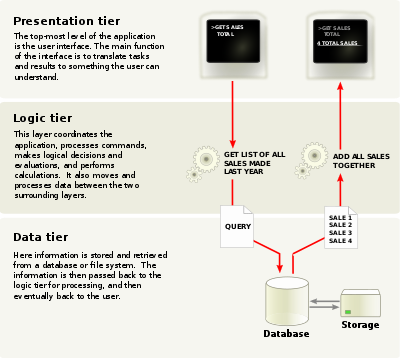
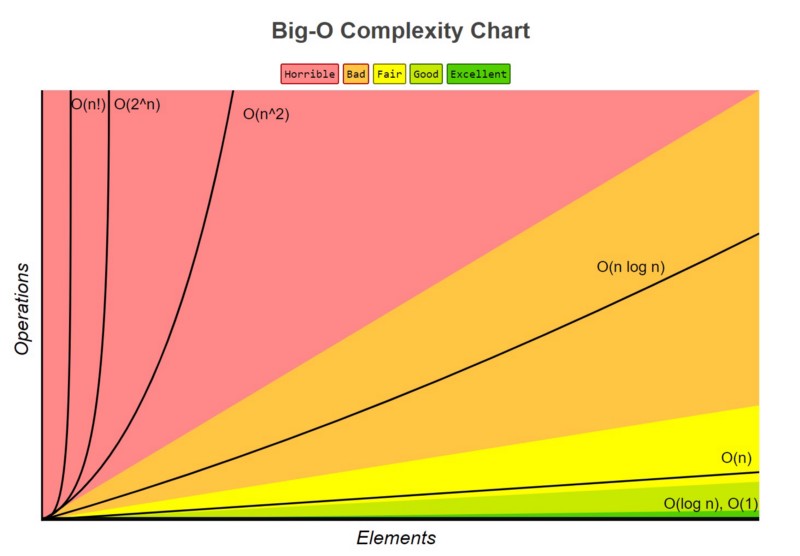
1. Otázka
   * Halda, zásobník
   * Hodnotové, referenční datové hodnoty
   * Ukázka v programu kde se co ukládá a jak se co odkazuje
   * Indexace
   * Garbage collector
2. Otázka
   * Co je to graf (hrana/vrchol)
   * Orientovaná, neorientovaná, násobná, rovnoběžná, smyčka
   * Graf úplný, izolovaný, jednoduchý, multigraf, prostý, pseudo graf
   * Matice sousednosti (vrchol/vrchol) incidence (vrchol/hrana)
   * Prohledávaní … slepé (BFS,DFS), heuristic (A\*, Best first search, beam search), náhodné
   * Bubble sort, inserction, rozděl a panuj … marge sort, quick sort
3. Otázka
   * Rekurze … dělíme na přímou(A->A)/nepřímou(A->B,B->A) a počet volání jednou(lineární)/víckrát(stromová) … ukázka faktoriálu
   * Brute force … prochází všechny možnosti s délkou klíče exponenciálně roste čas
   * Heuristyky … používá dodatečnou logiku, rychlejší jak brute force, nemusí uspět, výsledek pouze optimální
     1. Generický algoritmus … na základě předem daných kritérii rozhoduje jakou hodnotu upřednostnit (dokáže najít kvalitní řešení problému)
     2. Metoda lokálního hledání … vyhodnocují jen své nejbližší okolí, vydávají se tím směrem, který je pro ně výhodný, nemohou se vracet, nenajdou globální max
     3. Iterativní metoda … využívá postupného hledání řešení vs stále se zužující oblasti (dostává se z optimálního řešení k lepšímu)
   * Deterministický/nedeterministický … v deterministickém algoritmu může dojít u stejného vstupu k dvou různým výstupům opak determinismus (existuje alespň jeden výsledek vyhovující, pravděpodobnost výběru dalších kroků)
4. Otázka
   * Anonymní metoda (kus kódu který fungují jako metody, ale nemají jméno fungují podobně),
     1. dělí se na dvě skupiny Expresion lambda (input => expression), Satement lambda (input => { statements }),
     2. var, Func<params, output>, Action<params> (nevraci/void hodnoty), Predicate<params> (vraci bool) … delegate ukazuje na anonimni metodu
     3. linq … list.OrderBy(x => x).Where(x=>x>50).ToList();
     4. Obsah obrázku text, snímek obrazovky, software, displej

        Popis byl vytvořen automaticky
   * delegát je referenční datový typ (ukazatel na metodu přes proměnou), dosáhneme čistějšího kódu
     1. řešíme tři kroky … deklarace, nastaveni reference, vyvolání delegáta, multicast vtvoříme dva delegáty a pote je spojíme do jednoho (můžeme volat vice metod v jedné proměnné)
     2. Obsah obrázku text, snímek obrazovky, software, Multimediální software

