SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

USPOREDBA MAINSTREAM PROGRAMSKIH JEZIKA – PYTHON I C++

Katarina Ramljak

Split,Svibanj 2021.

Sadržaj :

1.Uvod 3

2. Upravljanje memorijom 4

2.1 Konstruktori 5

..... 6

2.2 Oslobađanje memorije- *garbage collector* i destruktori 6

3. Nasljeđivanje 8

3.1 Overriding 9

3.2 Shadowing 10

3.3 Višestruko nasljeđivanje 11

3.4 Apstraktne klase i interface 12

4. Polimorfizam 13

4.1 Generici i template 13

5. Iznimke 15

6. RAII 16

7. Zaključak 17

8. Literatura 18

1.Uvod

U ovom radu obrađena je tema usporedba dva objektno-orijentirana jezika više razine, Python i C++.

Cilj je napraviti usporedbu C++ i Pythona po značajkama vezanih za objektno orijentirano programiranje. C++ je kompajlerski, dok je Python interpreterski programski jezik. Ova dva programska jezika razlikuju se u sintaksi, vremenu izvođenja programa i količini memorije koju zauzimaju kodovi. Prva značajka za usporedbu je upravljanje memorijom (*memory management*)- konstruktori, destruktori, garbage collection, oslobađanje memorije. Zatim su prikazane sitne razlike između nasljeđivanja i plomorfizma. Nakon toga princip rada RAII metode u C++, te koju metodu koristi Python.

2. Upravljanje memorijom

Upravljanje memorijom je upravljanje memorijom računala rezervacijom dijelova memorije za podatke, te oslobađanje iste od viška podataka. Odnosno, kod programskih jezika odnosi se na stvaranje i uništavanje varijabli i objekata koji nam neće biti potrebni. Ukoliko se memorija alocira, a nikad ne oslobađa, tada ćemo doći do problema nestanka memorije. Kod nekih jezika se ovo radi ručno, a neki jezici sami implementiraju ovu mogućnost, ili ju dobivaju uključivanjem vanjskih biblioteka.

Kod ručnog upravljanja memorijom programer odlučuje koji će objekt ili varijabla, biti uništen i kada, ovisno o tome da li ga planira koristiti dalje u kodu. Automatsko upravljanje memorijom se naziva *„prikupljanje otpada“* engl.(*Garbage Collection)* ukratko, prikupljanje otpada je metoda suprotna ručnom upravljanju memorijom. Memorija se u prikupljanju otpada automatski oslobađa na temelju periodičnog vremena ili na temelju određenih kriterija koji govore da se više ne koristi. C++ zahtijeva ručno upravljanje memorijom, ali i za njega postoje vanjske biblioteke koje imaju zadaću automatskog prikupljanja otpada. Za razliku od C++ , Python programski jezik koristi uglavnom automatsko, ali i ručno upravljanje memorijom. U komunikaciji sa operacijskim sustavom „dohvaća“ memorijske lokacije ukoliko su slobodne te ih koristi za izvođenje programa. Nakon uporabe te lokacije se „očiste“ i vrate OS-u za ponovnu upotrebu.

2.1 Konstruktori

Prilikom kreiranja objekta neke klase poziva se njegov konstruktor. To je specijalna metoda koja ima isto ime kao i klasa, nema povratnu vrijednost te služi za inicijalizaciju objekta. Također, konstruktorom možemo definirati da li su za kreiranje objekta potrebni ulazni parametri ili ne.

Primjerice:

```
class Osoba{
public:
      // …
};
…
Osoba Ante; // Ante je inicijaliziran podrazumijevanim konstruktorom
```

Svaka klasa uvijek ima barem jedan konstruktor, a on se zove podrazumijevani konstruktor eng.(*default constructor).* Za klasu Osoba on bi glasio na sljedeći način.

```
Osoba(){}; // defaultni konstruktor
```

Defaultni konstruktor postoji samo u slučaju ako nismo definirali niti jedan drugi konstruktor. Tada je on automatski implicitno dodan za svaku takvu klasu. On omogućuje da se objekt može kreirati bez ikakvih dodatnih parametara. Međutim, ukoliko sami napišemo neki drugi konstruktor tada defaultni konstruktor više ne postoji.

```
class Osoba{
public:
    string ime;
    string prezime;
    // konstruktor s parametrima
    Osoba(string _ime, string _prezime){
        ime = _ime;
        prezime = _prezime;
    }
};

…
Osoba Ivan("Ivan", "Ivanov"); // konstruktor s parametrima
Osoba Ante; // greška! podrazumijevani konstruktor više ne postoji!
```

Konstruktor u Pythonu je specijalna metoda. Za razliku od C++, kod Python-a je naziv konstruktora uvijek isti *__init__*.Ako ne dodamo konstruktor u klasu ili ga zaboravimo deklarirati, Python automatski stvara defaultni konstruktor. Također, i u Pythonu pored defaultnog konstruktora postoji konstruktor sa parametrima.

