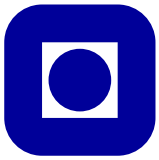

Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

TDT4102 Prosedyre og Objektorientert programmering Vår 2014

Øving 0.5-Python

Frist: 2014-01-17

Mål for denne øvinga:

- Overføre programmeringskonsepter vi kan fra Python til C++

Generelle krav:

- Bruk de eksakte navn og spesifikasjoner gitt i oppgaven.
- Det er valgfritt om du vil bruke en IDE (Visual Studio, XCode), men koden må være enkel å lese, kompilere og kjøre.

Denne øvinga er tenkt som ekstra hjelp til de av dere som har noe programmeringserfaring fra før men fra et annet språk en C++. Gjennom enkle eksempler ønsker vi å vise likhetstrekkene mellom det du kan fra før og C++. Øvingstempoet i faget blir ofte beskrevet som ganske høyt og denne øvinga kan være med på å gjøre oppstarten noe lettere og læringskurven mindre bratt.

I denne øvinga 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 kan kanskje være til hjelp som en slags oppfriskning av ITGK-kunnskapene, samtidig som de gir en mulighet til å overføre kunnskapen man har fra Python over til C++. Det vil også være viktig å legge merke til de fallgruvene og forskjellene som finnes mellom de to språkene.

Lynoppfriskning i Python, og C++ utgaven av tilsvarende kode

Generelt rammeverk

Siden vi ikke har gått så mye i dybden på C++ ennå, er det verdt å nevne enkelte ting som vi som regel trenger for å ha et gyldig program i C++. Et (nesten) minimalt C++-program, som ikke gjør noe som helst ser slik ut:

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6         // Kode her
7         return 0;
8 }
```

En kort gjennomgang av hva som foregår her:

- Linje 1 bruker vi til å si at vi trenger `iostream`-biblioteket, som inneholder ting utskift til skjerm, og input fra tastatur, andre interessante bibliotek er f.eks. `cmath`, som inneholder sinus/cosinus/tanges, og kvadratrot/potensfunksjoner.
- Linje 3 angir at vi ønsker å bruke disse funksjonene uten å måtte skrive `std::` foran dem hver gang, denne linja vil du ha i (nesten) alle .cpp-filer.
- Linje 5 definerer `main`-funksjonen, det er her programmet ditt starter når du kjører det. Legg merke til krøllparentesene.
- Linje 6 inneholder her en kommentar. Disse starter med //, og sier at resten av linja skal ignoreres. Det er, som linja sier, typisk her man skriver koden sin.
- Linje 7 returnerer verdien 0 fra `main`, dette er konvensjon for å si at programmet fullførte uten feil.

Utskrift til skjerm

Python skriver ut til skjerm ved hjelp av «`print`». Analogt til dette bruker vi «`cout`» i C++ som følger:

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

(a) Python

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

(b) C++

Figur 1: Utskrift til skjerm

I Python kan man enkelt skrive «`print (a,b,c,...)`» i C++ bruker vi istedenfor «`print`» «`cout`» eller standard utstrøm. Hver enkel gjenstand man ønsker å skrive ut må sendes til «`cout`» ved hjelp av «<<»-operatoren. Det er tillatt å nøste denne / å skrive ut flere gjenstander av gangen slik som på siste linje i figuren. «`endl`» er definert på forhånd og symboliserer linjeskift, dersom man ikke ønsker linjeskift etter utskrift kan denne droppes.

Variabler

I Python kan man introdusere variabelnavn ved å bare tilordne dem en verdi, i C++ må vi først definere hvilken type disse variablene er:

```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)
```

(a) Python

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

double e = c;
double f = e / 2.0;
cout << f << endl;
```

(b) C++

Figur 2: Variabeldeklarasjon og bruk

I C++ må man alltid definere typen til variablen. Dette trengs ikke i Python da Python gjør det automatisk basert på hva som gis inn. Øverst i Python eksempelet ser vi at tallet 1 og 2 tilordnes variablene a og b respektivt. Både 1 og 2 er skrevet som heltall (ingen komma) og vil derfor tolkes av Python som heltall. C++ krever at vi spesifierer dette i koden slik at kompilatoren vet hvilken datatype som er nødvendig.

Input fra bruker

Python kan ta inn tekst ved hjelp av input(). I C++ bruker man isteden «`cin`», dette fungerer på en lignende måte som utskrift til skjerm.

