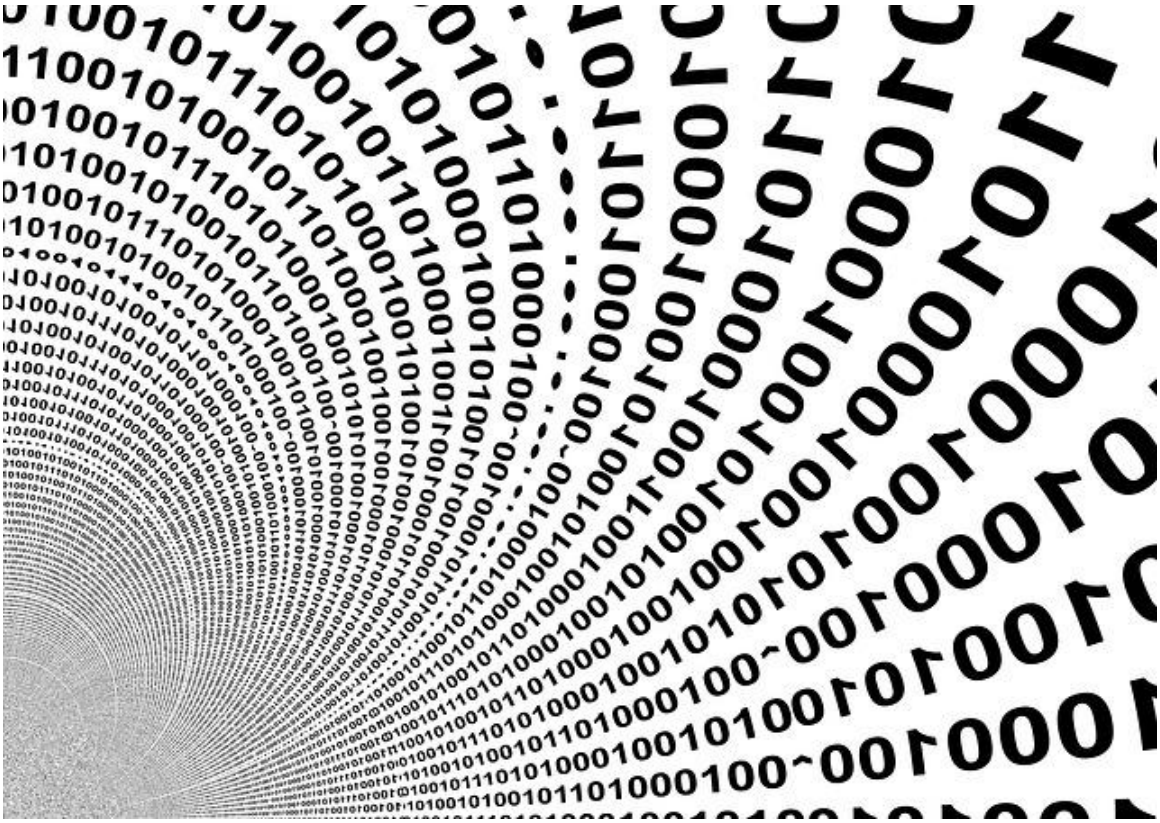
Python 3 Matematik Programmerings kursus:

Kompendiet indeholder:

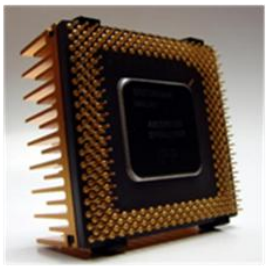
- Hello World (første program)
- Variable (String & Integer)
- Løkker (while-Loop)
- Regneoperationer
- If-else statement
- Funktioner
- Opgaver
  - Læg alle tal sammen
  - Fibonacci's tal
  - Kun lige tal
  - Et tals divisorer
  - Primtal
  - Problemløsning med programmering!


```
print("Hello, world!")
```

```
//Life motto
if(sad() == true){
   sad().stop();
   beAwesome();
}
```



Hvad de fleste computere har!


RAM:

Er korttidshukommelsen hvor resultatet af processorens beregninger bliver opbevaret!

Processor:

Er selve hjernen i computeren. Den kan foretage beregninger og operationer!

Harddisk:

Er langtidshukommelsen hvor processorens beregninger kan opbevares i længere tid!

