Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

TDT4102 Prosedyre- og objektorientert programmering Vår 2019

Øving 1 Python

Frist: 2019-01-18

Mål for denne øvingen:

Overføre programmeringskonsepter vi kjenner til fra Python til C++.

Generelle krav og anbefalinger:

- Bruk de eksakte navn og spesifikasjoner gitt i oppgaven.
- Vi anbefaler at du benytter et anerkjent utviklingsverktøy, et IDE (Integrated Development Environment) til å gjøre øvingene. Et IDE er en komplett pakke med de verktøyene vi trenger for å skrive, teste og kjøre programmer. Vi anbefaler Visual Studio Community 2017 for Windows og XCode for Mac. (Andre IDE kan benyttes, men da kan man i noe mindre grad forvente hjelp fra assistenter i faget da de fleste av dem enten bruker Visual Studio eller XCode). Et, muligens "gammelt", norsk ord for IDE er programmeringsomgivelse.
- 70% av øvingen må godkjennes for at den skal vurderes som bestått.
- Øvingen skal godkjennes av stud.ass. på sal.

Innledning

I denne øvingen skal vi demonstrere likheter og ulikheter mellom C++ og Python. Oppgavene vil i stor grad gå ut på å oversette Python-kode til C++, og vil også i noen grad gi en slags oppfriskning av ITGK-kunnskapene. Hovedhensikten er å overføre kunnskaper om Python over til C++. Du vil også lære om forskjeller mellom de to språkene, og om noen fallgruver. Merk at Python-versjonen som brukes i eksempelene er Python 3.x, som benyttes i IT grunnkurs. Dersom du er vant med Python 2.7 vil det være visse forskjeller, blant annet fungerer divisjon ulikt.

Hvis det er noe kode du er usikker på når du leser de neste sidene vil vi anbefale å taste inn det du er usikker på og prøve koden ut i kompilatoren (IDE'en) du bruker.

Lynintroduksjon til forskjellene mellom Python og C++

Utskrift til skjerm

I Python brukes funksjonen `print(a,b,c)` for å skrive ut tekst til skjerm. I C++ sender vi i stedet tekststrenger og andre objekter vi ønsker å skrive til skjerm til objektet `cout` ("console output", også kalt "standard output stream") ved å bruke `<<`-operatoren. Det er tillatt å nøste den og dermed skrive ut flere objekter om gangen. Siste linje i figuren under gir et eksempel på det.

I Python brukes '`\n`' i `print` for å eksplisitt skrive ut et linjeskift. Dette er også mulig å gjøre i C++: '`\n`' inni en tekststreng som sendes til `cout` vil gi et linjeskift, på lik linje som i Python.

```python
print(2+3)
print("Hello world!")
print("2+5=", 2+5, sep='')
```

Python

```cpp
cout << 2+3 << '\n';
cout << "Hello world!\n";
cout << "2+5=" << (2+5) << '\n';
```

C++

Variabler

I Python kan man introdusere variabler bare ved å tilordne et variabelnavn en verdi. I C++ må vi første gang en variabel brukes deklarere den ved å skrive datatypen foran variabelnavnet. Det er vanlig å tilordne verdier til variabler samtidig som vi deklarerer dem (initialisering), men det er ikke påbudt. En del moderne programmeringsomgivelser gir deg advarsel om variable ikke initialiseres.

```python
a = 1
b = 2
c = a + b
d = c // b
print(d)
d = d * b
print(d)

e = c
f = e / 2;
print(f)

g = c / b
print(g)
```

Python

```cpp
int a = 1;
int b = 2;
int c = a + b;
int d = c / b;
cout << d << '\n';
d = d * b;
cout << d << '\n';

double e = c;
double f = e / 2.0;
cout << f << '\n';

double g = c / static_cast<double>(b);
cout << g << '\n';
```

C++

Øverst i Python-eksempelet ser vi at tallene 1 og 2 henholdsvis tilordnes variablene `a` og `b`. Både 1 og 2 er skrevet som heltall (ingen komma), og `a` og `b` vil derfor automatisk bli tolket som heltall av Python. I C++ kreves det at vi eksplisitt spesifiserer typen til variabelen i koden.

