Vývoj aplikací pro iOS #3
Práce s proměnnými

Návod ■ Daniel Březina

Pojďme navázat tam, kde jsme **skončili**. Minule jsme si ukázali co to je proměnná, jak takovou proměnnou vytvořit a jaké základní datové typy proměnných ve Swiftu máme. Dnes tyto znalosti rozšíříme o další zajímavosti.

Jak už víme, práce s proměnnou je základ programování. Proměnné můžeme různě modifikovat, kompletně vyměnit nebo z její hodnoty zjistit další užitečné informace. Takových možností existuje nespočet, my si dnes ukážeme pár z nich. Cílem není vám ukázat, co všechno lze dělat s čísly nebo textem, ale jak se s těmito typy dá pracovat.

JEDNODUCHÁ KALKULAČKA

Určitě jste se setkali s tezí, že programátoři jsou perfektní v matematice. Že dokážou vypočítat kde co. Obecně vzato, pravda to není. Pokud budete chtít programovat grafické aplikace nebo různé vizualizace, neobejdete se bez znalostí lineární algebry nebo derivací a integrálů. Na tvorbu webových stránek nebo mobilních aplikací si vystačíte s obyčejnou aritmetikou, kterou znáte od první třídy. Výjimečně vám k tomu přibydou například goniometrické funkce.

Chcete důkaz, že vám stačí obyčejné počítání z první třídy? Vytvořte si dvě konstanty. Například **let a = 1** a **let b = 2**. Nyní přijde velké kouzlo. Pod tyto tři konstanty vytvořte novou konstantu: **let c = a + b**. Vzhledem k tomu, že jak **a**, tak **b** jsou celá čísla neboli datový typ Integer, lze je bez problému sečíst a výsledek, konstanta **c**, je pak také typu Integer. Můžete se přesvědčit pomocí známé klávesové zkratky **Option + levé tlačítko myši** na názvu konstanty.

Samozřejmě Swift podporuje všechny základní aritmetické operace. Tedy sčítání, odčítání, násobení a dělení. U dělení ale pozor. Když si teď odmyslíme programování a napíšeme si příklad 1 / 2, je nám jasné, že výsledek je 0,5, tedy desetinné číslo. Ovšem v programování, konkrétně ve Swiftu, rozlišujeme celá čísla a desetinná čísla. Když dělíte dvě celá čísla, výsledek bude také celé číslo. V našem případě to bude 0. To proto, že se uložila pouze celá část čísla a desetinná se zahodila. Schválně, kolik vám vyjde, když si za **a** dosadíte 3?

Velký problém také nastane ve chvíli, kdy chcete dělit celé číslo číslem 0. Jak asi víte ze školy, nulou dělit nelze. Bohužel v tomto případě si musíte dávat pozor vy, programátoři. Xcode vás upozorní až po samotném procesu dělení. Stát se to v reálné aplikaci, tak by tato aplikace spadla. Je tedy důležité si dávat pozor, jakým číslem se dělí. A to si ukážeme v jednom z dalších dílů série. Jiná situace ale nastane, pokud chcete dělit „desetinnou“ nulou desetinné číslo. Tedy **let a = 1.0** a **let b = 0.0**. V takovém případě vám vyjde výsledek nekonečno.

SLOŽITĚJŠÍ KALKULAČKA

Základní matematické operace jsou velmi jednoduché. Nyní přejdeme na složitější. Nejprve složitější z hlediska funkcionality. Při tvorbě aplikací se často potkáte s tím, že je potřeba nějaké číslo postupně inkrementovat. Zvyšovat ho o danou hodnotu. Ze znalostí, které aktuálně máte, by vás mohl napadnout tento zápis:

```
var a = 1
a = a + 1
```

Je to samozřejmě správně, ale druhá řádka se dá zkrátit na tvar **a += 1**. Tento zápis znamená, že se k hodnotě proměnné **a** přičte číslo 1. Místo znaménka plus může být jakékoliv ze znamének **+ - * /**. Tento zápis se využívá velmi často.

Existuje ještě jedna základní matematická operace. Jmenuje se modulo a dává nám zbytek po dělení. Vezměme si opět čísla **let a = 1** a **let b = 2**. Operace modulo má znaménko **%**. Když operaci provedeme, **let c = a % b**, dostaneme výsledek 1. Co to znamená? Znamená to, že výsledek po celočíselném dělení 1 / 2 je 0 a zbylo nám právě 1.

