

INFORMATIKA

PRO STŘEDNÍ ŠKOLY

Pavel NAVRÁTIL



- ZÁKLADY PROGRAMOVÁNÍ
 - ZPRACOVÁNÍ DAT
- INFORMAČNÍ SYSTÉMY
- ÚVOD DO DATABÁZÍ

TEMATICKÉ ROZDĚLENÍ UČIVA

INFORMATIKA PRO STŘEDNÍ ŠKOLY 1. díl

- Data a informace
- Kódování, komprese a přenos dat
- Základy hardwaru, operační systémy
- Modelování
- Algoritmizace

INFORMATIKA PRO STŘEDNÍ ŠKOLY 2. díl

- Základy programování
- Zpracování dat
- Informační systémy
- Úvod do databází

INFORMATIKA PRO STŘEDNÍ ŠKOLY 3. díl

- Počítačové sítě
- Umělá inteligence
- Moderní digitální technologie
- Digitální bezpečnost

OBSAH

ZÁKLADY PROGRAMOVÁNÍ

ÚVOD DO PROGRAMOVÁNÍ	8
CO JE TO PROGRAMOVÁNÍ?	8
FÁZE VÝVOJE PROGRAMU	8
PROGRAMOVACÍ JAZYKY	9
Jaké existují programovací jazyky?	9
PROGRAMOVACÍ JAZYK PYTHON.....	10
Jak programovat v Pythonu a co je k tomu potřeba? ...	10
Varianta 1: Python online – vývojové prostředí Pythonu v prohlížeči.....	10
Varianta 2: Python instalovaný v počítači	11
Vývojové prostředí Pythonu – IDLE Shell okno	11
Vývojové prostředí Pythonu – okno editoru kódu.....	12
PROGRAMOVÁNÍ V PYTHONU POMOCÍ EDITORU KÓDU A SHELLU	13
Varianta 3: Python instalovaný v počítači + externí vývojové prostředí	14
ZÁKLADY PROGRAMOVÁNÍ V JAZYCE PYTHON.....	14
SYNTAXE	14
VYPSÁNÍ VÝSTUPU – TEXT A MATEMATICKÉ VÝRAZY	15
PROMĚNNÉ	17
Proč se vůbec používají proměnné?	17
DATOVÉ TYPY PROMĚNNÝCH	18
Proč jsou datové typy důležité?.....	18
Jak se definují datové typy?.....	18
VYPSÁNÍ VÝSTUPU – PROMĚNNÉ.....	19
CHYBOVAT PŘI PROGRAMOVÁNÍ JE BĚŽNÉ	20
VSTUP OD UŽIVATELE.....	23
Vstup netextové informace od uživatele	24
Převod datových typů	24
PODMÍNKY (VĚTVENÍ)	27
Relační operátory	28
Příkaz podmínky – if	28
LOGICKÉ OPERÁTORY	39
Příklad užití operátorů and a or.....	40
CYKLY	43
Cyklus for – cyklus s předem daným počtem opakování	43
Cyklus while – cyklus s podmíněným počtem opakování.....	46
SEZNAMY	50
Jak seznam funguje a jak se s ním pracuje?	50
Vybrané metody a funkce pro práci se seznamy	51
FUNKCE ANEB PODPROGRAMY V PYTHONU	54
Obsah a parametry funkce (podprogramu).....	54
Funkce bez návratové hodnoty	54
Funkce s návratovou hodnotou	58
KOMENTÁŘE V KÓDU PYTHONU	62
Víceřádkové komentáře	62
CHYBY PŘI VÝVOJI PROGRAMU.....	63
DRUHY CHYB PŘI PROGRAMOVÁNÍ.....	63
Syntaktické chyby	63
Běžové chyby	64
Logické chyby	64

KROKOVÁNÍ A LADĚNÍ PROGRAMU (DEBUGGING).....

Jak krokovat program?.....	65
Metody debugování	66

ZÁKLADNÍ PRAVIDLA VÝVOJE PROGRAMU

OPTIMALIZACE PROGRAMU

Ukázka vhodného a nevhodného využití výpočetního výkonu.....	66
--	----

UŽIVATELSKÉ ROZHRANÍ

Konzistence vzhledu, chování a terminologie	68
---	----

NÁPOVĚDA K PROGRAMU

Interaktivní nápověda	69
Průvodce.....	69
Klasická nápověda.....	70

ETIKA PROGRAMÁTORA

ZÁKLADNÍ ETICKÉ ASPEKTY PROGRAMÁTORA.....

Kvalita a bezpečnost kódu	70
Ochrana soukromí	70
Duševní vlastnictví a licence	70
Odmítnutí manipulativních vzorců.....	70

INFORMAČNÍ SYSTÉMY

INFORMAČNÍ SYSTÉM

CO JE TO INFORMAČNÍ SYSTÉM?

Příklad informačního systému	72
------------------------------------	----

ÚČEL A PRINCIP FUNGOVÁNÍ INFORMAČNÍHO SYSTÉMU

HLAVNÍ VÝHODY INFORMAČNÍCH SYSTÉMŮ

Z ČEHO SE SKLÁDÁ INFORMAČNÍ SYSTÉM?

Hardware	75
Software	75
Procesy	75
Data	75
Lidé	75

UŽIVATELE A UŽIVATELSKÉ ÚČTY

Ochrana informací.....	76
Ochrana dat proti úpravám.....	76
Ochrana uživatele.....	76
Přístupová práva	76
Uživatelské role a uživatelské skupiny	77

VÝZVY A RIZIKA ROZVOJE INFORMAČNÍCH SYSTÉMŮ

Centralizace a shromažďování osobních dat	78
Riziko úniku dat a jejich zneužití.....	78
Digitální propast mezi skupinami společnosti.....	79

INFORMAČNÍ SYSTÉMY STÁTNÍ SPRÁVY

Příklady informačních systémů státní správy.....	79
--	----

STRUKTURA DAT V INFORMAČNÍCH SYSTÉMECH.....

Neuspořádaná data bez struktury	82
Uspořádaná data se strukturou	82

TABULKA – ZÁKLADNÍ POHLED NA DATA

Sloupec – třídící prvek v tabulce	84
Řádek – datový záznam v tabulce	84

TYPY DAT (DATOVÉ TYPY).....

Proč je důležité, aby v jednom sloupci tabulky byly všechny údaje stejného typu dat?.....	85
---	----