        Popis byl vytvořen automaticky
5. Otázka
   * Jsou způsoby návrhu řešení problému vynalezený architektem Christopherem Alexandrem
   * MVC (Model, View, Controller) … dlouhodobá udržitelnost kódu, využívá se i v webovkách, dohromady tvoří jeden celek
     1. kde Model zajišťuje práci s daty, většinou komunikaci s databází. View obstarává pohled uživatele na data. Poslední zmíněnou částí je Controller, jehož role je reagovat na změny dat a události vyvolané uživatelem. Na jeho základě je aktualizován View
     2. 
   * Monolitická Architektura … obtížná údržba (jednotlivé skupiny funkcí nejsou separovány), obtížná přenositelnost aplikace mezi různými platformami, obrovský masivní kus kódu plný závislostí, výhody snadný vývoj snadné zajištění ochrany
   * Multi-tier architektura … aplikace je rozdělena do několika vzájemná komunikujících vrstev, nejznámější je three-tier architektura.
     1. 
     2. Výraz layer a tier zde nejsou synonymy … layer (logická struktura nejčastěji software) kde tier (fyzická struktura tvořena hardwarem)
   * P2P(peer to peer) … skupina uzlů které všechny mají stejnou roli vykonávají stejnou funkci, neexistuje zde žádný centrální prvek (všechny prvky pracují jako přímač a vysílač zdrojů), náchylná na šíření virů, složitá administrace obnova dat backup
     1. jsou zapotřebí jen dva uzly
     2. další uzly se mohou kdykoli připojit
     3. uzly se mohou kdykoli odpojit
   * klient server … operuje na vtahu mezi dvěma programy, klient využívá služby serveru které poskytuje, (např. SMTP, http, DNS, apod.).
     1. Jednodušší správa, klient je závislý na centrální autoritě, nelze jednoduše přes server množit malware,
     2. nevýhody klent může čekat dlouho na odpověď nebo ji úbec nedostat
6. Otázka
   * Asymptotická (zabývá výkonností u programu, který může pracovat s neomezeným počtem dat) složitost programu je způsob, kterým se dá určit kolik prostředků bude potřeba k dosažení cíle.
     1. (neomezeně paměti) Časová složitost určuje dobu, kterou procesor vyžaduje k vykonání úlohy, tyto složitosti rozlišují tím, jak doba vykonání roste se změnou množství dat.
     2. (neomezeně času) kolik operační paměti bude využito, jak paměť vykonání roste se změnou množství dat
   * O (1) – konstantní (přístup k listu), O(log n) – logaritmická(for cyklus který se násobí/dělí po každé iteraci), O(n) – lineární(přístup prvku v linked listu, Třeba for cyklus, který přičítá po každém proběhnutí), O(n log n) - kombinace lineární a logaritmické (dva for cikly jeden přičítá druhý násobí), O(n^2) – kvadratická (dva for cykli které přičítají, plním dvojrozměrné pole), O(2^n) – exponenciální (fobinaciho posloupnost), O(n!) – faktoriálová (bruteforce, obchodní cestující, for ciklus běží faktoriálněkrát)
   * Značení:
     1. Big O (big ou) … nejhorší možný výsledek
     2. Big Ω (big omega) … nejlepší možný výsledek
     3. Big Θ (velká théta) … průměrný výsledek
     4. 
7. Otázka
   * Obsah obrázku text, snímek obrazovky, diagram, řada/pruh

     Popis byl vytvořen automaticky
     1. Primitivní (hodnotové) definovaná velikost a paměťová kapacita, neprimitivní (referencové) odkazují na oběkt (krom stringu programátorem vytvořené) začínaj se velkým písmenem nemají pevnou paměťovou kapacitu
     2. Byte 8-bit (0-255 U), sbyte 8-bit (-128;127 S), short 16-bit (-2^8;2^8-1 S), ushort 16-bit (0;2^16-1 U), int 32-bit (S), uint 32-bit (U), long 64-bit (S), ulong 64-bit (U)
     3. floating point: float 32-bit (S) 10f, double 64-bit 10.0, decimal 128-bit (S)
     4. bool … true, false
     5. char 16-bit Unicode … U +0000 a6 U+ffff
   * generika (obecný datový typ) hodí se jako vstupní proměnné do metod či pro listy,
     1. Obsah obrázku text, snímek obrazovky, Písmo, software

        Popis byl vytvořen automaticky
   * Chceme dát na výběr z pevně daných možností
     1. Obsah obrázku text, snímek obrazovky, software, Multimediální software

        Popis byl vytvořen automaticky
     2. Jsou očíslované od 0 do n jinak pondělí=5
   * Struktury jsou hodnotové datové typy, nemůžeme od nich nebo je dědit, nelze změnit defaultní konstruktor
     1. Obsah obrázku text, snímek obrazovky, software, Písmo

        Popis byl vytvořen automaticky
   * Anotace označuje a popisuje chování tříd, metod, parametrů a vlastností. Dovoluje přidávat metadata které budou následně použity v procesu kompilace nebo za běhu programu
     1. **[Obsolete(“use A()”)]** function, **[NotNull]/[CanBeNull]** param, **[Serializable]** class, [**Required]** property (used with Serializable), **[Range(0, 5, ErrorMessage = "error")]** property, **[StringLength(30, MinimumLength = 0,ErrorMessage = "Error")]** property
     2. Obsah obrázku text, snímek obrazovky, Písmo

        Popis byl vytvořen automaticky
   * Operátory … aritmetické, porovnávací, logické, bitové, ternary(?:), unary(++,--)
8. Otazka
   * Můžeme implementovat více jak jedno rozhraní, při dědění třída předává i své rozhraní, měl by začínat s „I“ místo class se použije interface, říká jaké specifické metody by měla třída implementovat (už nás nezajímá jak toho bude dosaženo)
     1. Obsah obrázku text, snímek obrazovky, Písmo, řada/pruh

        Popis byl vytvořen automaticky
   * Parent class, child class, můžeme přepoužít atributy a metody, není povolena mnohonásobná dědičnost, (sealed class) … neděditelné, třídu dědíme za pomocí „:“, protected vlastnost
     1. Obsah obrázku text, snímek obrazovky, software, Multimediální software

        Popis byl vytvořen automaticky
   * Známe ji taky jako compile-time (nebo static) polymorphismus protože všechny overloaded metody jsou vyhodnoceny ve chvíli kdy se aplikace kompiluje.
     1. Obsah obrázku snímek obrazovky, text, Písmo

        Popis byl vytvořen automaticky
   * Známe ji také jako runtime (nebo dynamic) polymorphismus protože typ objektu není předem jistý až za běhu programu. A z toho důvodu jaká z implementací metody se použije je jasné až za běhu programu
     1. Obsah obrázku text, snímek obrazovky, software, Písmo

