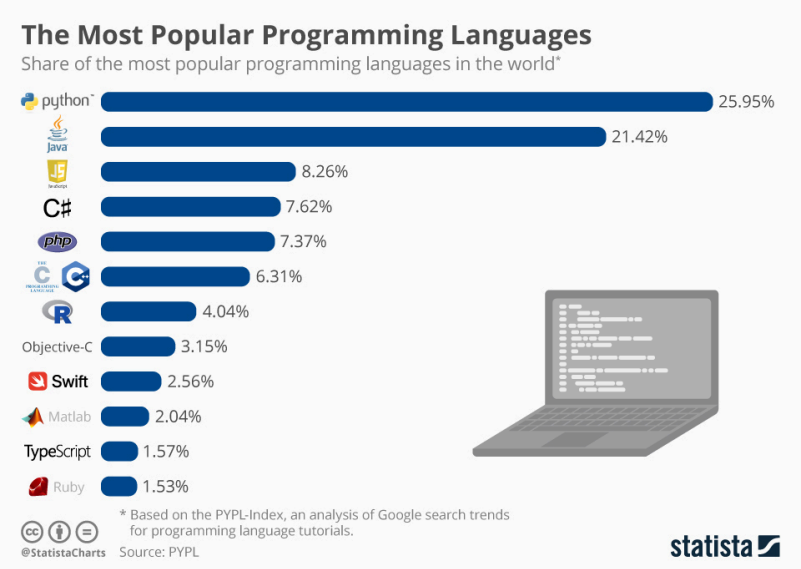
By the Power of Thor: Spielerischer Einstieg in Python

Willkommen zu unserem Python-Workshop!

Python ist eine ***Computerprogrammiersprache*** - ein Satz von Vokabeln und Regeln, mit denen man einen Computer anweisen kann, bestimmte Aufgaben auszuführen. Sie wurde von ihrem Schöpfer - Guido van Rossum - nach dem "Monty Python's Flying Circus" benannt. Seit ihrer Entstehung im Jahr 1991 hat sie sich zu einer der am häufigsten verwendeten Programmiersprachen entwickelt:



Quelle: Statista, 2019

Guido wollte eine Sprache entwickeln, die leicht zu erlernen und zu erweitern war. Dies ist auch im **"Zen von Python"** verankert, der von Guido geschriebenen Python-Stil-Guide.

In [ ]:
```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Python ist eine vielseitige Programmiersprache, die aufgrund ihrer Einfachheit und ihrer leistungsstarken Bibliotheken in vielen Bereichen eingesetzt wird. Hier sind einige wichtige Anwendungen von Python:

1. **Web-Entwicklung**: Python-Frameworks wie Django und Flask sind beliebt, um Webanwendungen schnell und mit wenig Code zu erstellen.
2. **Data Science und maschinelles Lernen**: Python ist eine führende Sprache für Datenanalyse, Visualisierung und komplexe maschinelle Lernalgorithmen. Bibliotheken wie NumPy, pandas, scikit-learn und TensorFlow machen Python zu einem Favoriten unter Datenwissenschaftlern.
3. **Automatisierung**: Python vereinfacht die Automatisierung sich wiederholender Aufgaben bei Softwaretests, Datenmanagement und Systemadministration.
4. **Software-Entwicklung**: Die Lesbarkeit und das robuste Ökosystem von Python machen es ideal für die Entwicklung einfacher Skripte und komplexer Unternehmenssoftwareanwendungen.
5. **Bildungsprogramme**: Die einfache Syntax und die gute Lesbarkeit machen Python zu einem ausgezeichneten Tool für das Erlernen einer Programmiersprache insbesondere in Hochschulen und Einführungskursen

In diesem Workshop werden wir die grundlegenden Konzepte der Sprache demonstrieren, die euch helfen können, in eure eigenen Projekte einzusteigen.

Viele der Beispiele beruhen auf dem Buch von Al Sweigart (2017): Invent your own computer games with Python.

In [ ]:
```
# Dies ist ein Kommentar in Python.
# Kommentare beginnen mit einem Hashtag und werden vom Python-Interpreter ignori

# Die print-Funktion wird verwendet, um Text oder Zahlen auf der Konsole auszuge
print("Hallo und Willkommen zu unserem Python-Workshop!")
```