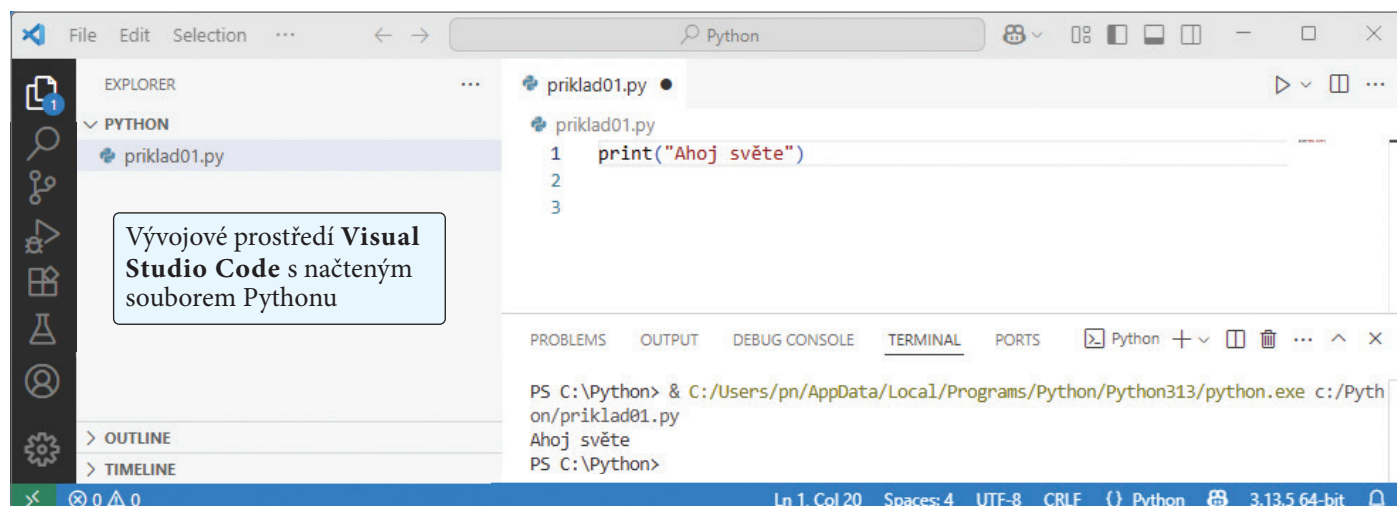
ZÁKLADNÍ PRINCIPY NAVRHOVÁNÍ	
DATOVÝCH TABULEK	87
Rozčlenění dat do jednotlivých sloupců	87
SPRÁVNÝ NÁVRH STRUKTURY TABULKY	88
ŘAZENÍ DAT V TABULCE	89
Seřazení tabulky podle počtu obyvatel	90
Víceúrovňové seřazení tabulky	90
FILTROVÁNÍ DAT	91
Aktivace filtru do záhlaví tabulky	91
Nastavení filtru	92
Sestavení složitějšího filtru	93
Filtr přes více sloupců	94
Zrušení filtru	94
VZORY A TRENDY V DATECH	95
Vzory v datech	95
VIZUALIZACE DAT PRO LEPŠÍ URČENÍ VZORŮ	98
Vizualizace tržeb pomocí grafu	99
Vizualizace tržeb pomocí podmíněného	
formátování	100
Trendy v datech	101
Vliv časového úseku na posouzení trendu	
v datech	103
ODHAD ZÁVISLOSTÍ V DATECH	105
Proč je důležité hledat, sledovat a odhadovat	
závislosti v datech?	106
Vizualizace závislostí v datech	106
BIG DATA (VELKÁ DATA)	109
KDE SE BEROU DATA?	109
CO JSOU TO VELKÁ DATA (BIG DATA)?	109
Jak velká jsou velká data?	109
Vlastnosti velkých dat (pravidlo 5V)	110
Zpracování velkých dat	110
Využití velkých dat	111
Využití velkých dat pro trénování modelů AI	113
DATABÁZE	114
Čím se liší databáze od běžné tabulky?	114
Výhody databází	114
JAK SE PRACUJE S DATABÁZÍ?	115
Relační databáze (SQL)	115
Nerelační databáze (NoSQL)	115
Jak programy a informační systémy pracují	
s daty v databázi?	115
ZÁKLADNÍ PRINCIP RELAČNÍCH	
DATABÁZÍ (SQL)	116
Klíče v tabulkách	117
Primární a cizí klíč	118
RELAČNÍ VZTAHY	119
Relační vztah 1:1 (jedna k jedné)	119
Relační vztah 1:N (jedna k mnohu)	120
Relační vztah M:N (mnoho k mnohu)	120
ER DIAGRAMY	123
Patky na propojovacích čarách ER diagramů	123
VÝVOJ INFORMAČNÍHO SYSTÉMU	128
ŽIVOTNÍ CYKLUS VÝVOJE SOFTWARE	
(INFORMAČNÍHO SYSTÉMU)	128
Plánování	128
Analýza	129
Design	129
Programování	129
Testování	129
Licencování a údržba	130
DESIGN INFORMAČNÍHO SYSTÉMU	130
SOUČÁSTI DESIGNU SYSTÉMU	130
Softwarová architektura a UX design	131
UI design	132
Jak to má, či nemá vypadat	135
Nástroje pro návrh uživatelského rozhraní (UI)	137
TECHNICKÉ ŘEŠENÍ INFORMAČNÍCH	
SYSTÉMŮ	140
FYZICKÁ ARCHITEKTURA KLIENT-SERVER	140
Jaké má architektura klient-server výhody?	141
LOGICKÁ TŘÍVRSTVÁ ARCHITEKTURA	
FRONTEND-BACKEND-DATA	141
Výhody třívrstvé architektury	142

Varianta 3: Python instalovaný v počítači + externí vývojové prostředí

U větších projektů anebo pokud to s programováním myslíte do budoucna opravdu vážně, neobejdete se bez výkonného a profesionálního vývojového prostředí, které nabízí řadu dalších funkcionalit, jež vývoj zefektivňují a zrychlují. Jedná se o **kom-binaci předchozí varianty instalace Pythonu spolu s externím samostatným vývojovým prostředím**. Vývojových prostředí je hned několik. Mezi kvalitní se řadí **Visual Studio Code**, jehož tvůrcem je společnost Microsoft. Lze je bezplatně stáhnout z následující stránky:



Visual Studio Code – instalační soubor: <https://code.visualstudio.com/>



Instalaci, nastavení a konfiguraci programu Visual Studio Code si ukazovat nebudeme, protože se jedná již o poměrně pokročilou aplikaci, vyžadující komplexní konfiguraci.

Výhody a nevýhody použití komplexních programovacích nástrojů

Výhody

- Robustní vývojové prostředí pro rozsáhlejší projekty.
- Nabízí automatizaci, pokročilé nástroje při práci s kódem, našeptávače, AI agenty, pomocníky v programování apod.
- Ocení je spíše zkušenější programátoři.

Nevýhody

- Náročnější nastavení a konfigurace.
- Využití potenciálu pokročilejších vývojových nástrojů předpokládá určitou minimální znalost technik vývoje a předchozí zkušenosti.

SOUVISLOSTI

Syntax není to samé co syntaxe

Pozor při použití podobných pojmů:

• **Syntax** (větná skladba / skladba věty) je soubor pravidel, která určují, jak se správně skládají slova do vět, aby měly smysl. Např. syntax „*Petr polévku jí*“ je méně obvyklá, zvláštně znějící.

• **Syntaxe** (v programování) je pravidlo, jak musí být napsán kód, aby ho počítač pochopil a mohl spustit. Je rozdíl mezi správnou syntaxí `print("Ahoj")` a nesprávnou `print "Ahoj"`, kvůli chybějícím závkám tento výraz počítač nepochopí a neprovede.

ZÁKLADY PROGRAMOVÁNÍ V JAZYCE PYTHON

Zvládnutí základů programování v jazyce Python je možné považovat za první krok k pochopení principů programování prostřednictvím tohoto i dalších programovacích jazyků. Pustíme se do toho.

SYNTAXE

◀ Při programování se setkáte s pojmem **syntaxe**. Co to je? **Syntaxe je soubor pravidel, která určují, jak mají být prvky programovacího jazyka (příkazy, instrukce apod.) uspořádány a zapisovány, aby byl kód správně počítačem pochopen.** Podobně jako v češtině existují pravidla gramatiky a pravopisu, tak při programování jsou pravidla programovacího jazyka pojmenována jako syntaxe.

Nemá smysl zde vypisovat seznam pravidel (resp. syntaktická pravidla) jazyka Python. Ta si osvojíte pouze opakovaným psaním kódu a programováním.

