Snelcursus Sage

1. Sage

Het programma **Sage** (in latere versies SageMath genoemd) is een computeralgebrasysteem dat in het jaar 2005 ontwikkeld werd door William Stein, wiskundige en professor aan de universiteit van Washington. Het is een opensourceprogramma dat een waardig tegengewicht wilde brengen voor commerciële wiskundeprogramma's zoals Maple, Mathematica en MATLAB. Pas in 2016 werd de eerste betaalde programmeur aangenomen om Sage verder te ontwikkelen. Momenteel wordt Sage ondersteund door een mix van vrijwilligers en betaalde krachten.

Vele universiteiten die gratis besturingssystemen en gratis software willen propageren, maken gebruik van Sage, zo ook de UGent. Sage is onderlegd in algebra, analyse, combinatoriek, getaltheorie, statistiek, numerieke analyse en grafentheorie. Maar Sage heeft ook enkele onverwachte talenten. Het bevat bijvoorbeeld een automatische sudokusolver.

Het programma Sage maakt gebruik van de sterkste kanten van verschillende andere wiskundige softwarepakketten zoals C++, Lips, Fortran en Python. Vooral Python is bekend als instapprogramma voor beginnende programmeurs.

SageMath (de opvolger van Sage) maakt gebruik HTML (HyperText Markup Language) om teksten op te maken. HTML is sinds lange tijd de standaardtaal op het wereldwijde web. Alle webbrowsers begrijpen het. De formules binnen de HTML-teksten worden gezet in LaTeX. Dit is momenteel de wereldwijde standaard voor wiskundige formules. Als je straks tijdens het programmeren opzoekingen wil doen, dan google je best op SageMath (eventueel ook op Sage of op Python).

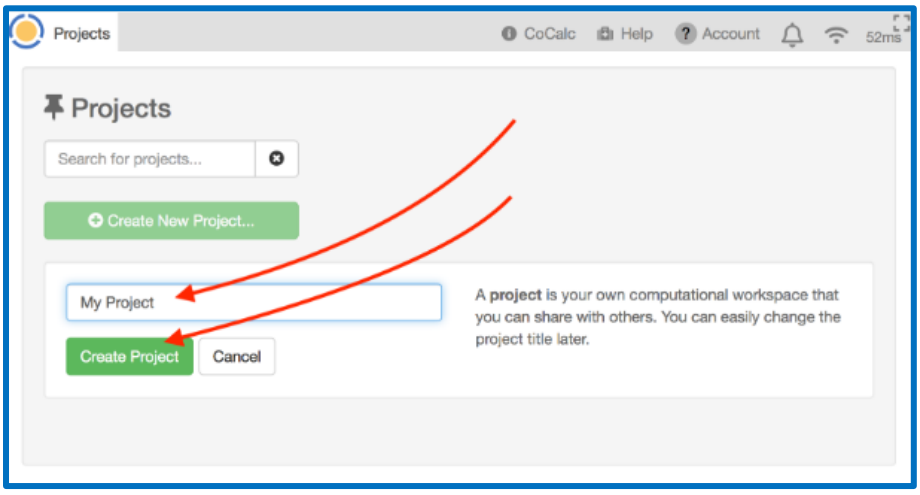
CoCalc is een online virtuele werkruimte waarin je berekeningen en onderzoek kan doen. Je kan er SageWorksheets (*.sagews) maken, die je in de cloud bewaart. Het enige wat je in dit project dus nodig hebt is een webbrowser en een internetverbinding. Geen gedoe met het installeren van software. Terwijl je een programma in CoCalc schrijft, kan je chatten met collaborators. Je hebt tijdens het schrijven van je verslag in Overleaf wellicht gemerkt dat dit handig kan zijn.

2. Een account maken en een programma opstarten

Ga naar de site van CoCalc en maak hier een nieuwe account aan. Vergeet niet je akkoord te geven voor de Terms of Service. Je kan intekenen via een van je mailadressen, bijvoorbeeld het toren.edugo.be-adres. Maar je kan je nieuwe account ook activeren via je Google-account, via

