Übungsblatt 0

Willkommen zum Praktikum zu Programmieren in Python.

Zur Vorbereitung des Praktikums können Sie schon einiges vor oder gleich nach dem ersten Praktikum tun, insbesondere, wenn Sie mit Ihrem eigenen Rechner arbeiten wollen.

- Installieren für die Arbeit zu Hause: `InstallPython.pdf`
  - VirtualBox-Image: `pb19`, der einfache Weg
  - Einzelteile installieren: die Herausforderung
- Git-Account für die Veranstaltung abholen

Sie können in der Präsenz-Veranstaltung natürlich auch Ihren Laptop verwenden, aber die Pool-Rechner zu verwenden macht Sie viel flexibler, belohnt die Verwendung des Versionskontrollsystems und geht wahrscheinlich etwas schneller.

Aufgabe 1. Starten Sie Linux (empfohlen) oder Windows, loggen Sie sich ein und starten dann Visual Studio Code mit `code` auf der Konsole oder durch klicken. Konfigurieren Sie Ihren Editor `code` wie in `InstallPython.pdf` beschrieben. Erstellen Sie einen Ordner `pytest`. Öffnen Sie den Ordner und erstellen Sie eine neue Datei, die Sie unter dem Namen `hello.py` speichern mit folgendem Inhalt.

```
1 #!/usr/bin/python3
2 print("Hello Python World")
```

Wir führen das Programm durch Klick auf den Play-Button rechts oben (sie haben die Erweiterungen wie in `InstallPython.pdf` erwähnt installiert?) oder durch den Keyboard-Shortcut `Ctrl-Alt-N`. Sie sehen dann in der Konsole die Ausgabe

```
Hello Python World
```

Hinweis 1. Wenn Sie im Pool arbeiten, dann wird Ihre Umgebung beim neu Einloggen nach einem Neustart des Rechners immer zurückgesetzt. Das ist schlecht zum Arbeiten, da dann Einstellungen vom Browser, des Desktops, für Git, für die IDE etc. immer wieder weg sind. Das trifft uns auch hier mit Python, aber es gibt eine gute Abhilfe für Linux und das Notwendigste für Windows.

TL;DR: Im Pool unter Linux sollten Sie sich die Datei

```
$HOME/export/Oop/nethomedotprofile
```

in die Datei

```
$HOME/nethome/.profile
```

kopieren. Erstellen Sie auch einen Ordner `$HOME/nethome/.vscode` und noch dazu eine Datei `$HOME/nethome/.gitconfig` und loggen Sie sich nach einem Neustart neu ein. Dadurch erreichen Sie, dass sich unter anderem die Einstellungen zu Visual Studio Code und zu Git gemerkt werden. Für mehr lesen Sie unter `$HOME/export/Oop` die Datei `READMEnethome.txt` oder `NethomeDotProfile.pdf`, um zu verstehen wie das geht und was man noch machen kann. Wenn es klappt, dann merkt sich Linux alle Ihre Einstellungen (`code`, Git, Desktop, Eclipse, ...) immer und man kann produktiv im Pool arbeiten.

Unter Windows ist wieder alles ein bisschen komplizierter. Einfache Links und flexible Konfiguration sind hier leider nicht möglich und Roaming Profiles auch keine gute Idee.

Also ist unter Windows nur Visual Studio Code so konfiguriert, dass die Einstellungen in `Z:\VSCODE` gesucht werden. Das heißt unter Windows ist alles weg, außer die `code`-Einstellungen, das dafür ohne weitere Aktionen. Dafür kann sich Windows sonst nichts merken (auch nicht die Git-Sachen).