PROMĚNNÉ

Při programování se neobejdete bez tzv. **proměnných**, s nimi jsme již pracovali v kapitole o vývojových diagramech (viz **1. díl učebnice**, str. 144).

Proměnná je pojmenování konkrétního čísla nebo hodnoty. Proměnnou si zjednodušeně můžete představit jako jeden šuplík na velké stěně plné jiných šuplíků. Uvnitř má nějaký obsah (číslo, text apod.). Když potřebujete, můžete do šuplíku (proměnné) něco trvale nebo jen dočasně uložit, průběžně můžete obsah vyměnit za jiný nebo šuplík i zcela vyprázdnit. Technicky je proměnná **pojmenované místo v paměti počítače**, kam lze uložit nějakou hodnotu (číslo, text).



Základní pravidla proměnných v Pythonu

Pravidla	Správně	Nesprávně
1 Proměnné definujte malými písmeny. Jedná se o doporučení , nikoliv pravidlo. Velká písmena proměnných jsou povolena, není to vyloženo chybou, ale pro přehlednost kódu je vhodné psát názvy proměnných malými písmeny.	<pre>a = "Zdar!" b = 10</pre>	Možná, sice ne chybná, ale nevhodná varianta. <pre>A = "Zdar!" B = 10</pre>
2 Omezení speciálních znaků V názvech proměnné nepoužívejte speciální znaky (@, #, \$, %, & atd.) a mezery. Znak _ použít lze.	<pre>moje_prijmeni = "Novák"</pre>	<pre>moje prijmeni = "Novák"</pre> (v tomto případě je chybou meze- ra mezi znaky)
3 Více znaků proměnné Proměnná může mít i více znaků , může to tak být např. slovo.	<pre>vek_osoby = 10 mesto_narozeni = "Brno"</pre>	
4 Proměnná nesmí začínat číslicí. Může však číslici obsahovat nebo jí končit.	<pre>udaj1 = 10</pre>	<pre>1udaj = 10</pre>
5 Proměnná nesmí být shodná s názvy funkcí a příkazů Pythonu.		<pre>print = 10</pre>

Proč se vůbec používají proměnné?

Nebylo by lepší rovnou pracovat s jejich obsahem, když proměnná = obsah? Ne, nebylo. Proměnná, jak už název napovídá, může svůj obsah během provádění programu měnit. To znamená, že v jednom okamžiku může být například v proměnné **a** hodnota **1**, ale za chvíli to již může být hodnota **2**, poté hodnota **10** atd. A to je při sestavování programů nejen velmi výhodné, ale prakticky i nezbytné.

Typy a definice proměnných

Do proměnné lze vložit nějakou hodnotu zápisem s rovnítkem, např.: **a = 10**. Znamená to: vlož číslo **10** do proměnné **a**. Přitom zápis za rovnítkem vypadá podle toho, o jaký typ proměnné se jedná.

Příklad definice proměnné	Komentář
<pre>a = "Ahoj"</pre>	Do proměnné a vloží text. Proměnná a nyní obsahuje text Ahoj .
<pre>b = 10</pre>	Do proměnné b vloží číslo. Proměnná b nyní obsahuje číslo 10 .
<pre>c = 3.14</pre>	Do proměnné c vloží desetinné číslo. Proměnná c nyní obsahuje desetinné číslo 3,14 .
<pre>d = True e = False</pre>	Do proměnné d uloží stav True (pravda), do proměnné e stav False (nepravda).
<pre>f = [1, 2, 3, 4, 5]</pre>	Do proměnné f uloží seznam o prvcích 1, 2, 3, 4, 5 .
<pre>g = a</pre>	Proměnné g přiřadí obsah proměnné a . Proměnná g nyní obsahuje to stejné co proměnná a .

Výsledek: **Jmenuji se Filip Novotný a je mi 16 let. Za rok mi bude 17 let.**

Komentář:

Navazuje na předchozí program a doplňuje ho. Nově obsahuje i matematický výraz `vek + 1`. Tento výraz je zařazen jako jeden z parametrů funkce `print`. Jak je vidět, funkce `print` kumuluje výpis pevných textů (např. **Jmenuji se**), textových proměnných (např. `jmeno`, `prijmeni`), číselné proměnné (např. `vek`) a vzorce (`vek + 1`).

4 Kód:

```
jmeno = "Filip"
prijmeni = "Novotný"
vek = 16
print("Jmenuji se", jmeno, prijmeni)
print("Je mi", vek, "let")
print("Za rok mi bude", vek + 1, "let")
```

Výsledek:

Jmenuji se Filip Novotný
Je mi 16 let
Za rok mi bude 17 let

Komentář: Podobný program jako v předchozím případě, ale tentokrát je výpis údajů rozdělen do tří samostatných funkcí `print`, takže výpis je proveden na třech řádcích pod sebou.

Rozdělení na tři řádky by však bylo možné zařídit i vložením `\n` do jediné funkce `print`. Vyzkoušejte si tuto optimalizaci kódu sami.

5 Kód:

```
nazev_telefonu = "Aligátor"
cena_bez_dph = 1300
dph = 21
print("Název telefonu:", nazev_telefonu)
print("Cena bez DPH:", cena_bez_dph, "Kč")
print("DPH:", cena_bez_dph * dph / 100, "Kč")
print("Cena s DPH:", (cena_bez_dph * dph / 100) + cena_bez_dph, "Kč")
```

Výsledek:

Název telefonu: Aligátor
Cena bez DPH: 1300 Kč
DPH: 273.0 Kč
Cena s DPH: 1573.0 Kč

Komentář:

Smyslem programu je vypsát název telefonu, jeho cenu bez DPH, samotnou DPH a pak cenu telefonu včetně DPH.

Nejprve jsou nadefinovány proměnné `nazev_telefonu`, `cena_bez_dph` a `dph`. Následují čtyři řádky s funkcí `print`, které kombinují výpis pevně daného textu s proměnnou či výpočtem.

Řádek pro výpočet DPH obsahuje vzorec `cena_bez_dph * dph / 100`, který vyčíslí 21 % ze základní ceny 1300 Kč.

Řádek pro výpočet ceny s DPH obsahuje vzorec `(cena_bez_dph * dph / 100) + cena_bez_dph`. Při zápisu vzorce bylo nutné použít závorku. Celý vzorec, bez ohledu na to, jak je složitý, je umístěn mezi čárkami a tvoří tak jeden z parametrů funkce `print`.

Pozor, všimněte si, že výsledné hodnoty **DPH** a **Cena s DPH** obsahují **desetinnou tečku**, a to i v případě, že samotné výsledky (tj. čísla 273 a 1573) žádné desetinné číslo neobsahují. Je tomu tak proto, že při výpočtu **DPH** a **Ceny s DPH** bylo pro dělení použito jedno lomítko (/), které automaticky předpokládá dělení na desetinné číslo. Python si tudíž výsledek automaticky převedl na datový typ s desetinným číslem `float`. Pokud byste chtěli, aby byl výsledek dělení v celých číslech bez desetin (ovšem s nutným zaokrouhlením), pak by bylo nutné použít ve vzorcích dvojité lomítko (//).





Procvičování

Použití podmínky **if**, **else** (pokračování).