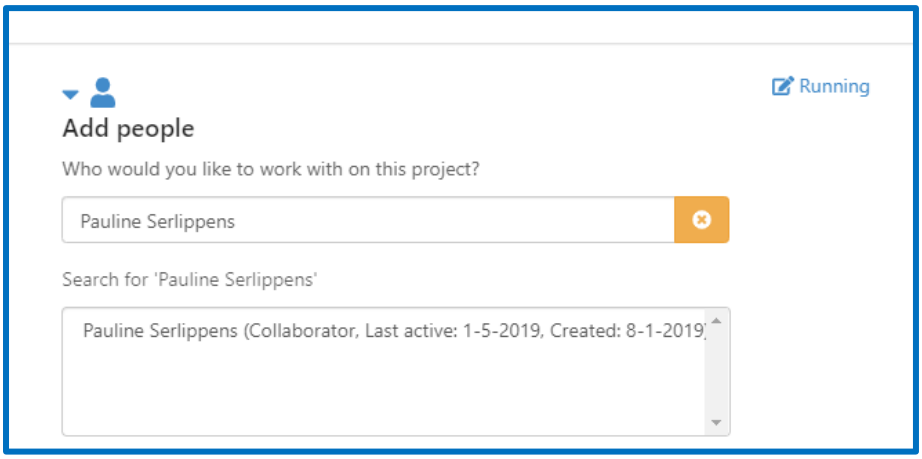
Facebook of via Twitter. Als je dit doet, hoef je geen nieuw wachtwoord meer te kiezen en te onthouden.

Als je ingelogd bent, kan je een nieuw project creëren. Kies als naam 'Corona'. We zullen immers een programma schrijven om voorspellingen te doen over een virusepidemie. Een project is een verzameling van verschillende bestanden, bijvoorbeeld Sage Worksheets, pdf-bestanden, LaTeX-documenten, Jupyter Noteboek-bestanden …



Binnen dit nieuwe project maak je een nieuw bestand aan door op ⊕ Create New Project te drukken. Kies voor Sage Worksheet. Noem het bestand 'Snelcursus'. Nu ben je bijna klaar om te starten met programmeren.

Wil je tot slot nog even je leerkrachten als collaborator aanduiden? Zo kunnen ze je online advies geven. Ga naar het tabblad projects linksboven, klik op het blauwe mannetje (add people) en vul hun email-adres in: chiaradevos@hotmail.com en vandenbroeck_luc@telenet.be .



3. Opdracht

Hieronder zie je een tiental blokjes programmatuur in Sage. Kopieer ze één voor één op een nieuwe regel van je programma.

Telkens wanneer je een blokje hebt ingevoerd, druk je op Run om het te laten lopen. Zolang CoCalc nadenkt over de opdracht, zie je groen horizontaal lijntje knipperen. Dat lijntje staan symbool voor werkende hersenen. Als CoCalc te lang moet nadenken, heb je wellicht een foutje gemaakt in de formulering van je opdracht. Je kan het denkproces dan onderbreken met Stop. Als je fout verbeterd is dan kan je op Restart drukken.

Na elke input in deze tekst volgt een tekstkader. Hierin kan je invullen waarvoor deze programmaregels dienen en wat je van dit stukje programmatuur moet onthouden.

We helpen je kritisch na te denken over de programmaregels door je vooraf een paar vragen te stellen. Aan de hand van deze vragen kan je wat experimenteren.

Deel 1: een centrale titel

Input:

%html   Snelcursus Sage

Vragen:

- Wat is de betekenis van %html?
- Wat betekenen de eerste vier tags tussen <>?

- Wat is de betekenis van de tags met de backslash?
- Waarom gebruiken we aanhalingstekens in CoCalc?

Verklaring:

Deel 2: verklarende tekst in HTML

Input:

%html   Teksten in Sage worden steeds in html (HyperText Markup Language) genoteerd, een taal die geschikt is voor het wereldwijde web en die kan gelezen worden door webbrowsers. Kenmerkend voor html zijn de markeringen (tags) met puntige haken. Sommige markeringen omarmen een tekstblok zoals en waarmee je stukjes tekst vet maakt. Andere markeringen staan alleen, zoals de markering
 waarmee je een regel afbreekt. Let er vooral op dat je in html voortdurend een voorwaartse slash gebruikt.

Vragen:

- Waarvan is
 de afkorting? En ?
- Wat gebeurt er als je weglaat?

Verklaring:

Deel 3: formules in verklarende tekst opnemen

Input:

%html <font size=4> <font color='darkblue'> Html ondersteunt ook formules in $\LaTeX$. Zo kan je de formule $\Sigma_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$ neerschrijven als je de n eerste kwadraten wil optellen. Let op: bij $\LaTeX$ gebruik je voortdurend de achterwaartse slash.

Vragen:

- Waarom wordt het woord LaTeX als een formule beschouwd?
- Wat gebeurt er als sigma met een kleine letter schrijft?
- Wat gebeurt er als je de tags vergeet te sluiten?

Verklaring:

Deel 4: enkele eenvoudige programmalijnen

Input:

```
a=4; b=-20; c=21
d=b^2-4*a*c
if d>0:
   print 'de oplossingen zijn', (-b+sqrt(d))/(2*a), 'en', (-b-sqrt(d))/(2*a)
elif d==0:
   print 'de oplossing is', (-b)/(2*a)
else:
   print 'er zijn geen oplossingen'
```

Vragen:

- Wat gebeurt er als je de puntkomma's in de eerste lijn weglaat? Waarom?
- Wat gebeurt er als je in de plaats van puntkomma's op de eerste lijn enters zet en je dus meerdere lijnen van dit commando maakt?
- Wat gebeurt er als je het dubbelpunt na het if-commando weglaat?
- Wat gebeurt er als je de inspringing weglaat de lijn na het if-commando? Zijn ze enkel esthetisch?
- Waarom staan er komma's in het print-commando?
- Waarom staat er twee maal een '=' bij het elif-commando?