Primjer defaultnog konstruktora:

```
class MyClass:

    # defualtni konstruktor
    def __init__(self):
        self.a = 10

# kreiranje objekta klase
obj = MyClass()
```

Primjer konstruktora sa parametrima:

```
class MyClass:
    a=0
    b=0

    # konstruktor sa parametrima
    def __init__(self, a1, b1):
        self.a = a1
        self.b = b1

# kreiranje objekta klase i poziv konstruktora
obj = MyClass(10,20)
```

2.2 Oslobađanje memorije- *garbage collector* i destruktori

Kako smo već prije naveli C++ i Python, imaju različite načine upravljanja memorijom, tj. imaju različite načine alociranja i oslobođenja memorije. Kako je već navedeno, Python ima automatsko upravljanje memorijom, dok C++ programer ručno upravlja.

U Pythonu se automatski brišu memorijske lokacije koje se više ne koriste, te se vraćaju OS-u za ponovnu upotrebu. Proces čišćenja obavlja sakupljač otpada(GC) ukoliko je uključen, ukoliko nije može doći do curenja memorije. Referentno brojanje, označi-pobriši, označi-sažmi, zaustavi-kopiraj su algoritmi koje GC koristi prilikom skupljanja otpada. Ti algoritmi sakupljaju iskorištene objekte te ih spremaju u otpad oslobađajući memorijske lokacije. Međutim, postoje i objekti koji su recimo ciklički elementi neke liste te ih nije moguće pokupiti. Ti objekti ostaju u memoriji te su neupotrebljivi. Biblioteka gc-a sadrži mnogobrojne metode kojima manipuliramo izvršavanjem programa i memorijskim lokacijama koje taj program koristi. Te metode nam prikazuju trenutačno stanje memorije, sakupljive i nesakupljive objekte, proces skupljanja po generacijama, čiste memoriju ukoliko to mi želimo prije nego se prag koji pokreće automatsko čišćenje nadmaši. Korištenjem automatskog upravljanja memorijom pomoću sakupljača otpada pruža, efikasan, pouzdan i jednostavniji rad programera. Ukoliko programer želi potpunu kontrolu nad upravljanjem memorijom koristit će ručno upravljanje koje je vrlo složenije.

Primjer:

```
a=[1,2,3]
#a pokazuje na objekt [1,2,3]
b=a
#a i b pokazuju na objekt [1,2,3]
a[1] = 2
#a i b pokazuju na objekt koji se u međuvremenu promijenio i sad glasi [2,2,3]
a = [4,5,6]
#sad samo b pokazuje na [2,2,3]
b="a"
#sad više nitko ne pokazuje na [2,2,3] i garbage collector će ga u nekom trenutku izbrisati iz memorije
```

Kod C++ nema „***garbage collector-a***“ jer zahtijeva ručno upravljanje memorijom, u kojem kako je već navedeno, sam programer odlučuje kada će alocirati i oslobodit memoriju. Za oslobađanje memorije u C++ se koriste ***destruktori.***

Destruktori su specijalne fukncije koje nemaju povratnu vrijednost, niti ulazne parametre, destruktori brišu odnosno uništavaju objekt iz memorije. Imaju isto ime kao i klasa s znakom „~“(tilda) ispred imena. Redoslijed poziva destruktora je suprotan od redoslijeda pozivanja konstruktora.