Vi har i dette eksempelet sett på to forskjellige datatyper, `int` og `double`. `int` er en type som kun kan representere heltall. En variabel av typen `int` opptar ofte (men ikke alltid) 32 bits i minnet, og kan dermed representere heltall mellom -2^{31} og $2^{31}-1$. `double` er en type som inneholder *flyttall*, som brukes til å representere reelle tall. Ulikt en variabel av typen `int` er en variabel av typen `double` vanligvis 64 bits stor, og har en presisjon på mellom 15 og 17 sifre avhengig av størrelsen på tallet. En ting som er verdt å merke seg er at flyttall ikke kan representere *alle* reelle tall (da det er et uendelig antall av dem i alle intervaller). (En må derfor være klare over at bruk av flyttall normalt vil føre til

større eller mindre «avrundingsfeil». Hvis en har to ulike flyttallsberegninger som rent matematisk skal gi eksakt samme verdi, så kan de på en datamaskin ofte bli ulike. Når en tester på likhet av flyttalsverdier kan en derfor være nødt til å legge inn en feilmargin.)

På linje 4 ser vi forskjellen mellom divisjonsoperatorene i Python og C++. I C++ gir den vanlige divisjonsoperatoren `/` *heltallsdivisjon* når den blir brukt til å dele to heltall. Dette betyr at svaret av divisjonen blir rundet ned til nærmeste heltall (hvis svaret er positivt), eller opp til nærmeste heltall dersom svaret er negativt (altså alltid "mot null"). Dette tilsvarer operatoren `//` i Python. Dersom man ønsker å få et desimaltall (flyttall) som svar fra en divisjon mellom to heltall, er man nødt til å *caste* det ene tallet til en flyttallstype, f.eks. `double`, og så utføre divisjonen. Å caste en variabel betyr å gjøre den om til en annen type. Caster man for eksempel heltallet 4 til `double`, vil man få flyttallet 4.0. Caster man et flyttall, for eksempel 4.5 til `int`, vil man få heltallet 4. Et eksempel på casting er vist til slutt i figuren over. Det finnes ulike former for casting, men foreløpig trenger du bare å bry deg om `static_cast` som er en eksplisitt måte å be om en type-konvertering. Det er nevnt i læreboka §8.5.7 og også dekket av forelesning.

Input fra bruker

Python kan ta inn data fra brukeren ved hjelp av `input()`. I C++ bruker man i stedet objektet `cin` ("console input", også kalt "standard input stream"), som fungerer på en lignende måte som utskrift til skjerm med `cout`.

```python
i = input("Skriv inn et tall: ")
j = input("Skriv inn et tall: ")
print("Summen av de to tallene: ", float(i)+float(j))
```

Python

```cpp
double i = 0.0;
double j = 0.0;
cout << "Skriv inn et tall: ";
cin >> i;
cout << "Skriv inn et tall: ";
cin >> j;
cout << "Summen av de to tallene: " << (i+j) << '\n';
```

C++

Python lar oss skrive ut en forespørsel til bruker om hva som skal skrives inn, mens vi i C++ vil måtte gjøre dette i to steg, først en utskrift, og så en forespørsel om input. Ved bruk av `>>`-operatoren tar C++ seg av å lese inn og tolke verdiens type. Dette fungerer slik at verdien som leses inn vil bli tolket som å ha samme type som variabelen den skal lagres i, altså `double` i eksempelet over.

If-setninger

En `if`-test ser slik ut i Python og C++:

```python
b = 2
if b > 2:
    print ("B is greater than 2")
else:
    print ("B is less than or equal to 2")
```

Python

```cpp
int b = 2;
if (b > 2) {
    cout << "B is greater than 2\n";
} else {
    cout << "B is less than or equal to 2\n";
}
```

C++

Store likhetstrekk i syntaksen og kontrollstrukturen fungerer som forventet.