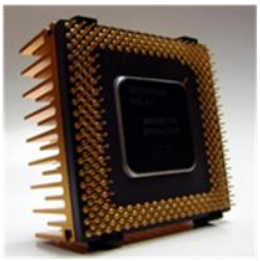
Et computer program!

Kode `print("hello world!")`

Compiler/Oversætter

Assembler/Maskinkode

```
.code
Start:
  push MB_ICONQUESTION + MB_APPLMODAL + MB_OK '
  push offset msgTitle
  push offset HelloWorld
  push 0
  call MessageBoxA
  push 0
  call ExitProcess
ends
end Start
```



Processor udfører programmet!

Handling

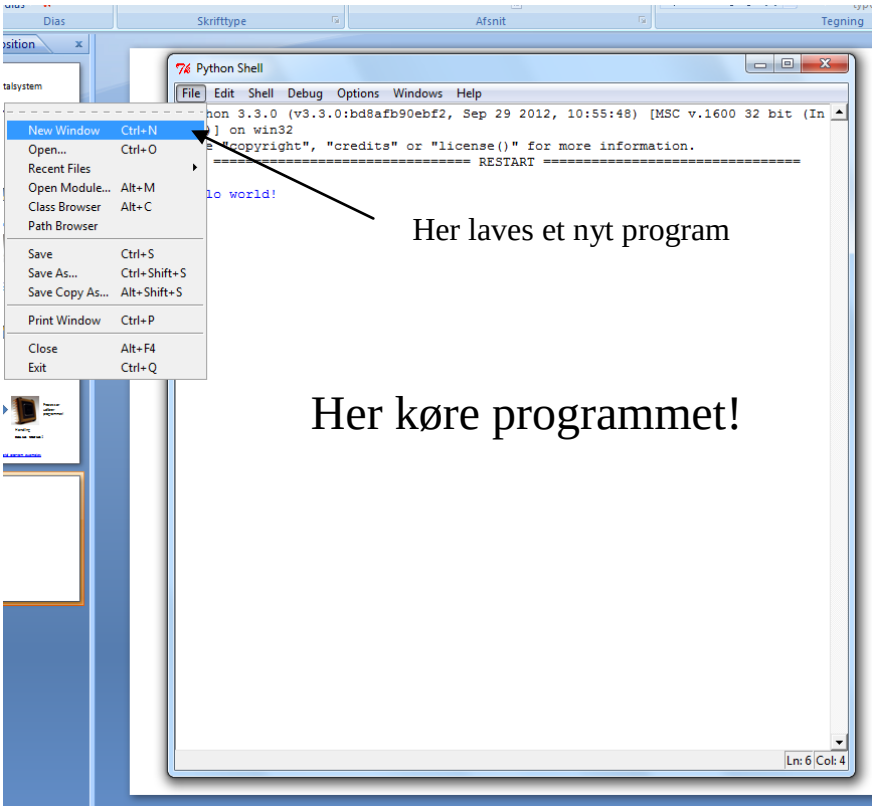
`hello world!`

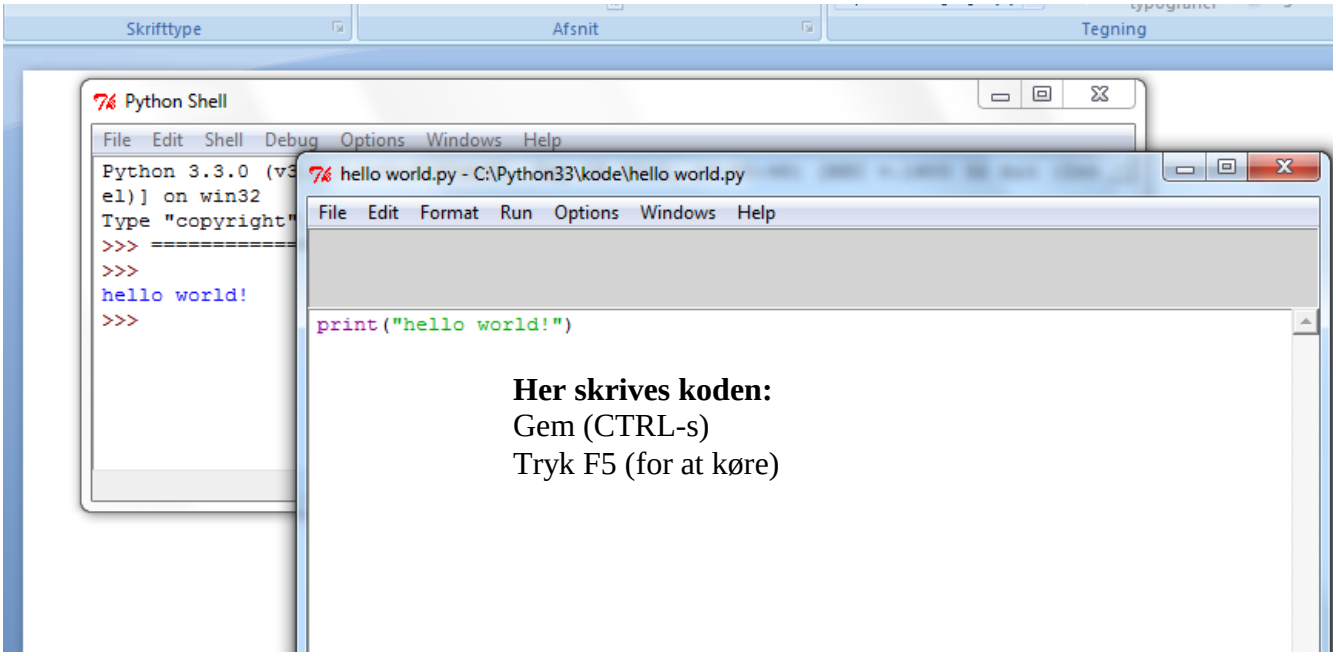
http://en.wikipedia.org/wiki/List_of_Hello_world_program_examples

Download og installer program: https://www.python.org/

Under download vælges passende version til ens styresytem - windows x86 MSI installer

Når programmet er startet laves nyt projekt ved File->New Window (se billede)





At skrive til skærmen: Hello World

Man kan skrive til skærmen ved at skrive følge i sin kode

```
print(”hello world”)
```

En funktion:

`Print` er det man kalder en funktion!

En funktion er et mini program der udfører en bestemt funktion! Denne funktion printer til skærmen en tekst! Teksten angiver man imellem ” ” (anførselstegnene)!

Funktionen skal jo have et navn så man kan genkende den og dette navn er `print`! Herudover får mange funktioner også det man kalder argumenter/værdier! Argumenterne anfører man bagefter funktionen i parentes

`FunktionNavn(argumenter/værdier)` **eks** `print(”tekst”)`