```
Hallo und Willkommen zu unserem Python-Workshop!
```

Wertzuweisung und Grundlegende Datentypen

Werte werden in Variablen mit einem Gleichheitszeichen (=) gespeichert. Dadurch können wir verschiedene Operationen mit den Werten durchführen.

In [ ]:
```
zahl = 5
zahl1 = 2

print(zahl + zahl1)
```

```
7
```

Neben Zahlen ist auch Text ein häufig verwendeter Datentyp. In Python werden die Textbausteine als "Strings" bezeichnet. Strings können genau wie Zahlen verwendet werden: Sie können gespeichert und gedruckt werden.

In [ ]:
```
gruß = "Hallo Welt!"

print(gruß)
print("Typ:", type(gruß))
```

```
Hallo Welt!
Typ: <class 'str'>
```

Wir können Strings mit dem „+“-Operator in einer Aktion namens String-Verkettung (*string contatination*) kombinieren.

In [ ]:
```
verkettung = "Hallo Welt!" + " " + "Ich lerne Python!"
print(verkettung)
```

```
Hallo Welt! Ich lerne Python!
```

Es gibt andere Datentypen in Python. Einer davon sind Warheitswerte (Boolesche Werte). Boolesche Werte haben nur zwei Werte: „True“ oder „False“.

In [ ]:
```
print(10 > 12)
print(10 < 12)
print(type(10 < 12))
```

```
False
True
<class 'bool'>
```

Weitere Datentypen und Indexing

Liste ist ein Datentyp, der mehrere andere Werte enthält. Listen beginnen mit einer linken eckigen Klammer ([) und enden mit einer rechten eckigen Klammer (]). Kommas trennen die Werte - Elemente (*items*) - innerhalb der Liste.

In [ ]:
```
nummern_liste = [1, 2, 3]
text_liste = ["Thor", "Odin", "Freyr"]

print(nummern_liste)
print(text_liste)
```

```
[1, 2, 3]
['Thor', 'Odin', 'Freyr']
```

Auf Elemente in einer Liste kann durch Indizierung (*indexing*) zugegriffen werden. Python-Listen sind "null-indiziert" (*zero-indexed*), was bedeutet, dass die Indizes bei 0 beginnen.

In [ ]:
```
print(nummern_liste[0])
print(text_liste[1])
```

```
1
Odin
```

Die Elemente in der Liste behalten ihren Datentyp, wenn sie einzeln indiziert werden.

In [ ]:
```
print(type(text_liste))
print(type(text_liste[1]))
```

```
<class 'list'>
<class 'str'>
```

Wir können eine "Liste der Listen" erstellen.

In [ ]:
```
liste_der_listen = [[2, 5, 3], [0, 1, 5], [1, 1, 7]]

print(liste_der_listen)
```

```
[[2, 5, 3], [0, 1, 5], [1, 1, 7]]
```

Achtung: Jetzt brauchen wir zwei Indizes, um zu einer einzigen Zahl in einer Liste zu gelangen. Der erste Index ruft eine Liste auf, der zweite das Element innerhalb einer Liste.

In [ ]:
```
print("Eine Liste:", liste_der_listen[1])
print("Ein Zahl:", liste_der_listen[1][0])
```

```
Eine Liste: [0, 1, 5]
Ein Zahl: 0
```

Mit dem Operator "+" können wir mehrere Listen zu einer einzigen zusammenfassen.

In [ ]:
```
print(nummern_liste + text_liste)
```

```
[1, 2, 3, 'Thor', 'Odin', 'Freyr']
```

Achtung: Um jedoch einzelne Elemente zur Liste hinzuzufügen, müssen wir die Methode append() verwenden.

In [ ]:
```
text_liste.append("Heimdall")
print(text_liste)
```

```
['Thor', 'Odin', 'Freyr', 'Heimdall']
```

Kontrollstrukturen

Aktionen in Python können durch Kontrollstrukturen wie if- und while-Anweisungen automatisiert werden:

- **if-Anweisungen:** Diese ermöglichen es uns, bestimmte Abschnitte des Codes nur dann auszuführen, wenn spezifische Bedingungen erfüllt sind. Zum Beispiel kann eine if-Anweisung überprüfen, ob eine Zahl positiv ist, und nur dann weitere Aktionen mit dieser Zahl durchführen.

- **while-Anweisungen:** Diese werden verwendet, um einen Block von Code wiederholt auszuführen, solange eine Bedingung wahr bleibt. Beispielsweise könnte eine while-Schleife so lange nach Benutzereingaben fragen, bis eine gültige Eingabe erfolgt.

- **for-Anweisungen:** Diese werden benutzt, um einen Code für jedes Element einer Liste oder eines vergleichbaren Datentyps auszuführen

In [ ]:
```python
# Beispiel für if-Anweisungen
zahl = 10

if zahl > 5:
    print("Die Zahl ist größer als 5.")
elif zahl < 5:
    print("Die Zahl ist kleiner als 5.")
else:
    print("Die Zahl ist 5.")
```