For-løkker

En enkel `for`-løkke som kjører fra 1 til 10 ser slik ut i Python og C++:

```python
for i in range(1, 10+1):
    print(i)
```

Python

```cpp
for (int i = 1; i < 10+1; ++i) {
    cout << i << '\n';
}
```

C++

Som du legger merke til så ser `for` i C++ litt annerledes ut, men vi finner igjen 1 og 10 som grenser også her. At vi har en steglengde på 1 er litt mindre intuitivt. Dette kommer av den siste delen av argumentet til `for`-løkken, `++i`, som inkrementerer verdien av variabelen `i`. I eksempelet under brukes steglengde 2 i stedet, noe som fører til at kun oddetallene skrives ut:

```python
for i in range(1,10+1,2):
    print(i)
```

Python

```cpp
for (int i = 1; i < 10+1; i = i + 2) {
    cout << i << '\n';
}
```

C++

Uttrykk på formen `i = i + x` skrives ofte som `i += x` i C++. I vårt tilfelle kunne vi altså i stedet skrevet `i += 2`. Legg også merke til at vi uttrykker lengden på for-løkka vår som en sammenligning, `i < 10+1`. Dette fungerer slik at `for`-løkken fortsetter så lenge denne sammenligningen er sann. Her kunne vi skrevet `i <= 10` i stedet for `i < 10+1`. Det er i C++ normalt å starte løkkene på 0 og dermed bruke `<` i stedet for `<=`, blant annet fordi indeksen til vectorer (og tabeller) i C++ starter på 0, på samme måte som i Python.

While-løkker

```python
i = 1
while i < 1000:
    i *= 2
    print(i)
```

Python

```cpp
int i = 1;
while (i < 1000) {
    i *= 2;
    cout << i << '\n';
}
```

C++

Igjen er det kun mindre forskjeller i syntaks mellom en `while`-løkke i Python og en i C++.

Funksjoner

```python
def get_four(seed):
    return 4
```

Python

```cpp
int getFour(int seed) {
    return 4;
}
```

C++

I Python defineres funksjoner ved å skrive `def funksjonsnavn(parametre)`. Dette er ulikt måten vi definerer funksjoner på i C++, der vi også må ha med *typen til funksjonens returverdi*. Hvis en funksjon ikke har noen retur-verdi deklarer vi retur-"typen" som `void`. Som når variabler vanligvis defineres må funksjonens argumenter defineres med typer. I dette tilfellet ser vi at funksjonen tar inn et heltall, `int`.

Eksponensierings-operatoren

I Python bruker vi operatoren `**` for å eksponensiere ("opphøye") et tall. I C++ finnes ikke denne, men vi har to andre alternativer. For enkle uttrykk, der eksponenten er et lite heltall, kan vi eksplisitt multiplisere tallet vi vil eksponensiere med seg selv, f.eks. $x^3 = x * x * x$. Alternativt kan vi benytte funksjonen `pow`.

```cpp
#include "std_lib_facilities.h"

int main() {
    int fourSquared = pow(4,2);
    int fourCubed = pow(4,3);
    cout << "4^2: " << fourSquared
         << " 4^3: " << fourCubed << '\n';
    return 0;
}
```

Eksempel på eksponensiering ved hjelp av `pow` i C++.

I dette eksempelet inkluderer vi filen `std_lib_facilities.h`. **Du får tilgang til denne ved å følge fremgangsmåten beskrevet i øving 0, og å velge prosjekt-malen TDT4102_std_lib_facilities når du oppretter ett nytt prosjekt**. En slik fil kalles ofte en include-fil eller en header-fil. Denne filen benyttes i mange av øvingene for å redusere mengden av detaljer en må huske på å ha på plass i programmet for å få til en øving. I dette tilfellet gir headerfilen oss tilgang til funksjonen `pow`. `pow(x,n)` er en innebygget funksjon i biblioteket som beregner og returnerer verdien x^n (Se evt. læreboka §15.5 side 532).

`int main()` er en funksjon vi definerer én gang for hvert program vi skriver. Det kalles ofte hoved-programmet eller hoved-funksjonen, og er den delen av programmet som først utføres. Normalt vil den kalle opp en rekke andre funksjoner for å utføre programmets ulike "gjøremål".

Uttrykket med `cout` går over to linjer. Dersom en linje ikke ender med semikolon, vil kompilatoren forsøke å tolke neste linje som en fortsettelse av program-setningen (her et uttrykk). Ofte separerer man logisk separate deler av lange program-setninger på denne måten, slik at koden blir lettere å lese og linjene ikke blir for lange.