        Popis byl vytvořen automaticky
9. Otázka
   * Kontrola datových typů, validace vstupů, zálohování dat … data musí zůstat konzistentní abychom zajistili správné fungování aplikace
   * validace vstupů, ověřování identity, šifrování, upadávání softwaru a balíčků
     1. bráníme s proti (SQL injection, Cross-site scripton, DoS, neoprávněný přístup k datům, útok na heslo)
   * python (logging)
     1. logging.basicConfig(format='%(asctime)s %(levelname)s: %(message)s', datefmt='%Y-%m-%d %H:%M:%S',filename='example.log', level=logging.DEBUG)
   * kontorlujem npař: datový typ, rozsah hodnoty, délka řetězce, platnost, jiné podmínky(regex), za pomocí: if/else, (assert condition, error), re.match(r„“,value), try/except
   * try/except/else/finally, ExceptionGroup(„text“,[excpetions…]), class Myexception(Exception): init == super().init(message)
10. Otázka
    * import mysql.connector, USERNAME: root, PASSWORD: student, HOSTNAME: 127.0.0.1, PORT: 3306
      1. Obsah obrázku text, snímek obrazovky, software, Písmo

         Popis byl vytvořen automaticky
      2. Obsah obrázku text, snímek obrazovky, Písmo

         Popis byl vytvořen automaticky
    * Repository je podobná jako DAO, ale pojmenováním metod je blíž kolekci. Dokonce může mít DAO uvnitř sebe a využívat ho. Typické je třeba přetížení hranatých závorek [] apod.

Table Gateway pattern se používá pro práci s celými tabulkami v databázi. Tento vzor vytváří objekt pro každou tabulku v databázi a poskytuje metody pro čtení a zápis dat do této tabulky. Objekt Table Gateway může také poskytnout metody pro vytváření, mazání a aktualizaci záznamů v této tabulce.

* + Sql injection sanatizer, re.sub(r'[;\'\"]', '', input\_str)

1. Otázka
   * Dedikovaný, nededikovaný (i pracovní) server, server klient (komunikuje za pomocí UDP nebo TCP), dělí se na obsáhlý (thick clinet) … vše provádí sám synchronizuje se serverem, tenký (thin client) … využívá prostředky serveru je to jen view, hybridní … např: načítání a ukládání dat nad daty dělá operace user bez potřeby serveru
   * V pythonu třída socket
   * Nezachovává pořadí packetů, není spolehlivý, sám protokol si nevyžádá opětovné poslání packetů, UDP
     1. Obsah obrázku text, snímek obrazovky, Písmo

        Popis byl vytvořen automaticky
     2. Obsah obrázku text, snímek obrazovky, Písmo

        Popis byl vytvořen automaticky
   * Spojovitá služba, nejdříve se naváže spojení, three way handshake (SYN,SYN+ACK,ACK), TCP, využívá http, SMTP, SSH
     1. Obsah obrázku text, snímek obrazovky, Písmo

        Popis byl vytvořen automaticky
     2. Obsah obrázku text, snímek obrazovky, Písmo

        Popis byl vytvořen automaticky
2. Otazka
   * Vodopádový model (sekvenční lineární proces) (requirements , design , development , testing , deplyment , maintaince)
     1. Máme jasně daný cíl, můžeme ušetřit část později když věnujeme dost času přípravě, klade důraz na dokumentaci
     2. Klient nemá úplnou představu, nemůžeme skákst mezi fázemi, dokud nedokončíme jednu nemůžeme jít na druhou, těžko se provádí změny
   * Prototypovací model, vytváříme prototyp aby měl zákazník představu o projektu, výsledek se může lišit (requirments, [quick desgin], prototyp development, customer evaluation feedback, start building /start from 2. with feedback)
     1. Horizontální prototyp UX/UI, vertikální prototyp funkcionalita
     2. Zahazovací prototypování prototyp se nepoužije ve výsledku jen myšlenka kterou představoval, Evoluční prototypování děláme robustní prototyp, na kterém pak stavíme a vylepšujeme ho
     3. Extrémní prototypování (především v webovek) skládá se z tří částí
        1. Jednoduchá statický návrh za pomocí html css
        2. Kódování obrazovek funkční UI se simulovanými službami
        3. Implementace potřebných služeb
     4. Inkrementální z prototypu se stane celek
     5. Snížíme čas a náklady, zvýšíme kvalitu požadavků, lepší zapojení zákazníka
     6. Výsledek != prototypu, nadměrný čas strávený na prototypu, nedostatečná analýza, zaměřujeme se na část a ne na celek, vyšší počáteční náklady
   * Agilni (inkrementální) model, může se přizpůsobit uživateli, zůstáváme s ním celou dobu v kontaktu, dělíme vývoj do sprintů, jedná se o sérii malých vodopádů
     1. (development na více menších částí), kontrolujme průběh vývoje (analýza [stanovení cílů, alternativ a velikosti iterace], vyhodnocení [probrání alternativ, identifikace a řešení rizik], vývoj [vývoj + testování], plánování [plán pro příští iteraci])
   * Extrémní programování (vše se řeší do nejmenšího detailu), agilní iterativní metoda, důležitá je komunikace, sprinty jsou mnohem kratší
     1. Programování ve dvou … víc očí víc vidí, víc hlav víc ví,1.programuje 2. kontroluje
3. Otázka
   * Jsou obecná řešení často vyskytujících se problémů v architektuře softwaru, jedná se o principy šablony, jak řešit problémy
   * Creational desing patern … popisují výběr třídy nového objektu a zajištění správného počtu těchto objektů, singleton, Object pool, Factory metoda
     1. Object pool … kontejner obsahuje určitý počet instancí nějaké třídy
        1. Minimalizuje náklady na vytváření a mazání objektů … zvyšuje výkon a snižuje paměťové nákroky, zabraňují zahlcení paměti apod
        2. Těžké odhadnout správný počet objektů, obtížné, pokud více vláken přistupuje současně
     2. Singleton … umožňuje vytvořit jenom jednu globálně sdílnou instanci třídy (db)
        1. Pouze jedna instance, snižuje paměťové nároky, poskytuje globální přístup, je vytvořen až ve chvíli, kdy je poprvé potřeba
        2. Můžeme jej použít odkudkoliv, obtížně testovatelný, nevodný ve více vláknovém prostředí (nesmí být vytvořen víckrát)
        3. Obsah obrázku text, snímek obrazovky, Písmo, software