Primjer destruktora

```
Ime_klase::~ime_klase()
{
}
```

Za dinamičko alociranje memorije koristi se operator ***new()***, a za oslobađanje memorije ***delete().***

Operator ***new*** i ***delete*** se koristi na sljedeći način:

```
int size = 10;          // broj elemenata budućeg polja
int *p_var = NULL;
p_var = new int [size];//veličina = broj elemenata(size) * veličina
tipa(int)

/* .......
kod koji koristi gore stvoreno polje
........*/

delete [] p_var;        // oslobađanje memorije
p_var = NULL;
```

Također, bitno je naglasiti da se u modernom C++ kodu operatori new i delete rijetko koriste, jer se dinamičkom alokacijom memorija alocira na heapu, što je sporije nego na stacku.

Python također ima destruktor, no on nije toliko potreban kao u C++-u jer python ima gc koji upravlja memorijom. Međutim, iako je memorija najčešće alociran resurs, postoje i socketi i veze baze podataka koje treba zatvoriti , te datoteke, međuspremnici i predmemorija koje treba isprazniti i još nekoliko resursa koje treba osloboditi kad objekt završi sa njima. Pa i python koristi destruktor, koji se za razlik od C++ uvijek zove isto: __del__.

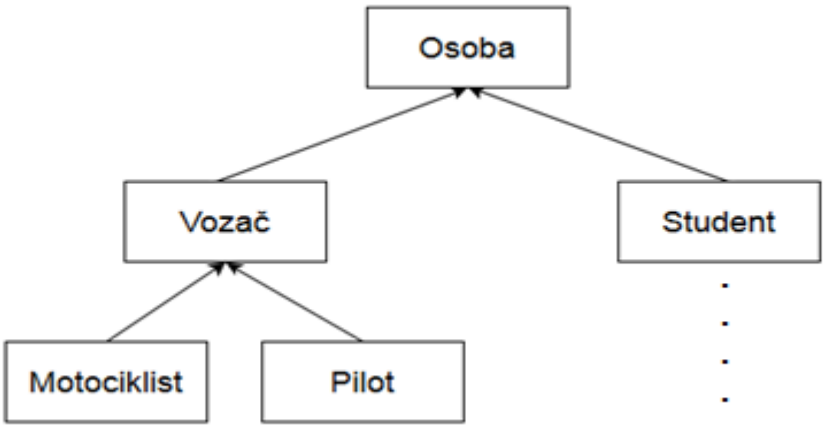
Primjer:

```
class MyClass():
        def __init__(self):
        def __del__(self):
```

3. Nasljeđivanje

Programski jezik C++ omogućuje deklaraciju nove klase na način da ona nasljeđuje neku već postojeću klasu. Prilikom nasljeđivanja nova klasa(izvedenica, derivacija) kopira članove bazne klase. Osim naslijeđenih članova bazne klase, nova klasa obično sadrži i svoje vlastite članove. Nasljeđivanjem smanjujemo količinu programskog koda pa je i vrijeme prevođenja aplikacije kraće. Bez svojstva nasljeđivanja bilo bi gotovo nezamislivo pisati složenije aplikacije.

Svrha korištenja nasljeđivanja ista je u Pythonu i C++.Jedna od razlika je to što je u Pythonu u izvedenoj klasi potrebno eksplicitno pozvati bazni konstruktor , dok se u C++ to obavlja automatski (skriveno).



Primjer nasljeđivanja u C++:

```
#include <iostream>
#include <string>
Using namespace std;
Class Osoba {
protected:
      string JMBG;
public:
      string ime;
      string prezime;
};
Class Student :  public Osoba {
public :
     string JMBG;
};
```

Primjer nasljeđivanja u Pythonu:

```
Class Osoba :
    def __init__(self,ime,prezime):
        self.ime = ime
        self.prezime = prezime
class Student (Osoba):
    def __init__ (self,ime,prezime):
        Osoba.__init__(self,ime,prezime): //poziavnje baznog konstruktora, što je u C++ automatski
```

Još jedna razlika između C++ i pythona vezana za nasljeđivanje je što python ima funkciju ***super(). Super()*** funkcija se upotrebljava u izvedenoj klasi i vraća objekt koji omogućuje referenciranje na baznu klasu.

Primjer nasljeđivanja sa funkcijom super() :

```
def __init__ (self, parametar1,..)
super().__init__(parametar1,..)
```

Prednost ovakog načina pisanja jest da prilikom slanja atributa u baznu klasu nije potrebno pisati self, te nije potrebno navoditi ime bazne klase , čime se olakšava potencijalna promjena bazne klase.