```
Die Zahl ist größer als 5.
```

In [ ]:
```python
# Beispiel für eine while-Schleife
counter = 0

while counter < 5:
    print("Counter ist noch immer weniger als 5:", counter)
    counter += 1  # Erhöht den Counter um 1
```

```
Counter ist noch immer weniger als 5: 0
Counter ist noch immer weniger als 5: 1
Counter ist noch immer weniger als 5: 2
Counter ist noch immer weniger als 5: 3
Counter ist noch immer weniger als 5: 4
```

In [ ]:
```python
liste = ["Thor", "Odin", "Freyr"]
for name in liste:
    print(name + " wünscht euch einen schönen Tag!")
```

```
Thor wünscht euch einen schönen Tag!
Odin wünscht euch einen schönen Tag!
Freyr wünscht euch einen schönen Tag!
```

Dictionaries

Dictionaries (Wörterbücher) in Python sind eine effiziente Möglichkeit, Schlüssel-Wert-Paare zu speichern und abzurufen. Ein Dictionary hat, wie eine Liste, ebenfalls eine Sammlung von Werten. Allerdings liegen diese Werte jetzt in Paaren von Schlüssel und Wert vor. Ein Paar wird durch einen Doppelpunkt (:) getrennt und ein Wörterbuch wird in geschweifte Klammern ({}) eingeschlossen.

In [ ]:
```python
# Ein Wörterbuch, das nordische Götter mit ihren charakteristischen Werkzeugen v
```

```
gott_werkzeuge = {
    "Thor": "Mjölnir",  # Thors Hammer
    "Odin": "Gungnir",  # Odins Speer
    "Freyr": "Schiff Skidbladnir",  # Freyrs magisches Schiff
    "Heimdall": "Gjallarhorn"  # Heimdalls Horn
}

print(type(gott_werkzeuge))
```

```
<class 'dict'>
```

Auf die Werte im Wörterbuch kann mit Hilfe des Schlüssels zugegriffen werden.

In [ ]:
```
# Zugriff auf Werte im Wörterbuch

gott = "Thor"
werkzeug = gott_werkzeuge[gott]

print(gott, "besitzt das Werkzeug:", werkzeug)
```

```
Thor besitzt das Werkzeug: Mjölnir
```

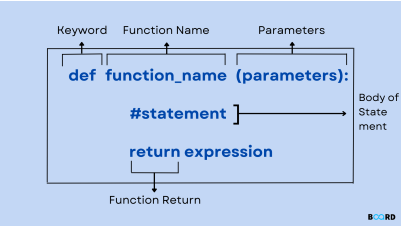
Wir können auch ein Wörterbuch mit komplexeren Schlüsseln haben.

In [ ]:
```
gott_stimmung = {

    ("Thor", "ohne Mjölnir"): "Traurig",
    ("Thor", "mit Mjölnir"): "Froh"
}

key = ("Thor", "ohne Mjölnir")

print("Wie ist die Stimmung heute?", gott_stimmung[key])
```

```
Wie ist die Stimmung heute? Traurig
```

Funktionen in Python

Funktionen sind zentrale Bausteine in Python und erlauben Code in wiederverwendbaren und organisierten Einheiten zu strukturieren. Die Funktionen bestehen aus mehreren Teilen:



Quelle: https://www.boardinfinity.com/blog/functions-in-python/

Anstatt immer wieder den gleichen Code zu schreiben, können wir eine Funktion für eine bestimmte Aufgabe schreiben.

In [ ]:
```
def is_it_five(zahl):

    if zahl > 5:
```

```
        antwort = "Die Zahl ist größer als 5."
    elif zahl < 5:
        antwort = "Die Zahl ist kleiner als 5."
    else:
        antwort = "Die Zahl ist 5."

    return antwort
```

Wir müssen die Funktion zuerst ausführen, um sie zu "aktivieren", und dann können wir sie in weiteren Operationen verwenden.

In [ ]:
```
x = 5

print(is_it_five(x))
```

```
Die Zahl ist 5.
```