           Popis byl vytvořen automaticky
     3. Factory metoda … když máme několik tříd které implementují stejný interface a chceme nechat program rozhodnout která z nich by se měla použít
        1. Jednoduchá výměna implementace bez změn v kódu klienta, zlepšení modularity
        2. Potřebujeme sub třídu pro každou implementaci (hodně tříd), těžké zjistit kterou sub třídu použít pro vytvoření objektu
        3. Obsah obrázku text, snímek obrazovky, Písmo

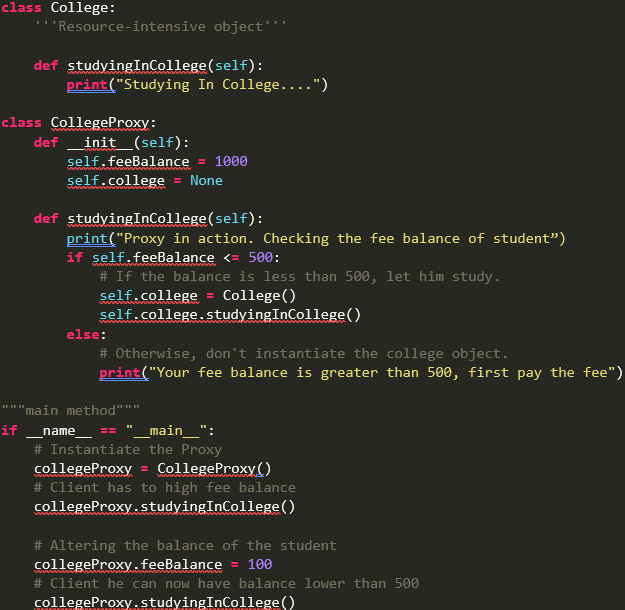
           Popis byl vytvořen automaticky
   * Structural design pattern … popisuje možnost uspořádání tříd nebo komponent v systému, fasáda, proxy
     1. Fasáda
        1. poskytuje jednoduché a srozumitelné rozhraní s komplexním systémem, izoluje klienta od implementace
        2. Omezuje pokročilé uživatele, které potřebují přístup ke komponentám, může být složité přidání dalších funkcí komponent a úprava fasády
        3. Obsah obrázku text, snímek obrazovky, Písmo, design

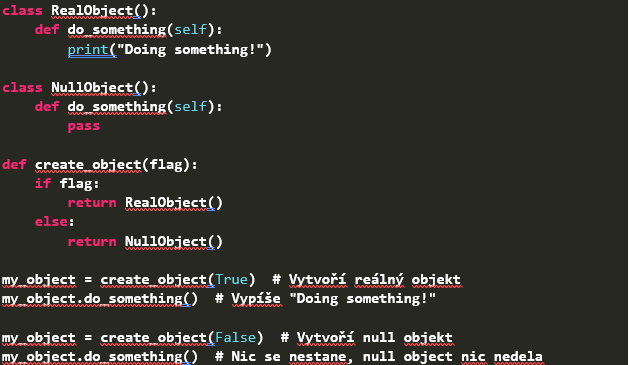
           Popis byl vytvořen automaticky
     2. Proxy … vytváříme prostředníka mezi klientem a skutečným objektem, dovoluje nám řídit přístup k objektu přes zástupce
        1. Remote Proxy – přistupujeme k vzdálenému objektu na serveru

Virtual Proxy – přistupujeme k velkému objektu, ale používá jenom jeho část, Proxy inicializuje jen ty části co se používají

Protection Proxy – kontrola přístupu k objektu klientská oprávněníl

Reverse Proxy – load balancer, rozděluje zátěž na jednotlivé servery obsah severu je identický, snaží se to rozdělovat spravedlivě

* + - 1. Snadné dodání nových funkcionalit bez nutnosti měnit kod klienta nebo skutečného objektu, šetří paměť (Virtual), ochraňuje (Protection)
      2. Může zpomalit systém snížit výkon, zpomaluje přístup k objektu (objekt někdy vyžaduje okamžitou odpověď)
      3. 
  + Behavior design paterns … popisují interakce mezi objekty a třídami v systému, chování, iterátor, null object
    1. Iterátor … umožňuje nám procházet kolekce bez ohledu na způsob, jakým jsou uloženy (implementovány)
       1. Umožňuje procházení objektů bez znalosti vnitřního mechanismu kolekce, používá se v kombinaci s dalšími návrhovými vzory (Fasáda)
       2. Může být pomalý pokud se používá s velkými datovými strukturami, nelze použít na datové struktury jako jsou stromy nebo grafy
       3. Obsah obrázku text, snímek obrazovky, Písmo

          Popis byl vytvořen automaticky
    2. Null Objekt … omžňuje nahradit null hodnotu objektům, implementující stejné rozhraní, to bez složitého kontrolování null
       1. Minimalizuje problémy s chybějícími hodnotami zlepšuje stabilitu aplikace, umožňuje snadnější testování
       2. Může být náročnější na implementaci u složitějších objektů, může zvýšit nároky na paměť, když se budou vytvářet často
       3. 