Kód (pozn.: stejný, jako je na počátku úkolu č. 2, str. 31):

```
cislo = 3
if cislo > 5:
    print("Číslo je větší než 5.")
else:
    print("Číslo je menší nebo rovno 5.")
print("A teď skončím.")
```



Úkoly:

Upravte kód tak, aby:

B)

Za předpokladu **nesplněné podmínky** vypsal kromě věty **Číslo je menší nebo rovno 5.** na samostatném řádku i vypočtenou číselnou hodnotu, **o kolik je hodnota proměnné cislo menší než 5.** Takže výsledek bude vypadat například takto:

Výsledek:

```
Číslo je menší nebo rovno 5.
O hodnotu 2
A teď skončím.
```

C)

Umístěte funkci, která vypíše obsah proměnné **cislo** prostřednictvím funkce **print("Číslo je", cislo)**, tak, aby se tento řádek zobrazil **při splnění i při nesplnění podmínce** před řádkem **A teď skončím.**

Výsledek:

```
Číslo je větší než 5.
O hodnotu 3
Číslo je 8
A teď skončím.
```

Použití podmínky **if**, **else**

Úlohy s komentáři na procvičení použití podmínky **if**, **else**:

- 1 Zadáním je vytvořit program, který se zeptá uživatele na heslo. Následně porovnáním s uloženým heslem vyhodnotí, zda je heslo správné, nebo špatné. Podle toho pak program vypíše odpovídající text.

Kód:

```
heslo = input("Zadej heslo: ")
if heslo == ("aD645"):
    print("Heslo bylo úspěšně zadáno. Vítejte, administrátore!")
else:
    print("Špatně zadané heslo, snad někdy příště.")
```

Výsledek (při zadání správné hodnoty aD645):

```
Zadej heslo: aD645
Heslo bylo úspěšně zadáno. Vítejte, administrátore!
```

Výsledek (při zadání jiné hodnoty než aD645):

```
Zadej heslo: ajw95
Špatně zadané heslo, snad někdy příště.
```



CYKLY

Cyklus slouží pro **opakování určité části programu**. Přitom je možné přesně určit, **jaká část programu se bude opakovat a kolikrát se bude opakovat**.

Opakovat je možné třeba jen jeden řádek programu, ale stejně tak i rozsáhlý blok o stovkách řádků.

V Pythonu existují dva druhy cyklů:

- **cyklus s předem daným počtem opakování,**
- **cyklus s podmíněným počtem opakování.**

Cyklus for – cyklus s předem daným počtem opakování

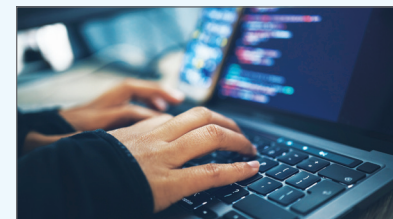
Cyklus **for** je cyklus s předem daným počtem opakování. To znamená, že je do předu známé, kolikrát se kód uvnitř cyklu zopakuje.



SOUVISLOSTI

Důležitost cyklů

Cykly jsou jedním z nejdůležitějších stavebních kamenů při programování. Zvládnutí jejich použití výrazně usnadní psaní efektivního a opakovaně použitelného kódu v Pythonu.



Použití cyklu s předem daným počtem opakování – **for**

for *prvek in range(x)* :
obsah cyklu

Příkaz **for** s funkcí **range(x)** zopakuje obsah cyklu **x**-krát. To znamená, že veškerý kód, který je umístěn jako odsazený pod řádkem s **for**, se provede **tolikrát, jaká hodnota je x**.

Prvek je dočasná proměnná, která se s každým provedením cyklu navyšuje. Jako prvek budeme v našem případě většinou používat proměnnou **i**.

Obsah cyklu může být libovolný, například pouze jeden řádek, ale třeba i stovky řádků kódu. Řádky s obsahem cyklu musí být **odsazeny zleva**, ideálně tabulátorem (**Tab**). Cyklus končí posledním odsazeným řádkem za řádkem **for**.

1 Výpis slova 5× za sebou.

Kód:

```
for i in range(5):
    print("Ahoj")
```

Ahoj

Ahoj

Ahoj

Ahoj

Ahoj

Výsledek:

```
Ahoj
Ahoj
Ahoj
Ahoj
Ahoj
```

Komentář:

Cyklus **for i in range(5)** : lze přechíst takto. Zopakuj obsah cyklu 5×. Přitom proměnná **i** bude na začátku cyklu automaticky rovna **0**. S každým průchodem cyklem se proměnná **i** automaticky **navýší o hodnotu 1**. Jakmile se cyklus provede 5×, program jej opustí a pokračuje dál v provádění následujících příkazů.

Uvnitř cyklu je v tomto případě funkce **print("Ahoj")**. To znamená, že 5× bude pod sebou vypsán text **Ahoj**.

2 Počítání do pěti.

Kód:

```
a = 1
for i in range(5):
    print(a)
    a = a + 1
print("Konec programu")
```

1...2...3...4...5

Výsledek:

```
1
2
3
4
5
Konec programu
```

Komentář:

První řádek **a = 1** definuje proměnnou **a**, které přiřadí hodnotu **1**.

Cyklus **for i in range(5)** : definuje, že obsah cyklu bude proveden 5×. Obsahem cyklu je:

print(a) – zobrazí obsah proměnné **a**. Ta má při prvním průchodu cyklem hodnotu **1**, při druhém průchodu cyklem hodnotu **2** atd. A to proto, že výraz na následujícím řádku zvýší při každém průchodu cyklem hodnotu proměnné **a** o **1**.

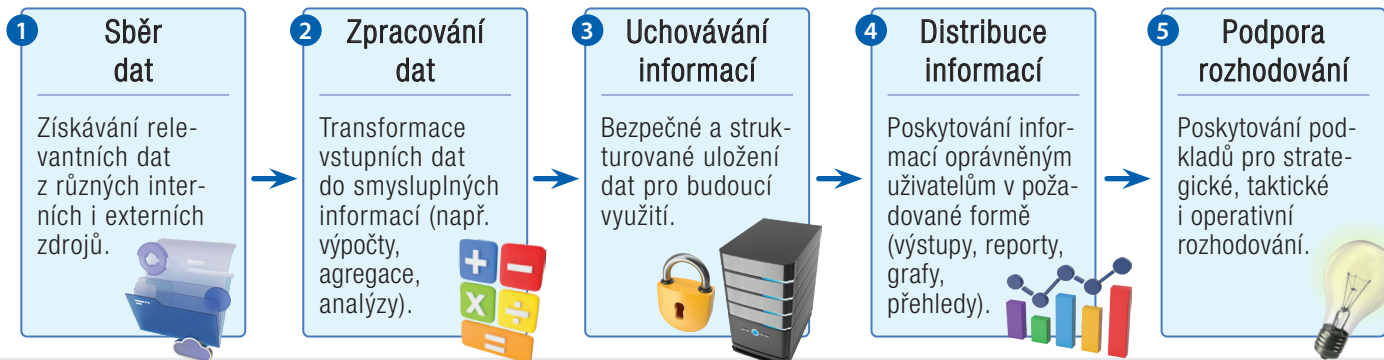
a = a + 1 – zvýší hodnotu proměnné **a** o **1**. Při každém průchodu cyklem se hodnota proměnné **a** zvýší o číslo **1**.

print("Konec programu") – funkce **print** již není odsazena zleva a není tak součástí cyklu. Text se proto vypíše na závěr po ukončení cyklu.

ÚČEL A PRINCIP FUNGOVÁNÍ INFORMAČNÍHO SYSTÉMU

Hlavním účelem informačního systému je **zefektivnit a zjednodušit nakládání s informacemi**. Princip fungování informačního systému lze znázornit následovně.

Podstata fungování informačního systému



SOUVISLOSTI

Výklad pojmu „informační systém“

Pojem **informační systém** často bývá prezentován a vnímán pouze jako označení pro software. Např. software **Pohoda** bývá označován jako **informační systém pro vedení účetnictví**, software **Bakaláři** jako **informační systém pro školství**. Proč tomu tak je?