3.1 Overriding

U postupku nasljeđivanja izvedena klasa kopira sve dostupne članove bazne klase. Međutim, ukoliko se u izvedenoj klasi već nalazi metoda s istim imenom i parametrima kao i u baznoj klasi tada se kaže da metoda u izvedenoj klasi nadjačava metodu bazne klase (*eng. Overriding*). U tom slučaju će pri pozivu te metode biti će izvršena njena implementacija iz izvedene klase, dok će se implementaciji iz bazne klase moći pristupiti samo pomoću operatora **„::“**.

Što se tiče Python-a nema bitnih razlika između overridinga u C++ i overridinga u Pythonu, osim što se u Pythonu može koristiti funkcija super().

Primjer overridinga u C++:

```
class A{
public:
	void f(){
		cout << "A::f()" << endl;
	}
};
class B : public A{
public:
	// nadređenje metode void f()
	void f(){
		cout << "B::f()" << endl;
	}
};
...
B obj;
obj.f();	// B::f()
obj.A::f();	// A::f()
```

Primjer overridinga u Pythonu:

```
class Osoba:

    def __init__(self, ime, prezime):
        self.ime = ime
        self.prezime = prezime

    def ispis(self):
        print("Ja sam:", self.ime, self.prezime)


class Zaposlenik(Osoba):

    def __init__(self, ime, prezime, placa):
        super().__init__(ime, prezime)
        self.placa = placa

    def ispis(self):
        print("Ime:", self.ime)
        print("Prezime:", self.prezime)
        print("Plaća:", self.placa)
z = Zaposlenik("Pero", "Perić", 10000)
z.ispis()
```

3.2 Shadowing

Shadowing ili „prikrivanje imena“ se događa kada u izvedenoj klasi imamo varijable ili funkcije istog imena kao i u baznoj klasi. Kada se to dogodi varijabla ili funkcija izvedene klase „skriva“ varijablu bazne klase.

Skrivanje varijabli ili funkcija u svim jezicima funkcionira na isti princip, stoga nema razlike između C++ i Pythona. Međutim, skrivanje varijabli se preporučuje izbjegavati. U Pythonu se „skrivanja“ izbjegavaju na način da se koristi ***nonlocal*** funkcija, čime se varijabla dodaje nelokalnim varijablama ili funkcijom ***global*** kojom varijabla postaje globalna.

3.3 Višestruko nasljeđivanje

Već dugo vremena u praksi objektnog programiranja općepoznato je da je višestruko nasljeđivanje "loša stvar". U višestrukom nasljeđivanju klase mogu naslijediti značajke iz više baznih klasa. Međutim, C++ i Python podržavaju višesrtuko nasljeđivanje, dok neki jezici ne podržavaju. C++ i Python su dosta slični što se tiče višestrukog nasljeđivanja.

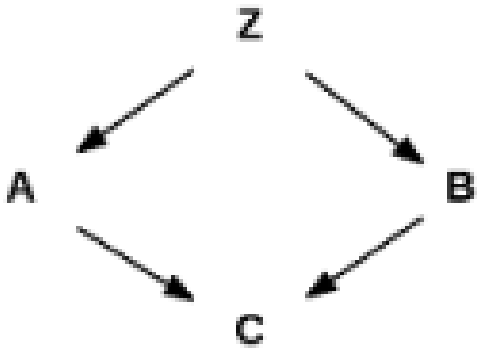
Primjer višestrukog nasljeđivanja u C++:

```
Class A{

};

Class B{

};

Class C : public A, public B{

};
```

Višestruko nasljeđivanje u Pythonu:

```
class Base: pass class Derived1(Base): pass class Derived2(Derived1): pass
```

Iako ova mogućnost može biti korisna, poželjno ju je izbjegavati zbog mogućeg problema dijamantne hijerarhijske strukture nasljeđivanja. Zbog toga u nekim jezicima višestruko nasljeđivanje poput C#-a i Jave nije moguće.



Kao što je na slici prikazano klase A i B su izvedene iz klase Z , a klasa C iz A i iz B. Ako postoji neka funkcija u klasi Z koju su A ili B ili čak oboje naslijedili , postavlja se pitanje koju funkciju će C naslijediti? Iz Z,A ili B?