1. Otazka
   * Snaha kopírovat reálný svět, jeden z programovacích paradigmatu, zakládá se na teorii objektů … základní pojem je třída, podpojem je objekt využívá předpis třídy (pamatuje si stav … atributy, poskytuje metody pro práci s ním)
   * Dědičnost … předáváme atributy a metody zjednodušení psaní (nechceme vytvářet instanci … abstrct, přepisujeme metody override)
   * Metody … volám „tečková konvence“, method overrideing, overloading (máme furt přístup k originální metodě „base“)
   * Rozhraní … píšeme hlavičky metod „I“, interface místo class, když chceme aby některá skupina sdílela metody
   * Polymorfismus … dovoluje volat metodu s jiným výsledkem podle toho jaký objekt, ukládat objekty do struktur které mají stejného rodiče, overriding, overloading
   * Zapouzdření … na venek se tváří jako jeden objekt, modifikátory přístupu podle asmebly:
     1. Public (all), private(same class), protected (same class, derived all), private protected (same class, derived same), internal(all same), protected internal (all same, derived different)
     2. Static nemůže být založena je globálni ma jednu verzi, nejde new
   * Agregace (shared aggregation) kompozice (composite agregation) … vztahy mezi třídami, obě jsou to agregace, kompozice může být část součástí maximálně jednoho celku. U agregace to neplatí, zde může část bez problémů být obsažena ve více celcích
     1. Agregace prázdný kosočtverec (učitel učí na více školách, předmět může být vyučován ve více oborech), tvoří volný celek jeden může existovat bez druhého
     2. Kompozice černý kosočtverec (motor může mít jenom jedno auto, okres je v jednom kraji), tvoří těsný celek jeden nemůže existovat bez druhého
2. Otazka
   * Prostředek pro zápis algoritmů, které mohou být prováděny na potírači. Posloupnost instrukci, jádro řídí chod počítače (základní program), dělíme na vyšší a niží abstrakce
   * Vyšší … bližší člověku, obsahuje konstrukce, které počítač nezná, méně kódu pro napsání složitých algoritmů, často se kompilují do mezikódu a následně interpretují, přenositelný mezi zařízeními, klade větší nárok na paměť pomalejší
   * Nižší … bližší počítači, nepřenositelné … často se kompilují, rychlé nemají velké nároky na paměť, používají se na psaní ovladačů, sytému a firmwaru, Asembler, C
   * Kód se zpracuje do hexadecimálního kódu (strojový) následně do binarky a pak do procesoru které vykonává dané instrukce
   * Asembler překladač, který se využívá k překladu programu napsaný v  [jazyce symbolických adres](https://cs.wikipedia.org/wiki/Jazyk_symbolick%C3%BDch_adres)(JSA) do strojového kódu, JSA už je velmi blízký strojovému kódu jen se využívají člověka srozumitelné názvy pro některé instrukce jako MOV (přesun dat)
   * Kompilovaný … C, C++ … kompilace dlouhá pak rychlé, závisí na místě kde zkompilováno, odhalí chyby při kompilaci
   * Interpretovaný … Python, Ruby … potřebuje tlumočníka, je jedno kde byl napsán, pomalý, neodhalí všechny chyby jako kompilovaný, nemusíme dávat datové typy
   * Hybrid … C#, Java … jsou kompilované do mezikódu a následně interpretované, interpretace i kompilace zabírá méně casu než u předchozích ale dohromady už ne
   * Syntaxe … python nevyužívá závorky ale mezery a podobně
   * Imperativní (procedurální) programování … OOP (metody, třídy, apod.) a Strukturované (pomocí funkcí)
   * Deklarativní (neprocedurální) ... logické (matematické logika)
3. Otázka
   * Musíme si dat pozor na výjimky, ukládáme data na delší dobu, používá se open
     1. Obsah obrázku text, snímek obrazovky, Písmo, řada/pruh

        Popis byl vytvořen automaticky
     2. r(read), w(write), a(append), b(Binary mode), t(text mode), +(open for r/w)
   * de(opačný proces, rekonstrukce)/serializace(proces transformování objektu na stream bitů) … ukládáni objektu do paměti, v pythonu se používá pickle
     1. Obsah obrázku text, Písmo, snímek obrazovky, řada/pruh

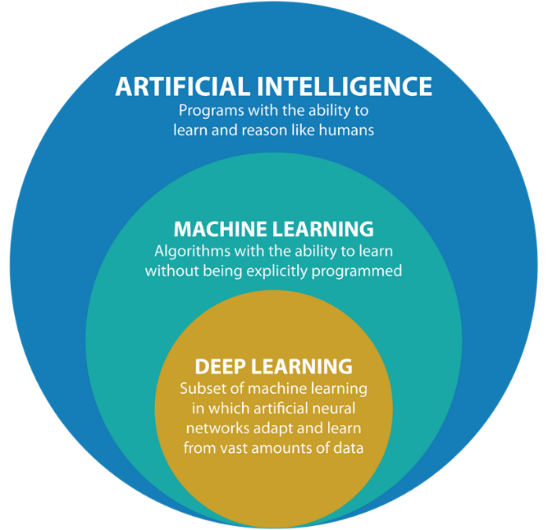
        Popis byl vytvořen automaticky
     2. x může být pole nebo instance třídy, import pickle
   * ukládání načítáni za pomocí formátů jako je csv, json
     1. Obsah obrázku text, snímek obrazovky, Písmo, číslo