Aufgabe 2. Sie haben schon früher mit Git gearbeitet. Für die Veranstaltung hier erhalten Sie auch ein Git-Repository. Das Repository ist verfügbar unter

```
https://scm.inftech.hs-mannheim.de/scm/git/19pythxx
```

wobei `xx` für eine Ihnen zugewiesene zweistellige Nummer steht. Loggen Sie sich zunächst mindestens einmal unter

```
https://scm.inftech.hs-mannheim.de/scm/
```

ein. Dann schicken Sie mir eine Email mit Matrikelnummer und Sie erhalten das Repo mit der zweistelligen Nummer (statt xx) für Ihren Account (Matrikelnummer, Passwort wie im SWT-Labor) freigeschaltet. Im Folgenden nehmen wir in Beispielen als Repository `19pyth00` (also xx=00) und als Matrikelnummer `6660815` an. Sie checken das Projekt entweder mit Tortoise-Git und anderen Klick-Tools oder einfach auf der Konsole mit

```
git clone https://6660815@scm.inftech.hs-mannheim.de/scm/git/19pyth00
```

aus. Wenn Sie einen weiteren Namen (wie `pyprakt`) dahinter angeben heißt der lokale Ordner `pyprakt` statt `19pyth00`. Sie sollten unter Linux im Verzeichnis `nethome` oder unter Windows im `Z:` Laufwerk sein. Achtung: Verwenden Sie ein ausgechecktes Repository **nie** gleichzeitig unter Windows und Linux, stattdessen checken Sie das Repository je Plattform neu aus! In dem ausgecheckten Repository finden Sie schon einen Ordner: `PyPrgs`. Sie können alle Python-Aufgaben in `PyPrgs` auf oberster Ebene halten (oberste Ebene! keine Unterordner, das bedeutet was anderes). Eine korrekte Verzeichnisstruktur sieht also so (links Linux, rechts Windows) aus.

```
nethome/19pyth00                          Z:\19pyth00
nethome/19pyth00/PyPrgs                   Z:\19pyth00\PyPrgs
nethome/19pyth00/PyPrgs/hello.py          Z:\19pyth00\PyPrgs\hello.py
nethome/19pyth00/PyPrgs/heron.py          Z:\19pyth00\PyPrgs\hreon.py
nethome/19pyth00/PyPrgs/calc.py           Z:\19pyth00\PyPrgs\calc.py
nethome/19pyth00/PyPrgs/imagemanip.py     Z:\19pyth00\PyPrgs\imagemanip.py
...                                       ...
```

Sie finden beim ersten Auschecken dort schon ein ähnliches Hello-Programm `hello.py`. Ändern Sie es ab, lassen Sie es zum Beispiel `Hello Python and Git World` ausgeben.

Sie können ab dann alle Git-spezifischen Sachen auch innerhalb von Visual Studio Code machen. Experimentieren Sie, die Benutzungsschnittstelle sollte recht intuitiv sein. Es geht immer los über die Wahl links Versionskontrolle. Falls ein Ordner offen ist und dieser unter Git-Versionskontrolle steht, dann erkennt das die IDE. Die Benutzungsschnittstelle zu Git sollte selbsterklärend sein. Vergessen Sie nicht zu pushen (das sollte über die Punkte ... im linken Fenster rechts oben gehen). Am besten legen Sie vor allem anderen noch eine Datei `$HOME/.gitconfig` an. Das klappt auch mit den `git config` Befehlen wie in `InstallPython.pdf` oder direkt mit einem Editor. Beachten Sie dazu vorher Hinweis 1.

```
[user]
```

```
    email= 6660815@stud.hs-mannheim.de
    name = susi
[push]
    default = simple
[credential]
    helper = cache --timeout=9600
[core]
    pager = less -r
    autocrlf = input
```

die häufiges Eintippen spart.

Hinweis 2. Es gibt mehrere Möglichkeiten die Übungsaufgaben zu verwalten. Das einfachste ist alle Python-Aufgaben in einem einzigen Ordner `PyPrgs` zu verwalten. Dann sind **alle** Dateien **direkt** in diesem Ordner. Sie haben **IN DIESEM ORDNER KEINE UNTERORDNER**. Wenn das für Sie persönlich gar nicht geht, dann legen Sie **neben** dem `PyPrgs` Ordner neue Ordner an wie `Blatt1`, `Blatt2`, etc. aber Sie werden die Ordner **nicht!** verschachteln (das bedeutet etwas anderes in Python). Sie müssen dann eben beim Wechseln von Aufgabenblättern unnötigerweise immer noch den Ordner wechseln.

Verboten sind Leerzeichen und Sonderzeichen in Ordner- und Dateinamen. Verzichten Sie auf – und _. Für Dateinamen nutzen wir nur Kleinbuchstaben und Zahlen nur am Ende.