Je informační systém opravdu pouze software, nebo je to ucelený soubor prvků, jak je v této učebnici uvedeno?

Informační systém je skutečně komplexní celek, který zahrnuje zmíněné složky (hardware, software, data, procesy, lidé).

Výrobci většího a komplexního softwaru však chtěli v minulosti ukázat, že jejich programy jsou natolik robustní a rozsáhlé, že dokážou uřídit celý informační systém nebo je lze nasadit ve všech jeho částech. Proto začali své programy označovat rovnou jako informační systémy. Přece jen označit vlastní software jako „informační systém“ zní daleko lépe než pouhý „program“. Tento trend se rychle rozšířil a postupně se vžil do povědomí uživatelů natolik, že dnes označení programu za informační systém nikoho nepřekvapuje. Striktně informaticky vzato je to ale špatně.

Zároveň však v praxi obvykle platí, že pokud výrobce svůj software označí jako informační systém, lze předpokládat, že jde o komplexní, rozsáhlý a robustní software, který v dané oblasti dokáže zpracovat ucelenou agendu.

HLAVNÍ VÝHODY INFORMAČNÍCH SYSTÉMŮ

- **Usnadňují lidem práci.** Informační systém zpracovává a vyhodnocuje data automatizovaně. To, co zvládne automatizovaný informační systém, by jinak musel dělat „ručně“ člověk nebo rovnou skupina lidí.
- **Zvyšuje efektivitu a produktivitu.** Toto souvisí s předchozím. Jestliže informační systém zastane práci za lidi, zvyšuje tak efektivitu a produktivitu.
- **Výrazně pomáhá zlepšovat rozhodování lidí.** Díky zpracovaným, uceleným a komplexním informacím z informačních systémů se mohou lidé lépe a rychleji rozhodovat, mohou pružněji reagovat na vzniklé situace, mohou lépe předvídat budoucí situace, lépe plánovat a odhadovat.
- **Centrální správa a sdílení dat.** Informační systémy spravují data na jednom místě. Díky tomu je možné je lépe „vytěžit“. Všichni uživatelé připojení k informačnímu systému mohou získat ucelené a komplexní informace bez ohledu na to, kde v organizaci se nachází. Navíc centrální sdílení dat omezuje nedorozumění, protože všichni lidé v jedné organizaci pracují pouze s jednými vždy aktuálními daty.
- **Lepší zabezpečení dat.** Díky centrální správě dat lze snadno nastavovat přístupová oprávnění. Tím se omezuje únik dat přes neoprávněné osoby. Data nejsou roztržena, lze je například centrálně zálohovat. Zabezpečení datového úložiště je nutné řešit jen na jednom místě, nikoliv na mnoha místech v organizaci.

Z ČEHO SE SKLÁDÁ INFORMAČNÍ SYSTÉM?

LIDÉ

Uživatelé, administrátoři a další osoby, které se systémem pracují.



DATA

Samotné informace a údaje, které systém zpracovává.

```
01011101101101100
10100101101010101
10101111000101101
10111101001101101
00010101001010011
```

PROCESY

Pravidla, postupy a opatření definující fungování, používání a správu systému.



SOFTWARE

Programové vybavení, tedy programy, aplikace a databáze.

```
def bmi(hmotnost, vyška):
    print("Váha BMI je: ", bmi)
    zadana_vyška = input("Zadejte svou
    zadana_hmotnost = input("Zadejte svou
    zadana_hmotnost = float(zadana_hmotnost)
```

HARDWARE

Fyzické vybavení, jako jsou počítače, servery, síťové prvky a další zařízení.





Praktický příklad:

Situace: Ukázku **přístupových práv** lze dobře demonstrovat na známkách v informačním systému školy. Informační systém školy obsahuje všechny známky všech žáků ze všech tříd. Z již uvedených důvodů ale musí mít každý uživatel přístup pouze k takovým datům, která nezbytně potřebuje. Naopak nemá přístup k datům, se kterými pracovat nepotřebuje.

Přístupová práva



Konkrétní žák(yně) má přístup pouze ke svým známkám u všech předmětů. Přidělené oprávnění umožňuje známky pouze číst.



Administrátor nebo ředitel školy má přístup ke všem údajům. Jeho oprávnění jsou neomezená.



Učitelka češtiny, která učí jak 1. A tak 1. B, má přístup ke známkám z češtiny u žáků 1. A i v 1. B. Přístupová práva jí umožňují známky číst, upravovat, vkládat i mazat.



Učitel matematiky ve třídě 1. A má přístup pouze ke známkám z matematiky jen u žáků v 1. A. Jeho oprávnění je známky číst, upravovat, vkládat a mazat.

Příjmení a jméno	Třída	Matematika	Český jazyk	Fyzika
Málková Veronika	1. A	1, 2, 1, 3, 4	2, 1, 1, 3	3, 4, 1, 1
Novák Radim	1. A	3, 2, 4, 1	1, 1, 1	2, 1, 3
Vrbický Jaroslav	1. B	2, 5, 4, 3	2, 1, 1	1, 2, 3, 2, 1
Zlámalová Petra	1. B	1, 1, 2	2, 2, 1, 3	1, 1, 3, 1, 2



Třídní učitel třídy 1. B má přístup ke všem známkám žáků 1. B, a to ze všech předmětů. Jeho oprávnění je známky pouze číst. Nemůže je mazat ani upravovat.

Uživatelské role a uživatelské skupiny

S většinou informačních systémů pracuje obvykle více uživatelů, kteří mají naprosto stejná oprávnění. Příkladem může být e-shop, ve kterém pracuje více skladníků majících totožná práva, každý z nich se ale přihlašuje pod svým účtem. Nebo prodejna, kde pracuje více prodavaček, které se do pokladního informačního systému také přihlašují pod svým uživatelským účtem, ale mají stejná oprávnění.

Proto při správě uživatelů existují v informačních systémech takzvané **uživatelské role**, případně **uživatelské skupiny**.



ZAPAMATUJTE SI

Uživatelská role (skupina) je pojmenovaný balíček přístupových práv odpovídající určité funkci nebo zařazení (např. účetní nebo administrátor), který se přiděluje uživatelům, aby se nemusela nastavovat oprávnění pro každého zvlášť.

Princip uživatelských rolí nebo uživatelských skupin spočívá v tom, že přístupová práva se nastaví pro tuto roli či skupinu a uživatel se pak jen k takové roli či skupině přiřadí.

Víceúrovňové seřazení tabulky

1 Klepněte na tlačítko **Seřadit**. Tím se aktivuje stejnojmenné okno.

2 V okně **Seřadit** nastavte v prvním řádku řazení podle **Názvu okresu**.

3 Tlačítkem **Přidat úroveň** přidáte druhý řádek. V něm nastavte řazení podle sloupce **Počet obyvatel celkem**.

4 Pořadí řazení pro oba řádky nastavte od A do Z, resp. **od nejmenšího k největšímu**.

5 Klepnutím na **OK** bude tabulka seřazena.

Tabulka bude seřazena podle první nastavené úrovně, to je v našem případě sloupec **Název okresu**. Jestliže program v tomto sloupci objeví shodné hodnoty, pak začne řádky řadit podle druhé úrovně, kterou je v našem případě sloupec **Počet obyvatel celkem**. Tím bude dosaženo požadavku – **tabulka zobrazí seznam okresů podle abecedy a v rámci nich seznam obcí řazených podle počtu obyvatel**.