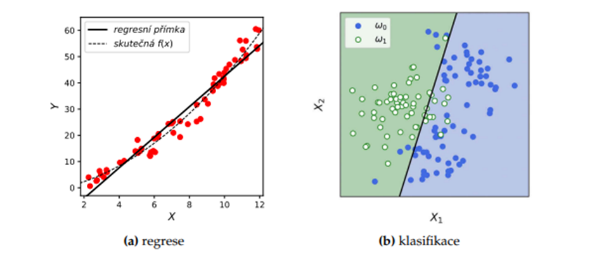
        Popis byl vytvořen automaticky
     2. Obsah obrázku text, snímek obrazovky, Písmo, číslo

        Popis byl vytvořen automaticky
     3. Obsah obrázku text, snímek obrazovky, Písmo

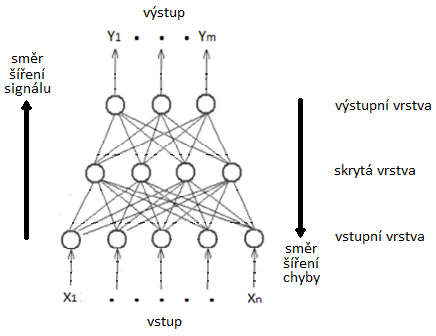
        Popis byl vytvořen automaticky

Shared (17, 18, 19):

* Machine learning jedná se o algoritmy pro učení které přímo neprogramujeme. On nalezne patern z dat namísto toho aby mu byla přímo tento patern poskytnut.
* Vytváří model s ohledem na sample datech (testovacích datech) za účelem predikování nebo rozhodovaní bez přímého zásahu programátora.
* 
* Učení s učitelem(supervised) … regrese klasifikace, Učení bez učitele(unsupervised) … shlukování, Kombinace s Učitelem a Bez učitele(semi-supervised), zpětnovazební učení(reinforcment) … učení posilováním, agent chce dostat odměny
* Klasifikace rozděluje na několik tříd, regrese odhaduje reálné číslo, shlukování zařazuje objekty do skupin
* Nelze nalézt dokonalý model snažíme se jen co nejvíce potlačit chybu, hledáme funkce s nejmenší chybou

1. Otazka
   * Zbavit se extrému a nesmyslů (zúžit rozptyl dat), zbavit se null záznamů, všechny číselné hodnoty by měli být mezi 0 a 1 (některé model neuronové sítě pak poskytují lepší výsledky), dostat parametry do standardizovaného stavu, zbavit se irelevantních parametrů, nejlépe mít všechny data vyjádřena za pomocí čísel, zbavit se duplikátů
   * Bias … je stav ke kterému se můžeme dostat, když budeme mít model s zavádějícími informacemi, model se odchyluje od reality kvůli nějakým extrémním případům,
   * Kauzalita jeden jev přímo ovlivňuje jiný, korelace dva jevy se vyskytuji nezávisle na sobě, př léto … zmrzlina, kriminalita, korelace neimplikuje kauzalitu, ale kvůli kauzalitě je potřebná korelace
2. Otazka
   * 
   * Lineární regrese … snažíme se proložit přímku mezi všemi body abychom dospěli k nejmenší chybovosti (sqare error, absolute error … průměr všech chyb) výsledek je reálné číslo
   * Klasifikace snažíme se proložit do grafu přímku tak by nám nejlépe rozdělila dvě množiny prvků, následně zařazujeme do jednotlivých kategorií výsledek pak řadíme na (true positive, false positive, true negative, false negative)
   * Obsah obrázku text, snímek obrazovky, diagram, Písmo

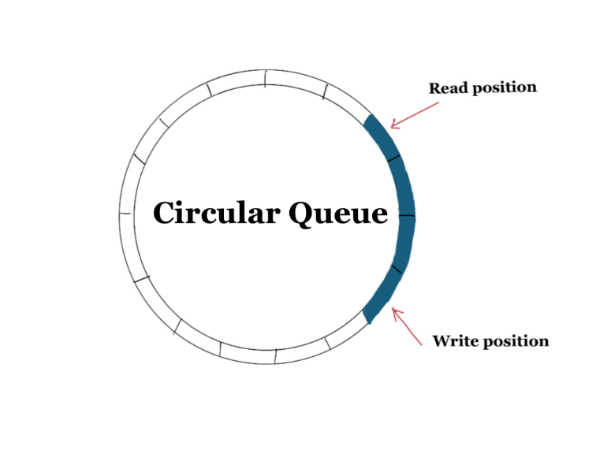
     Popis byl vytvořen automaticky
   * Rozhodovací strom(binární strom) je robustnější nemusí zachytávat tu střední hodnotu, v uzlech může být libovolná podmínka, predikce v koncových uzlech
   * Klastrová analýza shlukování do skupin objektů clusterů podle toho jak jsi jsou objekty podobné víc než v jiných clusterech
3. Otazka
   * Dokážeme vygenerovat složitější konstrukce, řešíme složitější úlohy, základním stavebním kamenem je neuron obsahuje jeden výstup a libovolné množství vstupů, každý vstup má váhu která se dá nastavit, každý výstup je zpracován funkcí a poskytnut dále, jejím vzorem je chování odpovídajících biologických struktur, u hodně případů se může jednat ooverkill
   * Perceptron neuronová síť s jedním neuronem … aktivační funkce logická funkce = logická regrese, linearní … lineární regrese, za běhu se bude algoritmus snažit upravovat váhy tak aby výsledek byl co nejpodobnější trénovacímu příkladu
     1. Obsah obrázku diagram