Hinweis 3. Ihr Repository ist schon mit einer Datei `.gitignore` vorbereitet, so dass viele offensichtlich falsche Dateien nicht eingecheckt werden. Die beinhaltet unter anderem

```
__pycache__/
.vscode
.DS_Store
*.log
*.zip
*.tar.gz
```

Das bedeutet auch, dass wir uns IDE-Einstellungen eben genau **nicht** merken, das sind lokale Einstellungen. Wir merken uns auch von Python generierte Dateien (`__pycache__/*` nicht, da auch diese je nach Plattform anders sind und anders erzeugt werden. Versionskontrolle != Dropbox, und das ist gut so.

Hinweis 4. Arbeiten Sie immer wieder mal ein bisschen an den Aufgaben, kleine kurze Phasen, von Ziel zu Ziel in kleinen Schritten. Auch wenn Sie später in Firmen-Projekten vielleicht nicht so häufig committen und pushen werden, sollten Sie es jetzt tun, damit Sie immer und überall (an der Hochschule oder zu Hause oder bei Kommilitonen) Fortschritt machen können. Also, am Ende einer Session in Ihrem individuellen Repository immer committen und pushen. Wer es an der Hochschule vergessen hat, der kann sich über VPN einloggen und dann mit

```
ssh swtssh.inftech.hs-mannheim.de -l 6660815
```

auf einen Rechner im SWT-Labor verbinden. Dort können Sie mit

```
cd ~/nethome/19pyth00
git status # pruefen
git commit -m 'changes' -a # mit Kommentar, alle geänderten Dateien
git push # hoch ins repo
```

die meisten Änderungen ins Repository pushen. Ihre Änderungen von zu Hause oder sonstwo spielen Sie zurück in das ausgecheckte Repository an der Hochschule mit

```
cd ~/nethome/19pyth00
git pull # aus dem repo
```

Sie haben sogar die Möglichkeit mit VPN von zu Hause graphisch an einem Rechner zu arbeiten mit `X2go` auf dem Rechner `kvmswt.vswt.swt.hs-mannheim.de`.

Hinweis 5. Namenskonventionen sind bindend. Wir sind Entwickler, kriegen Geld und machen es Kollegen einfach und halten uns daher an Namenskonventionen. Anwender bezahlen Geld, sind schwierig und dürfen Sonderzeichen und Leerzeichen tippen.

Die Namenskonventionen für Entwickler sind immer, dass in Pfaden jedwede Art von Sonderzeichen verboten sind. Auch und gerade Leerzeichen sind **verboten**. Die Aussage, “wieso, geht doch” rächt sich meist irgendwann, wenn man überhaupt nicht damit rechnet. Hilfestellung gibt es nur, wenn der Pfad keine Leerzeichen, Sonderzeichen etc. enthält. Groß-/Kleinschreibung in Pfaden mischen ist auch verpönt (wenn auch nicht verboten), da Windows einen recht entspannten Umgang mit der Unterscheidung von Groß-/Kleinschreibung an den Tag legt.

Selbst im Quelltext (der Unicode sein könnte) halten wir uns an einfache Namensregeln für Bezeichner und verbieten Sonderzeichen, Groß-/Kleinschreibung ist erlaubt und erforderlich. Also nur die Zeichen

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
```

und dann noch keine führenden Ziffern. Wer unbedingt Sonderzeichen in Kommentaren oder Strings verwenden will, der muss `UTF-8` im Editor einstellen, wenn er etwas exotisches verwendet. Die vorgeschlagenen IDEs machen das alle automatisch richtig. Dann klappt es auf allen Plattformen Linux, MacOSX und Windows.

Gewöhnen Sie sich an die Namenskonventionen und Formatierungskonventionen in Python (`www.python.org/dev/peps/pep-0008/`). Wir nutzen für Funktionen und Methoden `snake_case`, nur für Klassen `CamelCase`, Werte die man nicht ändern soll sind `ALL_CAPS`. Die sind vielleicht ein bisschen anders als gewohnt, aber Programmieren findet später in Teams statt und Konventionen gelten dann immer für ein Team und nicht für ein kreatives Individuum. Wir lernen im Kopf umschalten und formatieren wie der Mainstream/das Team es vorgibt.

moodle Python_Barth