U C++ ovaj problem se riješava virtualnim nasljeđivanjem. Virtualno nasljeđivanje omogućuje da izvedena klasa(u ovom slučaju C) nasljeđuje samo jednu funkciju iz bazne klase Z. Virtualno nasljeđivanje primjenjujemo dodavanjem ključne riječi ***virtual.***

```
Class Z{
};
Class A: virtual public Z{
};
Class B : virtual public Z {
};
Class D : public A, public B{
};
```

U Pythonu se ovaj problem rješava korištenjem **MRO** *(Method Resolution Order).* MRO određuje redoslijed po kojem se pretražuje neki atribut. Prvo se traži u trenutnoj klasi, ako nije nađeno traži se u baznim klasama , s lijeva u desno.

```
# Demonstration of MRO class X: pass class Y: pass class Z: pass class A(X, Y): pass
class B(Y, Z): pass class M(B, A, Z): pass # Output: # (, , # , , # , , # )
print(M.mro())


(,,,,,,,) IZLAZ
```

3.4 Apstraktne klase i interface

U objektno orijentiranom programiranju **apstraktna klasa** je računarska klasa koja ne može imati objekte. Njena osnovna svrha je da bude roditelj drugim, konkretnim klasama. Iz tog razloga ne može imati sve metode implementirane i treba imati barem jednu apstraktnu metodu, tj. čisto virtualnu metodu.

U C++ virtualne funkcije ne moraju imati tijelo. U tom slučaju one se zovu čiste virtualne funkcije *(eng. Pure virtual functions).* Klasa u kojoj se nalazi barem jedna takva funkcija se naziva apstraktna klasa. Čistoj virtualnoj funkciji se na kraju deklaracije dodjeljuje

vrijednost 0. Tada prevoditelj zna da ona neće imati tijelo, a time i klasa u kojoj se ona nalazi automatski postaje apstraktna. ***Interface*** jest apstraktna klasa samo sa čisto virtualnim funkcijama odnosno čisto apstraktna klasa.

Primjer:

```
class A {
public:
        virtual void f() = 0;
};
```

Što se tiče apstraktnih klasa u Pythonu, Python ne pruža apstraktne klase po defaultu, ali sadrži modul koji pruža kreiranje apstraktnih klasa. Modul se zove ***Apstract Base Class(ABC).*** Sintaksa kojom se neka klasa proglašava apstraktnom klasom je:

class ImeKlase(ABC):

Primjer:

```
from abc import ABC, abstractmethod
class myClass(ABC):
@abstractmethod
///
```

4. Polimorfizam

Polimorfizam ili jednostavno „moć poprimanja više oblika“ je osobina po kojoj metoda ima parametre različitih tipova. Npr. Dodaj() metoda može imati brojčane ili tekstualne parametre.

U C++ polimorfizam znači da će poziv funkcije članice izazvati pozive različitih funkcija u zavisnosti od tipa objekta koji je pozvao funkciju. Kod C++ i u polimorfizmu je neophodno korištenje *virtual* funkcija. Također je pravilo da se uvijek kreira virtualni destruktor kada radimo sa nasljeđivanjem i polimorfizmom.

U pythonu se polimorfizam koristi kada se metode isto nazivaju u dvije ili više klasa . Omogućuje korištenje objekata različitih vrsta i u različitom vremenu.

4.1 Generici i template

Generičko programiranje *(eng. Generic programming) je* stil računalnog programiranja u kojem su algoritmi pisani na način da predstavljaju „kostur“ koji je

impelentiran tek kasnije, kada se odrede argumenti. Ovaj pristup, koji je ML osnovao, omogućuje pisanje zajedničkih funkcija ili klasa koje se razlikuju samo u vrsti argumenata na kojima djeluju kada se koriste, čime se smanjuje dupliciranje.

C++ koristi predloške za omogućavanje generičkih tehnika programiranja. Generici se u C++ implementiraju pomoću ***templates. Template*** je jednostavan, a jako koristan alat u C++. Jednostavna ideja je proslijediti tip podataka kao parametar, kako se ne bi pisao isti kod za različite tipove podataka.

Primjer:

```
#include <iostream>

using namespace std;

template <typename T>

T max(T x, T y)
{
    return (x > y) ? x : y;
}

int main()
{
    cout << max <int>(1, 2) << endl; // poziv funkcije max za int
    cout << max <double>(1.3, 2.23) << endl; // poziv funkcije max za double
    cout << max <char>('a', 's') << endl; // poziv funkcije za max za char


}
```

U primjeru je prikazan princip pisanja i rada template kod poziva funkcije. Budući da je C++ objektno orijentiran jezik , template se uglavnom koristi kod klasa koje se i nazivaju „template klase“. Mogu biti korisne za klase poput LinkedList, BinaryTree, Array , Stack itd.