Når man kalder en funktion i Python (men også mange andre programmeringssprog) skal der være parenteser efter funktionens navn!

```
print(”den tekst der skal printes”)
```

Opgave 1: Prøv at lave et program der skriver Hello world. Erstat herefter teksten med andre sætninger! **HUSK:** tryk F5 når du har skrevet koden `print(”hello world”)`

Variable:

I ethvert program er der brug for værdier! Det kunne f.eks. være tal af forskellige slags eller sætninger. Disse værdier kan man bruge til f.eks. at regne med! Sådanne værdier kaldes for variable! I Python skal man inde i sin kode angive, hvilke variable man bruger - de skal erklæres så det er klart hvilke der bliver brugt! I det følgende erklæres en variabel som tekst streng:

En tekst variabel:

```
myString = ”hej med dig”
```

Et nummer/integer variabel.

```
myNumber = 1234
```

Regneoperationer:

```
+ addition
- subtraction
* multiplication
/ division
% modula (rest af division)
```

Opgave 2: Lav et program hvor du laver forskellige variable og printer dem! Prøv ligeledes, at oprette tal og læg dem sammen (+), gange (*) eller division (/)

Eks

```
number = 12 / 6
print(number)
```

Løkker også kaldet Loops:

I Pyton og i hvilket som helst andet programmeringssprog er loops vigtige og findes i forskellige former! Det vi skal se på nu hedder et `while` loop!

I et while loop kører loopet indtil udsagnet efter while viser sig at blive FALSE! Dvs. så længe udsagnet er sandt looper den!