Pomocí operace modulo tak můžeme například zjistit, jestli je číslo sudé nebo liché. Pokud je sudé, výsledek operace **x % 2**, bude vždy 0. Pokud je liché, výsledek bude vždy 1. Nebo můžeme zjistit, jestli je číslo dělitelné jiným číslem. V takovém případě bude výsledek operace modulo 0. Nebo můžeme znalost modula využít v algoritmu, který nám řekne, jaký den v týdnu připadá na konkrétní datum. A to si ukážeme v jednom z posledních dílů první části seriálu, až budeme mít dostatek znalostí.

KOLIK MÁ TEXT ZNAKŮ?

Práce s čísly je zábavná, ale ještě zábavnější je práce s textem. Na začátek důležitá otázka. Kolik má hodnota následující konstanty **let helloWorld = "Hello world"** znaků? Proč to počítat ručně? Můžeme si na to udělat následující konstantu **let count = helloWorld.count**. Datový typ string má určité vlastnosti, neboli properties. Nejenom stringy, ale obecně všechny datové typy, které ve Swiftu jsou. Některé z nich najdete v knížce The Swift Programming Language, kompletní seznam je potom k dispozici v dokumentaci. Property (ano opravdu se i mezi českými vývojáři používá tento výraz) **.count** vrací počet znaků daného stringu.

Na internetu jsou k dispozici zdrojové kódy k tomuto seriálu a ve zdrojovém kódu pro předchozí díl jste mohli vidět použitou funkci **print()**. Tato funkce vytiskne zadaný obsah do terminálu. Můžeme použít například **print(helloWorld).** Nebo **print(count)**. A úplně nejlepší je třetí způsob. Zkombinovat obě proměnné. Pozorní z vás právě

zbystřili, protože ví, že kombinování různých datových typů není ve Swiftu možné. Jak tedy zároveň vytisknout datový typ String s jiným datovým typem, v našem případě s datovým typem Int? Použije se na to proces zvaný String interpolation, který umožňuje převést jakýkoliv ze základních datových typů do datového typu String. A to i v případě, že proměnná nebo konstanta jsou datového typu String. Obsah interpolované proměnné se nezmění. Pro String interpolation se používá následující konstrukce **“\()“**. Mezi závorky se pak vloží daná proměnná nebo konstanta. V našem případě tedy můžeme použít: **print(“String \(helloWorld) has \(count) characters.“)**.

DALŠÍ SČÍTÁNÍ A POROVNÁVÁNÍ

Kdy jste naposledy sčítali slova? Nikdy? V programování začnete. Mějme dvě konstanty **let hello = “hello“** a **let world = “world“**. Jejich sečtením, **let helloWorld = hello + world**, dostaneme slovo helloworld. Ano, bez mezery.

Jednotlivé stringy se dají i porovnávat. A nejenom stringy, ale samozřejmě i čísla. Opět je to velmi podobné jako na základní škole. Pro porovnání se používají znaménka **>**, **<** (větší, menší), **>=**, **<=** (větší nebo rovno, menší nebo rovno) a **==**, **!=** (rovná se, nerovná se). V případě stringů se používá primárně porovnání nerovnosti. Pokud bychom udělali **print(hello > world)**, tak se neporovnává počet znaků, ale jejich unicode hodnota (více zde). Tedy v běžném programování iOS aplikací se s tím na 99 % nesetkáte.

Pokud si zkoušíte zároveň s textem psát kód, tak jste si všimli, že **print(hello > world)** vypsal **false**. Výsledkem porovnání je totiž další základní datový typ, který se jmenuje Bool. Bool můžete mít dvě hodnoty, buď **true** nebo **false**. Pravda, nepravda. Velmi jednoduché použití.

Tento díl byl zaměřený na práci s proměnnými. V dalším díle si ukážeme další datové typy, konkrétně kolekce dat. Velká část toho bude opakování. Datový typ String je také kolekce dat. Kolekce znaků. Více ale příště.

INFO

Postupem času budeme psát víc a víc kódu a ne všechno se vejde do článku. Je tedy dobré mít zdrojové kódy pohromadě a rozdělené podle jednotlivých článků. To vše najdete pod tímto odkazem.