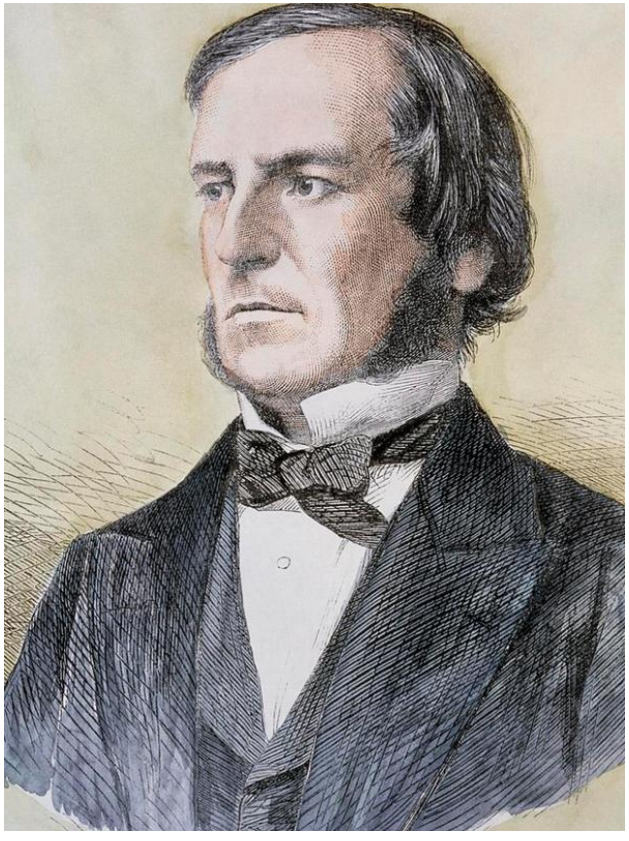
Podmienky a rekurzia

bool

Boolean je datový typ, ktorý nadobúda 2 hodnôt: `True` a `False`

```
print(type(True))
```



Obr. 1: George Boole

Väčšie, menšie, rovná sa

```
a = 1
b = 3
print(a < b)
```

Operátory, ktoré budeme potrebovať:

`<` `>` `<=` `>=` `==` `!=` `not`

`!=` je *nerovná sa*.

`not` je unárny operátor (1 argument, ostatné hore sú binárne):

```
print(not True)
```

Rozhodovanie

Napíšte funkciu, ktorá vypíše či je argument kladný, záporný alebo nula.

```
def print_sign(x):
    if x < 0:
        print("negative")
    else:
        print("non-negative")
```

Nezabudnúť na `:` a indentovaný blok inak dostanete `SyntaxError`.

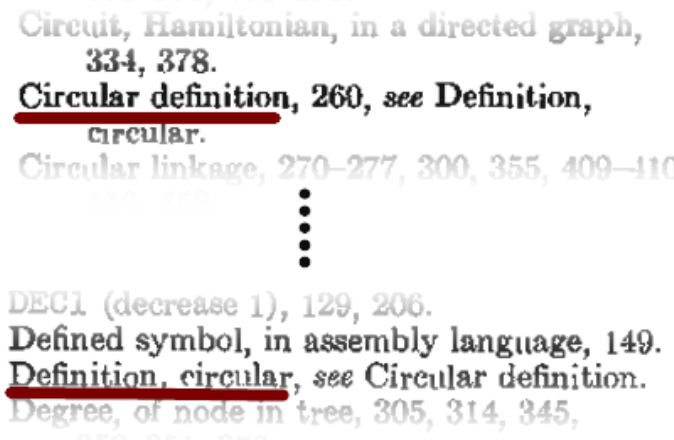
Rozhodovanie

```
def print_sign(x):
    if x < 0:
        print("negative")
    else:
        if x == 0:
            print("zero")
        else:
            print("positive")
```

Rozhodovanie

```
def print_sign(x):
    if x < 0:
        print("negative")
    elif x == 0:
        print("zero")
    else:
        print("positive")
```

Rekurzia



Obr. 2: Z knižky *The Art of Programming* od D. Knuth

```
def rekurzia():
    rekurzia()
```

Poznámka 1: V Pythone je rekurzia obmedzená a väčšinou sa preferuje `for`-cyklus (naučíme sa na ďalšej hodine).

Poznámka 2: `Ctrl + C` je váš kamarát, ak by ste sa niekedy dostali do problémov.

Rekurzia v praxi

Napíšte funkciu, ktorá odpočíta od daného čísla do 0.

```
def countdown(n):
    if n == 0:
        print(0)
    else:
        print(n)
        countdown(n - 1)
```

Každá užitočná rekurzívna funkcia niekde skončí

- Rekurzívne funkcie budú mať `if` príkaz.
- Jedna vetva `if` obsahuje tzv. *base case* -> žiadna rekurzia / koniec rekurzie
- Druhá vetva obsahuje rekurziu so zmeneným argumentom.

```
def countdown(n):
    if n == 0:          # base case
        print(0)
    else:               # rekurzivna vetva
        print(n)
        countdown(n - 1) # zmeneny argument
```

Zjednodušenie

- `print(n)` nastane v oboch vetvách

```
def countdown(n):
    print(n)
    if n > 0:
        countdown(n - 1)
```

Demo - Hanojské veže

- inšpirované videom
- anglicky Tower of Hanoi
- objekty a metódy na vykresľovanie

Teraz už naprogramujete všetko... teoreticky

Turingovká úplnosť

Tiež, niektoré veci dokázateľne nie je možné vyriešiť programom -> viz. halting problem

Koncept rovnosti vo väčšom detaile

```
a = 1
b = 1.0

print(type(a))
print(type(b))

a == b     # ma to byt True?
```

"Silnejšia" rovnosť

Objekty majú svoju identitu. V implementácií CPythonu je identita adresa v pamäti.

```
a = 3
b = "B"

print(id(a))
print(id(b))
```

Porovnanie identity je možné pomocou operátora `is` alebo `is not`:

```
a = 1
b = 1.0

print(a is b) # ekvivalentne: print(id(a) == id(b))
print(a is not b) # print(id(a) != id(b))
```

Poznámka: `is not` je jeden operátor, nie `is` + `not`.

Kedy použiť `is` namiesto ==?

- Keď vás zaujíma či dva objekty sú naozaj jeden objekt - sú na rovnakom mieste v pamäti.
- Porovnanie s `None` (`None` je len jeden)

Poznámka: Existuje celá rada podivností, keď pracujete s typom "ako keby bol" bool a pri použití `is` a `not`:

```
print(not None)
print(not not None)
print(not [])
print(not 0)
print(not 1)
```

alebo toto, alebo toto, alebo toto.

Poučenie: nechovajte sa k iným datovým typom ako k `bool`-om.