Det betyder at man har mulighed for at lave loops der kører uendeligt! Lad os se på et!

```
while True:
    print(”hej med dig”)
```

HUSK :

Brug tab til indryk:

Vi kunne også have brugt False!

tabs

Opgave 3: Lav et uendeligt loop! Det kan stoppes ved at trykke CTRL-c. Lav herefter et loop som aldrig starter ved at bruge False!

While loop indtil 10!

Efter `while` kan man sætte en sammenligning ind! Det er oplagt at bruge tal her! Lad os derfor printe alle tal op til 10!

```
counter = 0

while counter < 10:
    #Vi udskriver tallet a!
    print(counter)
    #vi lægger 1 til tallet a
    counter = counter + 1
```

Dette er en kommentar der ikke kommer med i det endelige program! Alle linjer der starter med # er kommentarer

Denne kode gentages/er i loop indtil counter = n altså 10 = 10!

Opgave 4: Du skal lave et while loop som udskriver alle tal til og med 100!

En almindelig fejl - indrykning: Syntaksen er meget vigtig. Hvis man f.eks. ændrer loopet til

```
while counter < n:
    print(counter)
counter = counter +1
```

Det eneste der gentages er at counter udskrives hvilket den gør i det uenedelige da counter altid vil være 0 og dermed mindre end n

Vil den skrive 0 i det uendelige fordi så er det kun print(a) der er med i loopet!

Loop & matematik:

Inde i loopet kan vi f.eks. regne - lægge tal sammen!

```
counter = 0
resultat = 0

while counter <= 10:
    result = result + counter
    counter = counter + 1
print(result)
```

Dette program vil lægge alle tal sammen fra 0 til 10!

Opgave 6: Lav et program der lægger alle tal sammen fra 0 til 100!

NB: det skal give 5050 selvfølgelig!

Fibonnaci's tal:

```
    0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, osv.
```

Fibonnaci's tal genfindes i spiralmønstre i naturen f.eks. i solsikkeblomster, kogler & blomkålshoveder men er også at finde i det gyldnesnit!

Algoritme til at beregne Fibonnaci's tal:

En algoritme fortæller hvordan et problem løses! En måde at løse problemet er at beregne et nyt tal i rækken ved at lægge de forgående 2 tal sammen. Dette gøres bedst inde i et while loop:

1. Læg de 2 foregående tal sammen og gem resultatet i en variabel.
2. Print det nye tal ud!
3. Overskriv det ældste tal med det nyeste tal
4. Overskriv det nyeste tal med resultatet fra nr 1.

Opgave 7: Du skal konstruere et loop som kan beregne de første 1000 tal af fibonaci's talrække!

Ekstra Opgave: beregn phi ϕ der findes ved at dividere det nye tal med det forrige!

Hvis du ikke kunne løse opgave 7 er her et eksempel på hvordan det kunne gøres:

```
counter = 0

fibNumberOldest = 0
fibNumberNewest = 1
newFibNumber = 0

while counter < 10:
    newFibNumber = fibNumberOldest + fibNumberNewest
    print(newFibNumber)
    fibNumberOldest = fibNumberNewest
    fibNumberNewest = newFibNumber

    counter = counter + 1
```

Regneoperationen Modula %:

I den følgende opgave kommer vi til at bruge en regne operation som kan returnere resten af et divisions stykke!

12 % 2

Dette regnestykke vil returnere resten af divisionsstykket 12 / 2 hvor resten jo er 0!

13 % 2

Dette vil resultere i en rest på 1!

11 % 3

Giver 2 til rest!

Opgave 8: Lav et program som finder ud af hvad resten af af 24 / 5 og printer det ud.

NB: det skulle gerne være 4!

if else statement:

I nogle tilfælde er det nødvendigt at programmet man laver kan gå i 2 forskellige retninger! Man kunne f.eks. forestille sig et program der skal skrive noget specielt ved tallet 42! Dette gør man ved et if else statement!

Er number ligmed 42 bemærk == (2 ligmed tegn)

```
if number == 42:
    print(”svaret er 42”)
else:
    print(”det er ikke 42”)
```

Køres kun hvis alderen er lig eller over 17 år

Køres kun hvis alderen er under 17 år

Bemærk her at hver if eller else linje afsluttes af :

Bemærk ligeledes at print rykkes 2 ind eller en tab!