        Popis byl vytvořen automaticky
   * Neuronové sítě dělíme na vrstvy v každé vrstvě je n neuronů
   * Nejprve provedeme krok v před s sempl daty s váhami které byly vygenerovány náhodně, výstupní hodnotu porovnáme a spočítáme jak bychom měli upravit váhy abychom chybu na výstupu zmenšili. Nejprve upravím v první vrstvě od zadu tím vznikne nová chyba a podle toho upravíme další. A takhle až na začátek. Tedy propagujeme chybu která nám vznikla na vstupu
     1. 
   * Deep learning používá s velkým počtem neuronů v vrstvách a s mnoho vrstvami. Potřebujeme velký výkon. Používá jiné aktivační chyby. Nabylo to v minulosti atraktivní, protože nebyl dostatečný výkon na ně.
4. Otázka
   * Obecně
     1. Testujeme: **funkčnost** (vše se chová jak má), **Použitelnost** (lze dosáhnout požadovaného cíle), **spolehlivost** (chová se vždy stejně i po přetížení), **výkon**(je systém adekvátně rychlí), **podpora**(funguje systém na všech zařízení, dobře se instaluje konfiguruje), **bezpečnost** (funguje zabezpečení před neoprávněnými),
     2. Fáze: **Unit testy** (nejmenší možnou jednotku kódu), **integrační testy** (testuje interakce mezi moduly), **Systémové testy** (testují celý systém jako celek), **akceptační testy** (splňuji požadavky uživatele, přetíženi apod, do produkce)
     3. Znalost kódu: **white box** (tester zná interní implementaci, testuje implementaci), **black box** (tester nezná interní implementaci, testuje design)
     4. Způsob: **Manuální** (tester postupuje podle návodu a zapisuje výsledek), **Automatizované** (zavolám funkci a očekáváme určitý výsledek**), Exloratory testing** (tester prozkoumává aplikaci a zkouší funkce jestli něco nenajde)
   * V pythonu vytváříme novou třídu pro unitestování, od děděnou od unittest.TestCase, každá metoda musí začínat test\_ aby byla následně za pomocí unittest.main() spuštěna
     1. Obsah obrázku text, snímek obrazovky, Písmo

        Popis byl vytvořen automaticky
     2. Nad hlavičkou funkce můžeme použít dekorátory: @unittest.skip(„reason“), @unittest.skipIf(condition, „reason“), @unittest.skipUnless(condition, „reason“), @unittest.expectedFailure
   * Dokumentace:
     1. vnější (mimo zdrojvý kód) … požadavky, návrh architektury. Popis rozhraní, tutoriál, FAQ. Uživatelská dokumentace (readme), popis návrhu, postupy
     2. vnitřní (součástí zdrojového kódu) … styl zápisu zdrojového kódu, programátorská dokumentace a komentáře k souborům, třídám, funkcím, bloků kódu, popis rozhraní
     3. readme … název projektu, popis projektu, autor, instalace, nastavení (konfigurace), správa a práce s aplikace, licence, stav projektu, kontakt, ukázky
     4. Obsah obrázku text, snímek obrazovky, Písmo, číslo

        Popis byl vytvořen automaticky
5. Otazka
   * Spojovy seznam … obsahuje hodnotu, odkaz na další prvek, odkaz na předchozí (pokud je obousmerny), jednotlivé složky nemusí ležet za sebou, první prvek = head, poslední = tail, jedná se o rekurzivní datovou strukturu protože každý prvek odkazuje na položku stejného typu
     1. Jednosměrný, každý prvek má svou hodnotu a odkaz na další prvek
     2. Cyklický, každý prvek má svou hodnotu a odkaz na další prvek, poslední ukazuje na první prvek
     3. Obousměrný, každý má vlastní hodnotu a ukazuje na další a předchozí prvek
     4. Cyklický obousměrný, předchůdcem prvního je poslední následovníkem posledního je první jinak jako obousměrný
   * Strom … jedná se o graf bez smyček, mí n-1 hran kde n je počet vrcholů, první vrchol se nazývá kořen, vrchol který nemá žádné potomky list, hierarchické uspořádání, rekurzivní datová struktura
     1. Vlastnosti: (kolik smí mít každý vrchol max potomků … binární strom 0,1,2), hloubka, pravidelnost (každý vrchol má 0 nebo N potomků), Vyváženost (všechny listy jsou ve stejné hloubce), Úplná pravidelnost (pravidelnost + vyváženost).
     2. Průchod stromem: Preorder (kořen,levy,pravy) (1 -> 2 -> 4 -> 8 -> 9 -> 5 -> 3 -> 6 -> 7), Inorder (levy, kořen,pravy) (8 -> 4 -> 9 -> 2 -> 5 -> 1 -> 6 -> 3 -> 7), Postorder (levy, pravy, kořen) (8 -> 9 -> 4 -> 5 -> 2 -> 6 -> 7 -> 3 -> 1)
     3. Obsah obrázku snímek obrazovky, kruh, žlutá, klipart

        Popis byl vytvořen automaticky
   * Fronta … FIFO, (prioritní fronta elementy v frontě se navíc řadí podle priority)
     1. Vyřízení požadavků, operátor pipe,
   * Zásobník … LIFO, ukládání stavu v algoritmu, prohledávání do hloubky a ve všech rekurzivních algoritmech, používá se v paměti, kde jsou ukládány hodnotové datové typy
   * Halda … binární halda je úplný binární strom, ve kterém platí, že každý potomek vrcholku má nižší nebo stejnou hodnotu nežli vrcholek sám, takže kořen je vždy maximum. Této haldě se říká Max-Heap. Opak je Min-Heap, nový prvek se vkládá na konec haldy pokud to bude nutné bude halda přeuspořádána
     1. Obsah obrázku skica, diagram, kresba, vzor