FILTROVÁNÍ DAT

Při práci s rozsáhlými daty často uživatel potřebuje pracovat pouze s jejich určitou částí. Nebo potřebuje zobrazit jen taková data, která splňují určité podmínky. Pro tyto účely se používají tzv. **filtry**.

Filtr je podmínka, která se aplikuje na tabulku, a z původních dat **zůstanou zobrazena pouze taková, která vyhovují zadané podmínce**.

Aktivace filtru do záhlaví tabulky

Aby bylo možné začít filtry používat, je nutné je aktivovat v záhlaví tabulky. Stačí se postavit do řádku se záhlavím tabulky a klepnout na tlačítko **Filtr**.



SOUVISLOSTI

Filtr data nemaže!

Pozor, **filtr omezuje jen zobrazení dat, nemaže je!**

Pokud si necháte zobrazit filtrovaná data, tabulka jako celek je stále obsahuje v plném rozsahu, pouze nejsou vidět.

Filtrování dat – aktivace filtru

2 Na záložce **Data** klepněte na tlačítko **Filtr**.

1 V tabulce se postavte do řádku se záhlavím (je jedno, do jakého sloupce).

3 U záhlaví každého sloupce se zobrazí malé ikony šipek. Tím jsou filtry připraveny k použití.

Filtr připraven k použití.

JAK SE PRACUJE S DATABÁZÍ?

Abychom s databází mohli vůbec pracovat, potřebujeme k tomu speciální software – **DBMS (Database Management System, česky systém řízení báze dat)**. Ten slouží jako prostředník mezi uživatelem (nebo aplikací) a samotnými daty.

Databáze se zpravidla dělí na dvě hlavní skupiny – **relační databáze** a **nerelační databáze**.

Relační databáze (SQL)

Ukládají data do klasických **tabulek**, které se skládají z **řádků a sloupců** (podobně jako v tabulkovém procesoru). Tyto tabulky jsou mezi sebou většinou propojeny pomocí **vztahů**. Pro komunikaci s nimi se používá **jazyk SQL (Structured Query Language)** a jeho příkazy (podobně jako například u programování).

Nejznámější zástupci: **MySQL, PostgreSQL, Oracle, SQLite**.

Tento typ databází je v současnosti **nejpoužívanější pro většinu systémů** a aplikací, které znáte z běžného života.

Nerelační databáze (NoSQL)

Tyto databáze ukládají data v jiných, neuspořádaných formátech, často bez pevné struktury a formátu (například jako dokumenty, grafy, tabulky, obrázky apod.). Tento typ databází pracuje se specifickou logikou zpracování a je určen pro obrovské objemy neustále se měnících dat (big data). Typicky se používají např. pro velké sociální sítě.

Nejznámější zástupci: **MongoDB, Redis, Apache Cassandra**.

Tento typ databází je určen spíše pro **speciální a velké počítačové systémy**, jež pracují s velkým objemem dat. Při běžné práci v klasických informačních systémech se s nimi většinou nesetkáte.

Jak programy a informační systémy pracují s daty v databázi?

Běžný program, který nepracuje s databází, spravuje data sám. Takže sám volí způsob, jakým data bude ukládat a jak je bude načítat.

Pokud ale program pracuje s databází, pak se již o organizování dat přímo nestará. Program prostě jen **pomocí SQL příkazů** pošle pokyn (dotaz) do databáze, ta pokyn zpracuje a zpět programu pošle výsledek. Databáze tak funguje jako jakýsi podprogram, který je určen pro obhospodařování dat.

SOUVISLOSTI

Relační databáze všude kolem nás

Aniž si to možná uvědomujete, s relačními databázemi pracujete denně při běžných činnostech s počítačem, mobilním telefonem nebo tabletem. Jak? Například seznam kontaktů ve vašem mobilním telefonu pravděpodobně využívá některou z SQL databází. Stejně tak poštovní klient pro odesílání a příjem e-mailů zcela jistě využívá SQL databázi.



SOUVISLOSTI

Fungování databází

Principy, na nichž pracují databáze, naznačuje následující video.

Vítejte ve světě SQL:
<https://youtu.be/LJx3Xq8UuMI>



2:04

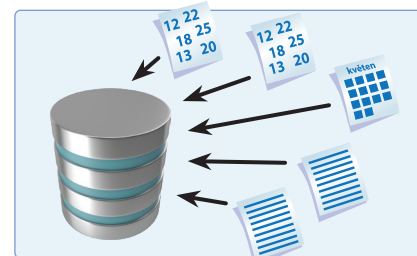
Spolupráce programu s daty v databázi



program

dotaz / pokyn

výstup dotazu / data



databáze

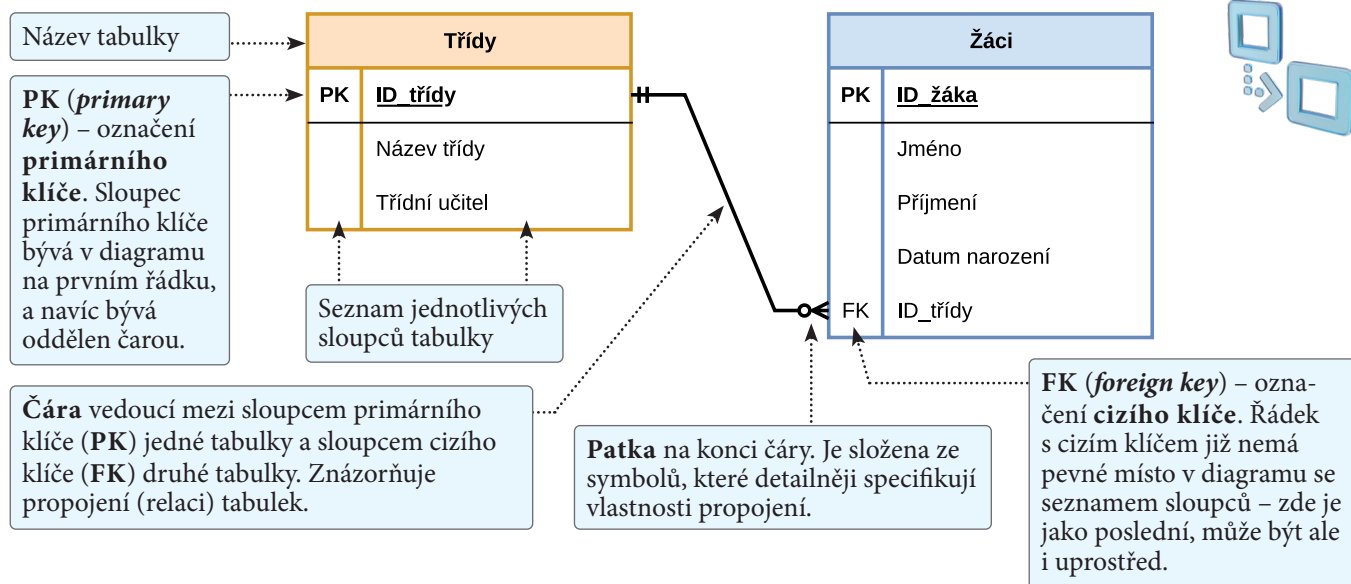
Dejme tomu, že budete pracovat s programem, který zobrazuje naměřené meteorologické údaje za posledních 10 let:

- V programu zadáte (nebo naklikáte) požadavek na to, jaké meteorologické údaje potřebujete zobrazit.
- Program pošle dotaz (pokyn) do databáze. Databáze obsahuje všechna data přehledně uspořádaná do databázových tabulek. Data se do databáze dostala z měření a čidel přístrojů.
- Databáze zpracuje požadavek, vyhledá konkrétní data a zašle je programu jako výstup dotazu.
- Program data z databáze přijme a ve vhodné podobě je zobrazí uživateli.