Sammenligning operatorer:

==	ligmed
!=	ikke ligmed
>	større end
>=	større end eller ligmed
<	mindre end
<=	mindre end eller ligmed

```
counter = 0

while counter <= 100:
    print(counter)
    counter = counter + 1
```

Opgave 9: Loop’et ovenfor udskriver alle tal fra 0 til 100! Ændre loop’et så det kun udskriver lige tal! **NB:** tal % 2 (modula) vil altid give 0 ved et lige tal og 1 ved et ulige tal!

```
counter = 1

while counter <= 10:
    rest = n % counter
    if rest == 0:
        print(counter)
    else:
        pass
    counter = counter + 1
```

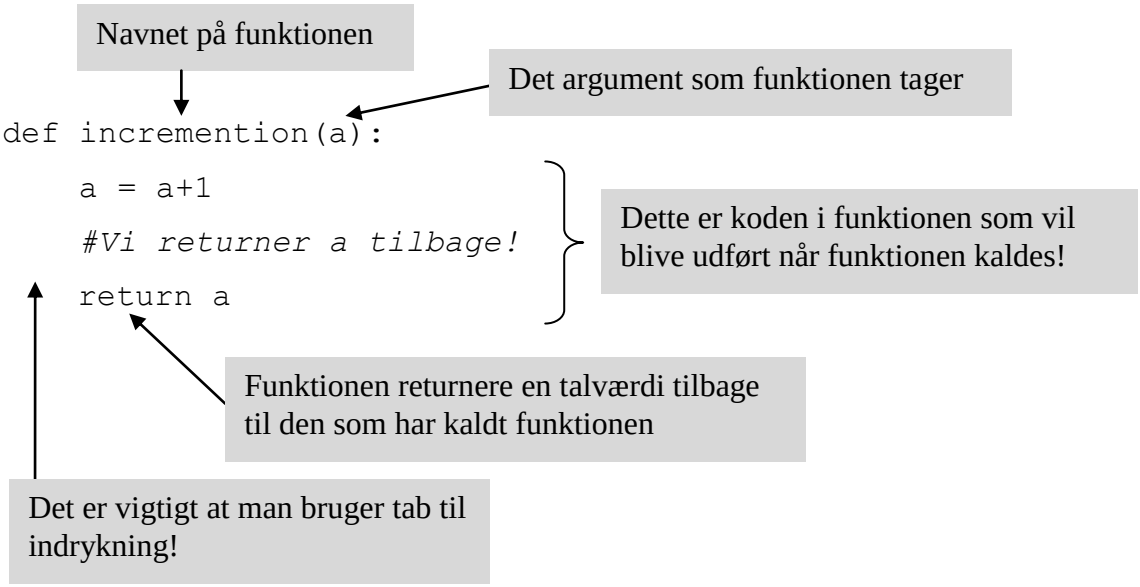
Program sendes videre med koden: `pass`

Ekstra Opgave: Undersøg hvad koden gør ovenfor? Skriv det evt. ind i pyton.

Hvad gør koden?_____

Funktioner:

Indtil videre har vi lavet programmer som er simple og køre fra A til B! Dog oplever man ofte at man bruger den samme kode om og om igen! Da vi jo er dovne ville det være smart at kunne samle denne kode som man så kan kalde om og om igen - uden at skulle skrive den hele tiden! Dette kaldes for en funktion! En funktion skal selvfølgelig have et navn - få overført nogle variable (kaldt argumenter) og kunne returnere en værdi! Lad os se på et eksempel:



Eksempel på brug af funktionen `incremention` i program!

```
counter = 0
result = 0

while counter < 101:
    print(result)

    counter = incremention(counter)
    result = result + counter
```

Her gemmes det tal som funktionen returnere i variablen a!

Her kaldes funktionen fra koden med argumentet a der er et tal som funktionen skal at arbejde med!

Opgave 10: Lav en funktion der lægger 1 til og brug den i et program!

Et primtal!

Et primtal er et tal hvor der kun er 2 divisorer 1 og tallet selv!