        Popis byl vytvořen automaticky
   * Kruhový buffer … funguje na principu FIFO, jedná se o pole kterému se slepil konec a začátek, pokud je head a tail na stejném místě není nic v poli, buď se po naplnění překopíruje do nového a většího nebo se do nekonečna přepisuje
     1. 
6. Otazka
   * Každá aplikace běží minimálně na jednom vlákně, počet vlákána = počet jader (může být víc ale už se začnou střídat), čím víc vláken tím větší nároky na procesor a nevodné přepínání vláken, jo důležité uzamykat vláknové nebezpečné operace protože se můžou přerušovat,
     1. Obsah obrázku text, snímek obrazovky, software

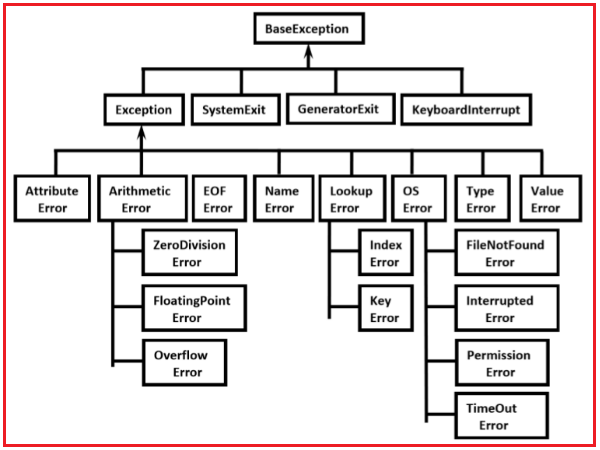
        Popis byl vytvořen automaticky
   * Řeší problémy spolupráce procesů na jedné úloze, jakmile více zájemců o jeden zdroj dochází k konfliktu, kritická sekce … část kódu kde může dojít ke konfliktu, pracuje se sdílenou datovou strukturou
     1. Zámek … mechanismus pro řízení přístupu ke kritické sekci
        1. Obsah obrázku text, Písmo, snímek obrazovky

           Popis byl vytvořen automaticky
     2. Mutex … do krytycké sekce může vždy jen jedno vlákno
        1. `Obsah obrázku text, snímek obrazovky, Písmo, design

           Popis byl vytvořen automaticky
   * Pouřívá me zde dva operátory async a await, async … jedná se o asynchroní metodu obsahuje volání asynchroní metody za pomocí await, zpracovává se v hlavním vlákně, await … vrátí kontrolu ve chvíli kdy je úloha s operátorem await dokončena,
     1. Obsah obrázku text, snímek obrazovky, software, Písmo

        Popis byl vytvořen automaticky
   * Concurent design patterns … jsou návrhové vzory které se zabývají více vláknovými programy
     1. Event-based asynchronous … Na událostech založený asynchronní návrhový vzor řešící problémy s Asynchronním vzorem, které nastávají ve vícevláknových programech
     2. Balking … je softwarovým vzorem, který na objektu vykoná nějakou akci, pouze pokud je objekt v určitém stavu
     3. Thread pool … V bazénku vláken je vytvořen nějaký počet vláken pro řešení nějakého množství úloh, které jsou organizovány ve frontě. Zpravidla je výrazně více úloh než vláken.
     4. Scheduler … Jde o souběžný vzor, který se používá pro explicitní kontrolu, kdy mohou vlákna vyvolávat jednovláknový kód.
     5. Read write lock … Tento vzor, také známý jako RWL, je vzor, který umožňuje souběžný přístup k objektu pro čtení, ale vyžaduje exkluzivní přístup pro zápis.
     6. Monitor object … je přístup k synchronizaci dvou nebo více počítačových úloh, které používají sdílené zdroje, zpravidla hardwarové zařízení nebo sadu proměnných.
7. Otazka
   * Seřazenost: SortedSet, SortedList, SortedDictionary (rychlé vyhledávání, dlouhé vkládání)
   * Bez opakovani: HashSet, Sorted Set (při pokusu vložit již existující prvek se prvek nevloží)

* Klíčové (bez opakování): Dictionary, SortedDictionary, ConcurrentDictionary, ListDictionary (small amount), SortedList, HashTable (same throw error)
* Indexované: List, String, Array
* Hashované: HashTable, Hashset, (Vypočtení hashe, Porovnání hashe, Určení indexu … do hashove tabulky, vloženi prvku do sady), rychlé najít prvek, spočítám hash a podívam se do hashove tabulky, ze stjného důvodu je rychlejší i remove, ovšem přidání prvku je pomal

1. Otazka
   * Raise vyvolá výjimku, zakázané stavy a chyby, pro odchytávání try/except/else/finally, jsou hierarchické, dobré logovat chyby
   * Aserce se používají pro kontrolu podmínek a debugování vyvolává AssertionError, aserce ve výsledném programu nesetrvává na rozdíl od raise, … assert conditon, „error“
   * Debugování lze přímo v IDE například pycharm kde pomocí breakpointu označíme místo kde začneme debutovat a následně můžeme jít řádek po řádku, můžem použít knihovnu pbd (pbd.set\_trace()), příkazů, step, (next), p n … n je proměná která nás zajímá, list, help, print(content)
   * Zpracování chyb je proces kterým se zabíváme když v programu nastane výjimka, používáme try/except/else/finally, můžeme odchytávat přesné výjimky podle jejich názvu
   * 
2. Otazka
   * Můžeme s stringem pracovat jako polem pomocí indexu, split, spojeni, porovnávání, replace, feild join do stringu, lower, upper, strip, is..something, string je pole charů, len, je to referenční datový tip, string je neměnný
   * Pro regex se používá re třída a u textu se dává r““, match, split, sub, search, findall, fullmatch
   * pro encode využíváme v pythonu encode a pro dekódování decode, standartně se v pythonu používá utf-8 kódovací sada, pokud cheme dosta into do bylu tak int.to\_bytes(), chr … convert unicode number to character, ord … convert chracter to unicode number