Primjer template klase:

```
#include <iostream>
using namespace std;

template <typename T>
class Array {
private:
    T *ptr;
    int size;
public:
    Array(T arr[], int s);
    void print();
};

template <typename T>
Array<T>::Array(T arr[], int s) {
    ptr = new T[s];
    size = s;
    for(int i = 0; i < size; i++)
```

```
        ptr[i] = arr[i];
}

template <typename T>
void Array<T>::print() {
    for (int i = 0; i < size; i++)
        cout<<" "<<*(ptr + i);
    cout<<endl;
}

int main() {
    int arr[5] = {1, 2, 3, 4, 5};
    Array<int> a(arr, 5);
    a.print();
    return 0;
}
```

Python nije statički pisan jezik kao C++ , i zbog toga nema koncept *generics*. Moglo bi se reći da je kod njega svaka funkcija generička jer parametri nemaju tip. Međutim, python može koristiti „duck typing“(if it looks like a duck, walks like a duck, and quacks like a duck, then it's a duck) to je metoda koja radi na sličan način kao i templates u C++.

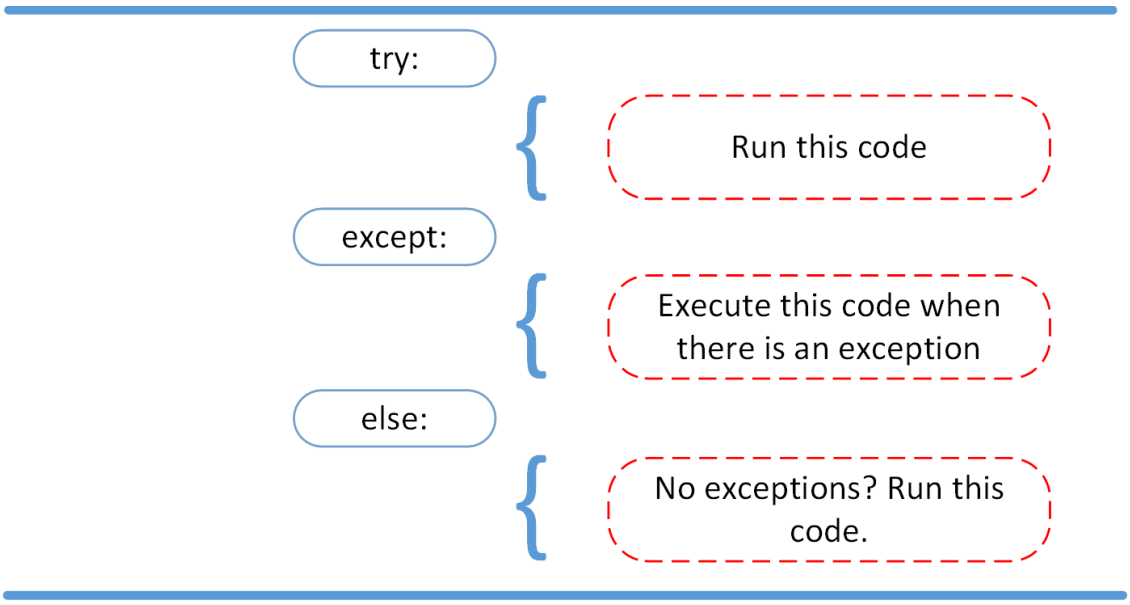
5. Iznimke

Iznimka je situacija kada se tijekom rada programa zbog nepredviđenih okolnosti dogodi greška. Takav tip greške može uzrokovati prekid rada programa, pa je od velike važnosti znati kako predvidjeti takve situacije i kako obrađivati iznimke. U tu svrhu C++ nudi mehanizam rukovanja iznimkama *(eng. exception handling),* pomoću ključnih riječi *try*, *throw* i *catch*. U bloku pokušaja (try) se navode problematične operacije koje bi mogle uzrokovati iznimku. Sama iznimka se baca (generira) ključnom riječi *throw,* a u ovisnosti o tipu iznimke izvršava se odgovarajući blok hvatanja. Ukoliko se bačena iznimka nije obradila u niti jednom prethodnom bloku hvatanja onda će se obraditi u bloku hvatanja koji kao parametre ima 3 točke – catch(…).

```
try{
 // blok pokušaja - problematični dio koda
        if (problem)
                throw Iznimka; // baci iznimku
 }
catch (parametar){
// blok hvatanja - obrada iznimke određenog tipa
 }
catch (...){
```

```
// obrada ostalih tipova iznimki
}
```

