollo wreathed with a suppliant's wreath and he bes
ought the Achaeans, but most of all the two sons of Atreus, who were their
chiefs.

"Sons of Atreus," he cried, "and all other Achaeans, may the gods who dwell
in Olympus grant you to sack the city of Priam, and to reach your homes in
safety; but free my daughter, and accept a ransom for her, in reverence to
Apollo, son of Jove."

On this the rest of the Achaeans with one voice were for respecting the pri
est and taking the ransom that he offered; but not so Agamemnon, who spoke
fiercely to him and sent him roughly away. "Old man," said he, "let me not
find you tarrying about our ships, nor yet coming hereafter. Your sceptre o
f the god and your wreath shall profit you nothing. I will not free her. Sh
e shall grow old in my house at Argos far from her own home, busying hersel
f with her loom and visiting my couch; so go, and do not provoke me or it s
hall be the worse for you."
```

Funkcja write

Do zapisywania w pliku należy używać funkcji **write**. Zaprezentujemy jej działanie przepisując 5 pierwszych linijek z pliku **iliada.txt** do pliku **kopia.txt**

In [10]:

```
plik = open('iliada.txt', 'rt')
kopia = open('kopia.txt', 'wt')
for i in range(5):
    linijka = plik.readline()
    kopia.write(linijka)
plik.close()
kopia.close()
```

Upewnijmy się odczytując całą zawartość pliku kopia.txt - przy okazji pokażemy jak można iterować po linijkach pliku

In [11]:
```
plik = open('kopia.txt', 'rt')
for linijka in plik:
    print(linijka)
plik.close()
```

```
Book I


Sing, O goddess, the anger of Achilles son of Peleus, that brought countles
s ills upon the Achaeans. Many a brave soul did it send hurrying down to Ha
des, and many a hero did it yield a prey to dogs and vultures, for so were
the counsels of Jove fulfilled from the day on which the son of Atreus, kin
g of men, and great Achilles, first fell out with one another.


And which of the gods was it that set them on to quarrel? It was the son of
Jove and Leto; for he was angry with the king and sent a pestilence upon th
e host to plague the people, because the son of Atreus had dishonoured Chry
ses his priest. Now Chryses had come to the ships of the Achaeans to free h
is daughter, and had brought with him a great ransom: moreover he bore in h
is hand the sceptre of Apollo wreathed with a suppliant's wreath and he bes
ought the Achaeans, but most of all the two sons of Atreus, who were their
chiefs.
```

Przydatna może być również następująca wersja

In [12]:
```
plik = open('kopia.txt', 'rt')
for no, linijka in zip(range(5),plik):
    print(f'{no} - {linijka}')
plik.close()
```

```
0 - Book I

1 -

2 - Sing, O goddess, the anger of Achilles son of Peleus, that brought coun
tless ills upon the Achaeans. Many a brave soul did it send hurrying down t
o Hades, and many a hero did it yield a prey to dogs and vultures, for so w
ere the counsels of Jove fulfilled from the day on which the son of Atreus,
king of men, and great Achilles, first fell out with one another.

3 -

4 - And which of the gods was it that set them on to quarrel? It was the so
n of Jove and Leto; for he was angry with the king and sent a pestilence up
on the host to plague the people, because the son of Atreus had dishonoured
Chryses his priest. Now Chryses had come to the ships of the Achaeans to fr
ee his daughter, and had brought with him a great ransom: moreover he bore
```

in his hand the sceptre of Apollo wreathed with a suppliant's wreath and he besought the Achaeans, but most of all the two sons of Atreus, who were the ir chiefs.