1 Kodeforståelse: oversett til Python(10%)

a) Oversett følgende kodesnutt til Python.

```
bool isFibonacciNumber(int n) {
        int a = 0;
        int b = 1;
        while (b < n) {
                int temp = b;
                b += a;
                a = temp;
        }
        return b == n;
}
```

Nyttig å vite: Organisering og kjøring av kode

I de fleste øvingene i dette faget er det ikke nødvendig med mer enn **ett prosjekt per øving** dersom du bruker en IDE (f.eks. Xcode eller Visual Studio). Der du trenger mer enn ett prosjekt vil dette stå i øvingsteksten. I denne øvingen trenger du bare en *kildefil* (`.cpp`-fil) i prosjektet. Denne skal inneholde alle funksjonene i Oppgave 2. For denne øvingen bør du opprette et *Konsoll-prosjekt med* ***`std_lib_facilities.h`***, se Oppgave 1.3 punkt 2 i øving 0. Har du fullført øving 0 skal du nå ha en ferdig prosjekt-mal for et slikt prosjekt, som gjør det raskt og enkelt å opprette et prosjekt av denne typen.

For at programmer ikke skal bli uoversiktlige deler man dem opp i funksjoner, som inneholder gjenbrukbar kode. Denne koden kan bli kjørt fra andre steder i programmet ved hjelp av et funksjonskall. (Alle programmer har minst en funksjon. Når programmet startes vil operativsystemet kjøre funksjonen som heter `main`.)

For å teste funksjonene du har laget kan du kalle disse fra `main`, som i eksempelet under.

```
#include "std_lib_facilities.h"

// funksjonen 'add' som har to parameter av
// typen 'int' og returverdi av typen 'int'.
int add(int a, int b) {
        // legg sammen verdiene av 'a' og 'b' og returner resultatet
        return a + b;
}

int main() {
        // Skriv 'Hello world!' i konsollen
        cout << "Hello world!\n";

        // Kall på 'add' med heltallene 1 og 2 som argumenter.
        // Returverdien (3) skrives ut i konsollen
        cout << add(1, 2) << '\n';
}
```

2 Oversett fra Python til C++ (90%)

Oversett følgende kodesnutter til C++, og sjekk at de både kompilerer og kjører i ditt IDE.

a) Største av to tall

```python
def maxOfTwo(a, b):
    if a > b:
        print("A is greater than B")
        return a
    else:
        print("B is greater than or equal to A")
        return b
```

b) `main`-funksjonen

- Lag en `main`-funksjon som inneholder følgende kodesnutt:
  ```
  cout << "Oppgave a)\n";
  cout << maxOfTwo(5, 6) << '\n';
  ```
- Sørg for at koden kompilerer, og gir forventet output.
- For alle de resterende deloppgavene, lag tilsvarende testkode i `main` etter at du har oversatt funksjonen(e) og sørg for at du får forventet output.

c) Fibonacci-rekker

```python
def fibonacci(n):
    a = 0
    b = 1
    print("Fibonacci numbers:")
    for x in range(1, n + 1):
        print(x, b)
        temp = b
        b += a
        a = temp
    print("----")
    return b
```

Husk å teste funksjonen!

d) Sum av kvadrerte tall

```python
def squareNumberSum(n):
    totalSum = 0
    for i in range(n):
        totalSum += i * i
        print(i * i)
    print(totalSum)
    return totalSum
```

e) Trekanttall[1]

```
def triangleNumbersBelow(n):
    acc = 1
    num = 2
    print("Triangle numbers below ", n, ":", sep="")
    while acc < n:
            print(acc)
            acc += num
            num += 1
    print()
```

f) Primtall 1

```
def isPrime(n):
    for j in range(2,n):
        if n%j == 0:
            return False
    return True
```

g) Primtall 2

```
def naivePrimeNumberSearch(n):
    for number in range(2, n):
        if isPrime(number):
            print(number, "is a prime")
```

h) Største nevner mindre enn tallet selv

```
def findGreatestDivisor(n):
    for divisor in range(n-1,0,-1):
        if n%divisor == 0:
            return divisor
```

Har du husket å teste alle funksjonene? Vi vil også nevne at du generelt, og i enda større grad i de kommende øvinger **bør teste koden din etterhvert som du skriver kode**. Det er mye lettere å finne feil i et lite program enn et stort, og stegvis utvidelse av et program er en anerkjent arbeidsmåte.

[1] https://no.wikipedia.org/wiki/Trekanttall