ER DIAGRAMY

Při návrhu databází používají profesionálové speciální druh diagramů, které znázorňují propojení mezi tabulkami a jejich základní vlastnosti. Těmto diagramům se říká **ER diagramy** (*entity-relationship* – **entitně vztahový diagram**).

ER diagram



Zobrazený ER diagram znázorňuje naprosto stejnou situaci, jako je schéma s propojením tabulek relačním vztahem **1:N** na straně 120. Na první pohled je zřejmé, že ER diagram je úspornější a obsahuje všechny základní údaje pro sestavení struktury databáze. Při návrhu databází vůbec nemusíme pracovat s vlastními daty, proto je ER diagram ani neobsahuje.

Důležité je, aby bylo možné z diagramu vyčíst rozvržení tabulek, výčet sloupců, definici klíčů a propojení tabulek.

Patky na propojovacích čarách ER diagramů

Čáry propojující tabulky v ER diagramech databází mají na obou koncích specifické symboly, kterým říkáme **patky**. Patka se obvykle skládá ze dvou symbolů umístěných těsně vedle sebe.

Tyto symboly definují dvě základní vlastnosti vztahu – **kardinalitu** a **potencialitu**.

Kardinalita

Určuje **maximální počet záznamů**, které se mohou k protější tabulce vázat. Tento symbol je **umístěn vně čáry**, tj. vždy blíže k hraně tabulky.

Může mít následující podobu:

- **Jedna svislá čárka (—+)** – znamená **právě jeden**.
- **Tři rozbíhavé čárky** (tzv. vraní patka —<) – znamená **mnoho**.

Potencialita

Definuje, zda je vztah **povinný**, nebo **volitelný**. Tento symbol je umístěn **uvnitř čáry**, tj. vždy dál od hrany tabulky, těsně vedle symbolu kardinality.

Může mít následující podobu:

- **Kroužek (—○)** – znamená **volitelný vztah**. Záznam v protější tabulce **nemusí** mít v této tabulce žádný odpovídající řádek, ale stejně tak jej (volitelně) mít může.
- **Svislá čárka (—+)** – znamená **povinný vztah**. Záznam v protější tabulce **musí** mít v této tabulce vždy alespoň jeden odpovídající řádek.

SOUVISLOSTI

Způsob označování čar pomocí symbolů patek není jedinou formou, jak definovat kardinalitu a potencialitu vztahů mezi tabulkami. Existují ještě další způsoby (například číselné a textové označení nad čarou). Nebuďte proto překvapeni, pokud se setkáte s **odlišnou vizuální podobou ER diagramů** či **odlišným značením vztahů mezi tabulkami**.

SOUVISLOSTI

Různé podoby ER diagramů

ER diagramy SQL databází se dnes v profesionálním světě navrhují prakticky výhradně prostřednictvím k tomu určených programů. Jejich výhodou je, že v diagramech lze provádět změny a diagram jako celek se jim dynamicky přizpůsobí. Některé z programů pro návrh ER diagramů jsou dostupné zdarma každému. Můžete si vyzkoušet například oblíbený program **draw.io**.



Draw.io – tvorba ER diagramů:

<https://app.diagrams.net/>



Podoba ER diagramů může být různá. Některé diagramy propojují čarami pouze tabulky, nikoliv konkrétní řádky. Jiné diagramy obsahují kromě názvů sloupců i jejich datové typy apod.

SOUVISLOSTI

Co vše obnáší tvorba softwaru?

Tvorba softwaru nespočívá jen ve vlastním programování. Mnoho věcí je nutné vyřešit ještě před tím, než programátor začne vůbec programovat, a řadu věcí je nutné vyřešit poté, co je program dokončen. Samotné programování je paradoxně pouze jednou z činností v celkovém životním cyklu vývoje softwaru či informačního systému. Tvorba softwaru je vlastně „stavba programu“ podle předem daného a dobře promyšleného plánu.

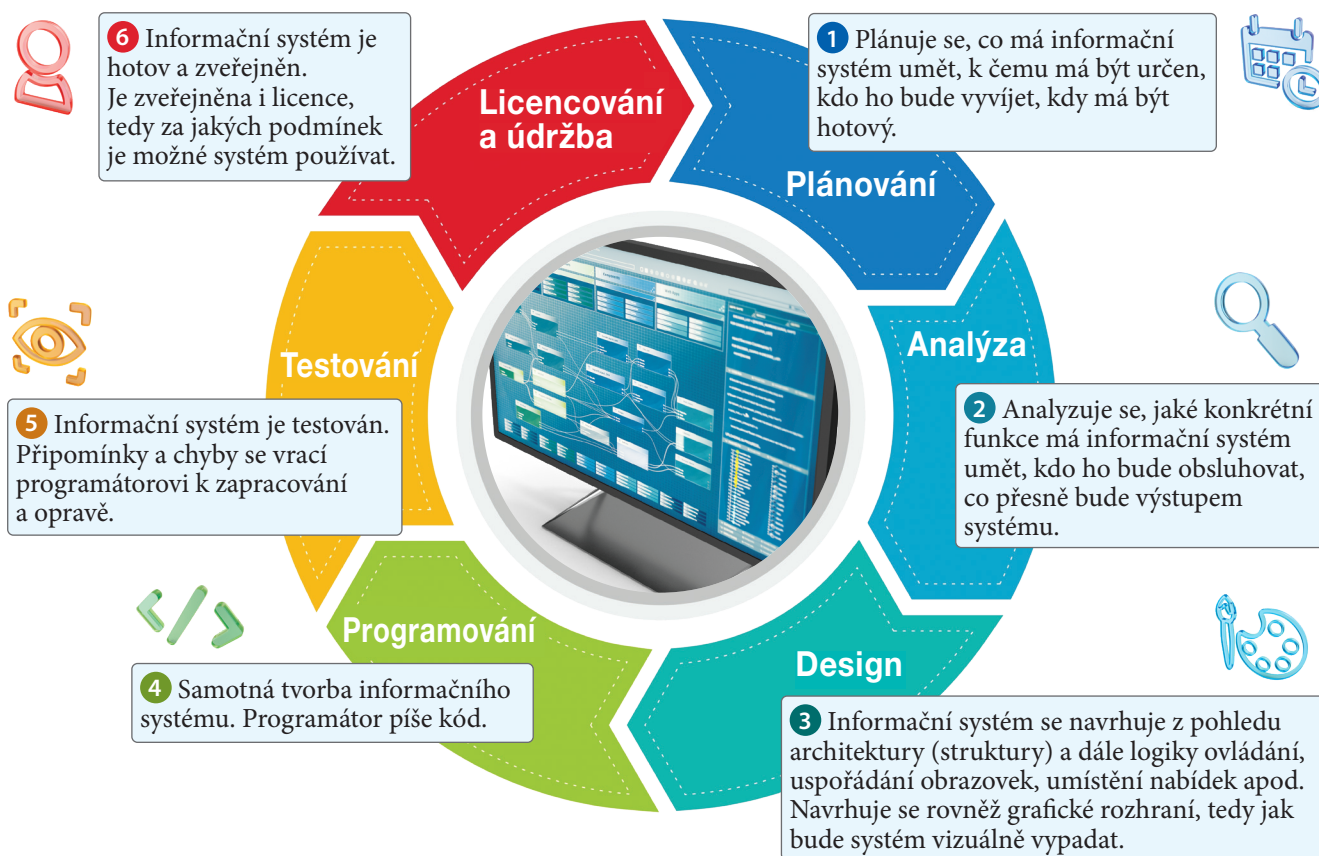
VÝVOJ INFORMAČNÍHO SYSTÉMU

V této fázi již dokážete naprogramovat vlastní program v programovacím jazyce a znáte princip ukládání dat do databáze. To jsou základní předpoklady pro tvorbu nových programů a informačních systémů. Nyní se proto zaměříme na to, jak se software a informační systémy obecně vytváří.