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

(a) Python

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

(b) C++

Figur 3: Input fra bruker

I Python må vi konvertere inputdataen til flyttall før vi kan legge dem sammen. C++ vil ved bruk av "»" operatoren ta seg av å lese inn og tolke verdien til riktig type. Tolkningen vil da gjøres, om mulig, til den datatypen variablen som brukes til lagring er av.

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")
```

(a) Python

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

(b) C++

Figur 4: If-tester i Python og C++

De viktigste forskjellene for If-tester mellom Python og C++ er at Python ikke krever paranteser rundt det som skal testes og klammer ({}) brukes rundt koden som er i If-testen i C++.

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):
        print (i)
```

(a) Python

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

(b) C++

Figur 5: Løkker i Python og C++

"for" ser 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, men bakgrunnen for det ligger i "i++", som tilsvarer "i = i + 1", altså "øk i med 1". For et eksempel som skriver ut bare oddetallene (steglengde 2):

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

(a) Python

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

(b) C++

Figur 6: Løkker med steglengde i Python og C++

Som en notis: uttrykk på formen «`i = i + x`» skrives ofte som «`i += x`» i C++, i vårt tilfelle ville vi kunnet skrive «`i += 2`». Legg merke til at vi uttrykker lengden på for-løkka vår som en sammenligning «`i <= 10`». For ordens skyld er det normalt å starte løkkene på 0, og bruke «<» istedenfor «<=» i C++. Dette er blant annet fordi tabeller i C++ har første indeks på 0, akkurat som i Python!

While-løkker

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

(a) Python

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

(b) C++

Figur 7: While løkker i Python og C++

Igjen er det kun mindre syntatiske (skrivemåte) forskjeller mellom en while-loop i Python og en i C++. Det er likevel greit å merke seg at C++ her også krever at det er paranteser rundt utrykket som testes og at istedenfor indentering bruker C++ krøllparanteser.

Tabeller

```python
t = [0]*10
for i in range(0,10):
        t[i] = i
        print (t[i])
```

(a) Python

```cpp
int t[10];
for (int i = 0; i < 10; i++){
        t[i] = i;
        cout << t[i] << endl;
}
```

(b) C++

Figur 8: Tabeller i Python og C++

Tabeller opprettes litt anderledes i Python enn i C++. For å opprette en tabell i C++ er vi (for øyeblikket) avhengig av å vite den endelige størrelsen før programmet kompileres. Dette er fordi det må settes av nok minne til å holde tabellen. Når tabellen er opprettet vil den oppføre seg på tilsvarende måte som i Python. Se kode eksempel over for detaljert syntaks.

Funksjoner

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

(a) Python

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

(b) C++

Figur 9: Funksjoner i Python og C++

Python skriver funksjoner som "def funksjonsnavn", istedenfor "def" må vi i C++ definerer returtypen til funksjonen. Dette er variabel typen til returverdien fra funksjonen. I tillegg må alle argumenter ha en type, i dette tilfellet tar funksjonen et heltall.

1 Kodeforståelse / oversett til Python (10%.)

a) Oversett følgende kodesnutter til Python.

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

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

a) Fibonacci rekker

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

Figur 10: Pythonkode

b) Trekanttall

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

Figur 11: Pythonkode

```
def isTriangleNumber(number):
    acc = 1
    while (number > 0):
        number = number - acc
        acc = acc + 1
    if number == 0:
        return True
    else:
        return False
```

Figur 12: Pythonkode

c) Sum av kvadrerte tall

```
def squareNumberSum(n):
    totalSum = 0
    for i in range(n):
        totalSum += i*i
        i = i + 1
        print (i*i)
    print(totalSum)
    return (totalSum)
```

Figur 13: Pythonkode

d) Største av to tall

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

Figur 14: Pythonkode

e) Primtall 1

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

Figur 15: Pythonkode

f) Primtall 2

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

Figur 16: Pythonkode

g) Største fellesnevner

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

Figur 17: Pythonkode

h) Telling med lister

```
def compareListOfNumbers(l):
    r = [0]*3
    for i in l:
        if i < 0:
            r[0] += 1
        elif i == 0:
            r[1] += 1
        else:
            r[2] += 1
    print (r[0]," numbers were below zero")
    print (r[1], " number were zero")
    print (r[2], " numbers were greater than zero")
```

Figur 18: Pythonkode