Kod pythona je iznimke moguće također obraditi na jednostavan način pomoću ugrađenih mehanizama. BaseExeption. BaseExeption je bazni razred za sve iznimke, dok je Exeption bazni razred za iznimke koje ne uzrokuju izlaz iz programa. Kao i u C++ , postoji rukovanje iznimkama *(exception handling)* sa naredbom *try .. except* . Unutar tijela naredbe try se izvode neki kod, te ako se pojavi iznimka daljnje izvođenje se prekida i kreće se izvoditi tijelo naredbe except. Python ima i završne blokove*: else i finally*. Blok else se izvodi ukoliko se unutar try ne dogodi iznimka ili se on ne završi pozivom naredbe return,break ili continue. Kod koji se nalazi unutar bloka finally uvijek će se izvršavati prije završetka obrade iznimke (bez obzira da li se iznimka dogodila ili ne). Obično se koristi za otpuštanje zauzetih resursa.



6. RAII

Resource Acquisition Is Initialization ili ***RAII*** je jedna od najkorisnijih metoda u C++, gdje konstruktor klase stvara resurs (npr. Otvaranje datoteke) , a odgovarajući destruktor oslobađa resurs.To je tehnika programiranja u C++ koja omogućuje programerima da osiguraju oslobađanje resursa pri napuštanju trenurnog „područja“ u kojem djeluju. Kada program napusti područje bez obzira na sve, resurs se oslobađa. Najčešće se koristi u radu sa mutexom i datotekama.

U pythonu ne postoji tehnika RAII. Na prvi pogled moglo bi se reći da se RAII u pythonu može postići putem __init__ i __del__ međutim to je pogrešno. U pythonu postoji metoda koja je slična RAII-u i obavlja sličan posao, zove se Context managers. Slično kao i

RAII context managers stvaraju resurs u nekom području i oslobađaju ga pri napuštanju. Lako je stvoriti context manager , korištenjem metoda __enter__ i __exit__ ili koristeći contextmanager uređivača.

7. Zaključak

Proveli smo neko vrijeme čitajući i upoznavajući se s razlikama C++ i Pythona. Iako Python ima lakšu sintaksu, nije savršen za sve probleme. Pogledali smo upravljanje memorijom, oslobađanje memorije, nasljeđivanje i druge aspekte ova dva jezika.

Dok C++ koristi ručno upravljanje memorijom, Python ima automatsko, te Python za oslobađanje memorije koristi Garbage collector,a C++ destruktore. Budući da je nasljeđivanje jedno od glavnih svojstava objektno orijentiranih jezika, nema velike razlike između ova dva jezika. Također, bitno je naglasiti da se u C++ mogu koristiti apstraktne klase, dok Python ne pruža apstraktne klase po defaultu. Još jedna velika bitna razlika između Pythona i C++ je ta što Python nije statički pisani jezik, te zbog tog nema koncept generics. Što se tiče iznimki u Pythonu i C++ rade na vrlo sličan način. Također u C++ se korisiti metoda RAII, dok u Pythonu se ne koristi što smo naveli prethodno.

8. Literatura

https://www.learncpp.com/cpp-tutorial/variable-shadowing-name-hiding/

https://bib.irb.hr/datoteka/979189.Objektno_orijentirano_programiranje.pdf

https://hr.edu-base.org/7780754-python-multiple-inheritance

https://www.geeksforgeeks.org/abstract-classes-in-python/

https://www.geeksforgeeks.org/polymorphism-in-python/

https:/https://www.geeksforgeeks.org/templates-cpp/

https://stackoverflow.com/questions/6725868/generics-templates-in-python

https://en.wikipedia.org/wiki/Duck_typing

https://dev.to/fronkan/comparing-c-raii-and-python-context-managers-50eg

https://en.cppreference.com/w/cpp/language/raii