ŽIVOTNÍ CYKLUS VÝVOJE SOFTWARE (INFORMAČNÍHO SYSTÉMU)

Proces vývoje softwaru (nebo informačního systému) lze obecně popsat prostřednictvím tzv. **životního cyklu vývoje softwaru**, angl. **SDLC (software development life cycle)**. Jedná se o standardizovaný postup složený z několika po sobě jdoucích kroků, kdy je jasně dáno, co se ve kterém kroku má odehrát a jaký je výstup z každého kroku.

Životní cyklus vývoje softwaru – informačního systému



SOUVISLOSTI

Životní cyklus vývoje softwaru

K lepšímu pochopení jednotlivých fází životního cyklu vývoje softwaru (procesu SDLC) vám pomůže následující video.

Životní cyklus vývoje softwaru:

https://youtu.be/SaCYkPD4_K0



12:30
české titulky

Plánování

Jedná se o úplný začátek práce na informačním systému. Fáze plánování pokládá základy celého informačního systému:

- **plánují se cíle informačního systému** – tedy k čemu bude informační systém určen, pro koho bude určen, co má umět, jaký zhruba bude jeho výstup, jaká bude ekonomika systému a jak se bude financovat;
- **odhaduje se finanční náročnost (rozpočet)**, stanovuje se **časový harmonogram (milníky)** a **plánují se lidské zdroje**, tj. kolik lidí a v jakém složení bude na projektu pracovat.

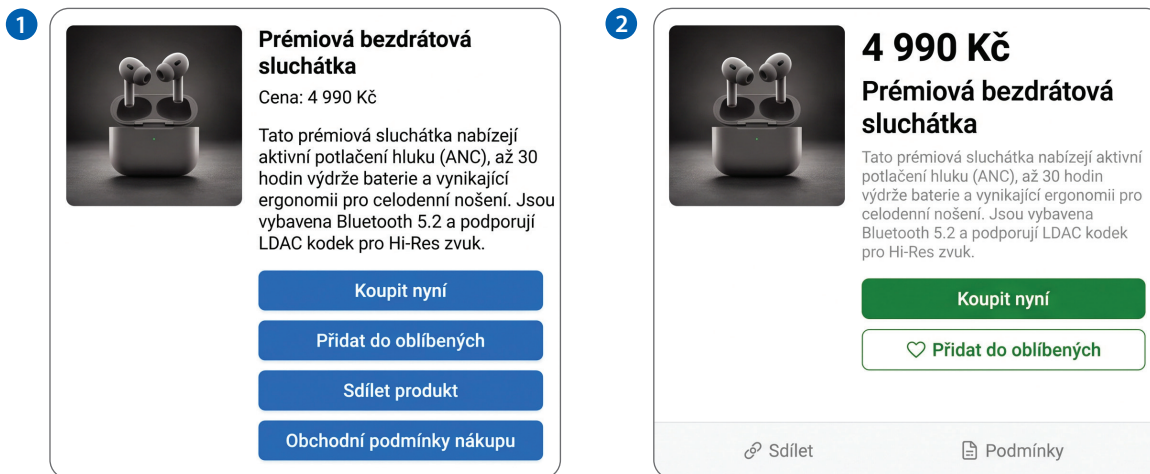
Výstup z fáze plánování: Rámcový popis informačního systému, časový harmonogram a hrubý plán realizace informačního systému.

Vizuální hierarchie

Oko uživatele musí **na první pohled poznat, co je na obrazovce nejdůležitější**. Toho lze docílit velikostí prvků, jejich výrazností, barvami a umístěním. Čím je akce důležitější (např. tlačítko **Koupit nyní** v e-shopu), tím více musí v okně nebo na stránce dominovat. Naopak prvky, které se používají zřídka nebo mají okrajový význam, by měly být vizuálně upozaděny.

Ukázka principu vizuální hierarchie

Na následujícím obrázku je jedna položka e-shopu zobrazena ve dvou provedeních.



- 1 Obrázek vlevo zobrazuje informace o ceně sluchátek a jejich popis ve stejném formátu písma. Rovněž všechna tlačítka jsou na stejné vizuální úrovni, takže **uživatel na první pohled nevidí, které tlačítko je důležitější** (tlačítko pro nákup) a které je méně podstatné (například tlačítko pro sdílení produktu). Takto zobrazený produkt v e-shopu **nerespektuje vizuální hierarchii**.
- 2 Obrázek vpravo naopak jasně a strukturovaně odděluje důležité informace od méně důležitých. Na první pohled **je jasné, jaká je cena produktu**, což je údaj, který je na e-shopu klíčový. Na první pohled **je rovněž jasné, na jaké tlačítko má uživatel klepnout, pokud si chce produkt koupit** (což je na e-shopu logicky nejzásadnější operace). Naopak popis produktu (jakožto druhotná informace) je napsán šedým, méně výrazným písmem. Stejně tak odkazy **Sdílet** nebo **Podmínky** jsou méně nápadné, protože jsou v daném okamžiku méně významné. **Takto zobrazený produkt v e-shopu respektuje vizuální hierarchii**.

SOUVISLOSTI

Jakobův zákon uživatelského zážitku

Uživatelé tráví většinu času na jiných webech a v jiných aplikacích. To znamená, že očekávají, že váš software bude fungovat podobně jako ty, které už znají odjinud (např. nákupní košík bude vždy vpravo nahoře, nastavení uživatele bude pod ikonou uživatele apod.). **Podříďte se osvědčeným zvyklostem a nevymýšlejte nové uživatelské standardy tam, kde to není nutné.**

Poznámka: **Jakob Nielsen** je dánský informatik a vysokoškolský učitel.

10 pravidel pro UI design webu podle Jakoba Nielsena:

<https://youtu.be/ETGtsleVOpE>



1:00
české titulky

Jasnost a srozumitelnost (*clarity*)

Uživatel by **nikdy neměl přemýšlet, co která ikona nebo tlačítko znamená**. Texty musí být stručné a výstižné. Například na tlačítku pro uložení změn je vhodnější použít text **Uložit změny** než vágní (nekonkrétní) text **OK**.

Zpětná vazba (*feedback*)

Na každou akci uživatele musí systém nějak reagovat. Například otevřít okno, vypsát hlášku, vygenerovat text apod. To se týká dokonce i nejzákladnějších prvků. Například když uživatel klikne na tlačítko, mělo by se ideálně vizuálně „promáčk-nout“, nebo pokud lze očekávat delší odezvu, pak by se mělo ukázat typické načítací kolečko. **Pokud totiž uživatel provede akci a nestane se nic** (i když program bude ve skutečnosti pracovat a připravovat reakci), **uživatel znejistí, zda jeho požadavek počítač zaregistroval**, a to může vést ke zmatkům (uživatel pak třeba opakovane mačká tlačítka nebo z programu odejde).

Přístupnost (*accessibility*)

Software by měl být **ovladatelný každým uživatelem, včetně lidí se zrakovým nebo jiným omezením**. To znamená, že by v programu mělo být možné nastavit dostatečný kontrast textu vůči pozadí, nastavit zvětšení písma nebo ovládat program jen pomocí klávesnice.