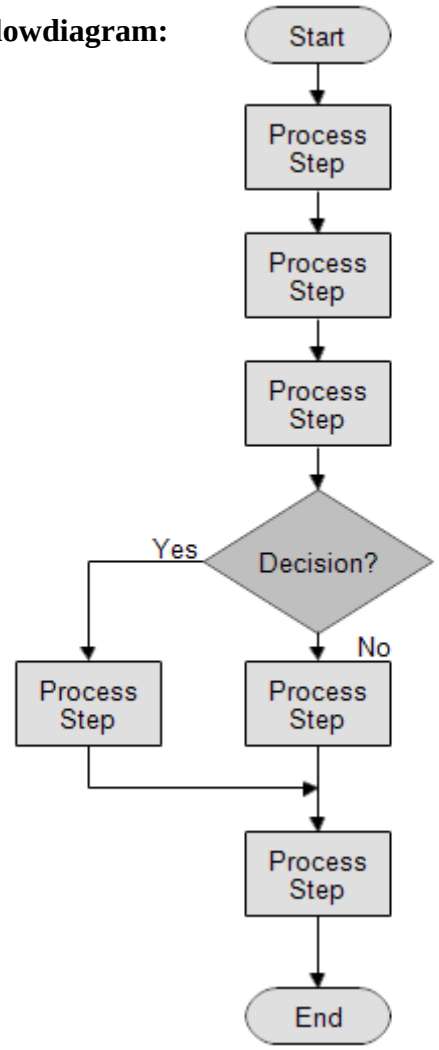
```
2  3  5  7  11  13  17  19  23  29  31  37  41  43  47  53  59  61
```

Opgave 11: Lav et program der udskriver alle primtal op til 1000. Du kan med fordel bruge følgende to funktioner!

```
def unEqualNumber(a):
    if a % 2 == 1:
        return True
    else:
        return False

def gotDivisors(n):
    i = 2
    while i < n:
        if n % i == 0:
            return True
        else:
            pass
        i = i+1
    return False
```

Flowdiagram:



Overvej algoritmen til at finde primtal!

Her kan et flowdiagram benyttes!

Brug programmering til at løse ligninger:

En computer er jo blot en stor regnemaskine så hvorfor ikke lade den slave for en! I det følgende skal vi se på, hvordan man vha. programmering kan finde løsningen til et problem som ellers ville kræve advanceret matematik (log), geogebra eller meget tid med en regnemaskine i hånden!

Løsning af eksponentiel funktion: Københavns befolknings tilvækst

Befolkningen i København regner man med vil vokser med 1,8 % pr år fra 2013 og frem. I 2013 var der 720.000 indbyggere og da væksten følger en eksponentiel forskrift kan dette stilles op således:

$f(x) = \text{startværdi} * \text{fremskrivningsfaktor}^{\text{år}}$

$f(x) = 720.000 * 1{,}018^{x}$

$$fremskrivningsfaktor = \frac{100\% + vækst\%}{100}$$

Spørgsmålet er hvor mange år der går før befolkningen når 1 mio mennesker?

Mange vil måske side og taste ind og prøve forskellige år som f.eks.

$f(1) = 720.000 * 1{,}018^{1} = 732.960$ (i år 2014)

$f(2) = 720.000 * 1{,}018^{2} = 746.153$ (i år 2015) osv.

Sådan kunne man så forsætte i lang tid indtil man ville nå over 1.000.000 indbyggere - eller dette:

```
counter = 0

while counter < 100:
    calc = 720000*1.018**counter

    if calc >= 1000000:
        print(calc)
        print(counter)
        break
    else:
        pass
    counter += 1
```

** betyder ^ opløftet i

I USA er det . i stedet for ,

Afbryd loop

Samme som `counter = counter + 1`

Opgave 12: Du skal vha. programmet finde ud af hvornår befolkningen i KBH runder

- 1 mio:
- 2 mio:

Opgave 13: Verdens befolkningen var i 2012 på 7,046 mia mennesker! Man regner med, at befolkningen vokser med 1,15 % pr år! Hvornår runder verden 10 mia. mennesker?

Følgende er et bud på et program der kan finde primtal!

```
#Dette program kan udskrive de første primtal op til 1000!
def unEqualNumber(a):
    if a % 2 == 1:
        return True
    else:
        return False

def gotDivisors(n):
    i = 2
    while i < n:
        if n % i == 0:
            return True
        else:
            pass
        i = i+1
    return False


counter = 2
result = 0

print(gotDivisors(7))
while counter < 1000:
    if unEqualNumber(counter):
        if gotDivisors(counter) == False:
            print(counter)
        else:
            pass
    else:
        pass

    counter = counter +1
```