Verklaring:

Deel 5: werken met lijsten

Input:

```
[i for i in range(10)]
[i for i in range(1,10)]
[i^2 for i in range(10)]
[numerical_approx(random(),digits=4) for i in range(10)]
[p for p in [1,2,..,100] if is_prime(p)]
sum([p for p in [1,2,..,100] if is_prime(p)])
```

Vragen:

- Waarvoor dienen vierkante haken?
- Wat is het eerste getal van range(10)? En het tiende getal van range(10)?
- Wat gebeurt er als je de haakjes na random() weglaat? Kan je dit vergelijken met de input op je zakrekenmachine?
- Wat betekent numerical_approx?

Verklaring:

Deel 6: grafieken van functies plotten

Input:

```
f(x) = 2^x - 2*cos(x)
g(x)=derivative(f,x)
print 'de functie f(x)=', f(x)
print 'de afgeleide functie g(x)=', g(x)
plot(f, (x, -2, 2), color='green',thickness=3)+plot(g,(x,-2,2), color='magenta',thickness=3)
```

Vragen:

- Wat is het verschil tussen een afgeleide die op je zakrekenmachine berekend wordt en een afgeleide die door Sage berekend wordt?
- Wat gebeurt er als je (x,-2,2) bij beide plots weglaat?
- Wat doet de '+' in de laatste lijn?

Verklaring:

Deel 7: een plot maken op basis van een lijst

Input:

```
A=[p for p in [1,2,..,100] if is_prime(p)]
len(A)
B=[(i,A[i]) for i in range(len(A))]
B
u=list_plot(B, plotjoined=True, color='orange',thickness=3)
u
```

Vragen:

- Kan je ook met de lijst A een grafiek plotten? Waarom wel/niet?
- Waarvoor staat de afkorting ‘len’?
- Waaraan is len(A) in dit programma gelijk?
- Hoe zorg je ervoor dat er geen ‘error’ komt als je plotjoined=False zet?
- Waarom staat er geen print voor ‘u’?

Verklaring:

Deel 8: een functiedefinitie met een while-loop

Input:

```
def klokrekenen(a,b):
   while a>b:
      a=a-b
   return(a)
klokrekenen(51,12)
```

Vragen:

- Wat gebeurt er als je return(a) binnen de while-lus zet, dus als je ze verder laat inspringen?
- Wat is dus het verschil tussen print en return?
- Voor welke wiskundige operatie dient de functie 'klokrekenen'?
- Wat gebeurt als bij de instructie klokrekenen(a, b) als a kleiner is dan b?

Verklaring:

Deel 9: een functiedefinitie met een for-loop

Input:

```
def fibo(n):
   F=[1 for i in [0..n]]
   for i in [2..n]:
      F[i]=F[i-2]+F[i-1]
   return(F)
fibo(20)
```

Vragen:

- Wat gebeurt er als je de tweede lijn weglaat? Waarom?
- Waarom kan je de parameter i niet laten lopen in [1..n]?
- Hoe heten de getallen uit fibo(20) officieel?

Verklaring:

Deel 10: een serie grafieken in één assenstelsel plotten

Input:

```
def kleurigeperioden(n):
   f=sin(x)
   v=plot([])
   for i in [0..n-1]:
      v=v+plot(f, (x, i*2*pi, (i+1)*2*pi), color=hue(random()),thickness=3)
   return(v)
kleurigeperioden(5)
```

Vragen:

- Waarvoor dient de derde lijn? Wat doet deze lijn?
- Wat is de betekenis van hue? En van hue(random())?
- Waarom staat 'hue(random())' niet tussen aanhalingstekens zoals wel al steeds het geval was bij kleuren?
- Waarom gebruiken we het interval [0..n-1] en niet [0..n]?

Verklaring:

Deel 11: grafieken maken met schuifknoppen

Input:

```
@interact
def paraboolmetschuifknoppen(a=slider(-10,10,1,2),b=slider(-5,5,1,1),c=slider(-5,5,1,-3)):
   f(x)=a*x^2+b*x+c
   v=plot(f,(x,-5,5), color='olive', thickness=3)
   show(v)
```

Vragen:

- Wat doet @interact?
- Wat betekent het woord ‘slider’?
- Waarvoor staan de vier getallen binnen de slider-functie?

Verklaring: