**Pametna specializacija in analiza zahtev**

Na prvem predavanju pri predmetu Uvod v računalništvo smo poslušali o pametni socializaciji in analizi zahtev ter o rabi pametne specializacije v vsakdanjem življenju (pametne aplikacije, računalništvo v megli, orkestracija komponent).

Pametno specializacijo najdemo v devetih različnih strokah:

**Zdravje**

·         aplikacija Smart Healthcare – aplikacija za informiranje o zdravstvenih zadevah

·         **moje mnenje:** zdravje je vsakemu človeku med najpomembnejšimi vrednotami življenja, zato je vsakršna tehnologija, ki bi nam omogočila hitrejšo pomoč več kot dobrodošla, problem nastane, če tehnologija zataji

**Gospodarstvo**

·         podjetje Betterment – aplikacija za pomoč pri finančnih odločitvah

·         **moje mnenje:**razvoj gospodarstva je pomemben, brez razvoja gospodarstva se družba ustavi oz. celo nazaduje, pride do izgub finančnih sredstev ali celo do kriz

**Mobilnost**

·         podjetje Tesla –samovozeči avtomobili

·         **moje mnenje:** samovozeči avtomobili so že dolgo stvar prihodnosti, vendar ta prihodnost se že spreminja v sedanjost, pravilno sprogramiran računalnik se bo v prometu obnašal bolje od človeškega voznika, vendar pa so ljudje nezaupljivi

**Javne storitve**

·         v Franciji se v promet uvajajo samovozeči avtobusi

·         **moje mnenje:**kot za vozeče avtomobile, so stvar prihodnosti, ki bo vsak čas del vsakdana, ljudje so tudi do tega nezaupljivi

**Energija in trajnost**

·         podjetji Google in Nnergix – optimizacija porabe energije

·         **moje mnenje:**vsak bi hotel prihraniti denar na račun neporabljene energije, pride lahko do kake napake v sistemu, ki lahko privede do nelagodja

**Novice, mediji in zabava**

·         podjetji Apple in Amazon – virtualna asistenta Siri in Alexa

·         **moje mnenje:**sam ju nisem nikoli uporabljal, je pa res da znajo biti te asistenti zelo v pomoč, žal pa ljudi še bolj »polenijo«, še ena negativna stvar je prisluškovanje

**Industrija**

·         podjetja Toyota, BC Machining in Chevron – optimizacija dela

·         **moje mnenje:**reševanje problemov preden sploh do njih pride je ena izmed najboljših stvari, ki jih lahko dosežeš kjerkoli

**Turizem**

·         podjetja Booking.com, AltexSoft in Winding Tree – lažje in hitrejše rezervacije

·         **moje mnenje:**uporabniku poenostavijo iskanje in rezervacijo prenočišč, letov, … seveda lahko pride do kake napake v sistemu

**Izobraževanje in izmenjava znanja**

·         podjetji Content Technologies in Microfsoft – lažje in hitrejše učenje

·         **moje mnenje:**sam uporabljam MS Office 365 zelo pogosto, tudi za to nalogo sem ga, skupna raba nudi medsebojno pomoč in povezovanje z drugimi

**Pametne aplikacije**

Učili smo se tudi o pametnih aplikacijah, njihovem okolju in rabi. Pametne aplikacije nam olajšajo delo in v večini primerov zmanjšajo število napak oz. zmanjšajo toleranco, kar človeku ne uspe. Umetna inteligenca je meni osebno znana stvar, ampak iskreno nimam pojma, kako bi recimo sprogramiral kamero, da ugotovi identiteto človeka ali pa če le-ta nosi čelado, me pa to vsekakor zanima in bi se rad to naučil. Govorili smo tudi o tem, kako enostavne oz. zahtevne so določene zahteve, ki jih hočemo sprogramirati. Za primer smo si vzeli gradbišče in s pomočjo umetne inteligence preverjali koliko imamo materiala, ali delavci nosijo čelade, si pogledali senzorje v stavbah, ki so še v procesu načrtovanja in/ali gradnje. Učili smo se tudi o postopkih dela, ki bi se jih lahko držali pri vsaki stvari, ki se je lotimo in si s tem morebiti olajšali delo.

**Računalnik kot stroj**

**Zgodovina**

Znotraj teme računalnik kot stroj smo se pogovarjali o zgodovini in razvoju računalnika in o njegovi uporabi v samem začetku. Razvoj računalnika je bil potreben za reševanje nalog in ugank, za katere je človek potreboval veliko časa oz. mu jih je bilo težko rešiti. Najstarejše civilizacije kot so Grki, Egipčani, Babilonci, Indijci, Kitajci in Perzijci so vsak po svoje “sestavljali” današnjo matematiko. Grki so prispevali geometrijo in logiko, Egipčani in Babilonci numerične metode, Indijci desetiški sistem in koncept ničle, Kitajci in Perzijci pa algoritmično reševanje problemov. Leta 1614 je John Napier uvedel logaritme. Z letom 1623 se je začel razvoj prvih računalnikov, ki so lahko reševali le lažje matematične operacije kot so seštevanje, odštevanje, množenje in deljenje, kar pa je bilo za tisti čas nekaj neverjetnega. Ti stroji so spadali v družino mehanskih računal. V začetku 19. stoletja so ustvarili prvi programabilni računalnik. Leta 1833 so mehanska računala “izpodrinili” mehanski računalniki. Ta je poleg osnovnih operacij znal delati tudi z polinomi in začelo se je tudi programiranje, shranjevanje in procesiranje podatkov. V letih med 1940 in 1950 so se uveljavile elektronske naprave in prišlo je do “rojstva računalnikov”. Računalniki so bili še vedno mehanski, uveljavil pa se je tudi dvojiški sistem. Druga svetovna vojna je prizadejala veliko hudega, je pa bila tudi poziv k temu, da se izboljšajo računalniki, saj bi lahko z njimi prestregli takrat šifrirana sporočila sovražnikov, kar se je leta 1943 tudi zgodilo. Leta 1945 je Von Neumann postavil arhitekturo računalnika, ki velja že več kot 60 let in po letu 1950 so se računalniki iz leta v leto čedalje bolj razvijali. John Moore je uveljavil tako imenovani Moorov zakon, ki pravi, da se bo na vsakih 18 - 24 mesecev število elementov na vezjih podvojilo, cena pa bo ostala enaka.

**Model računalnika**

Von Neumannova arhitektura torej še danes kroji sleherni računalnik. Kako sploh izgleda in kako deluje? Von Neumann je za model računalnika uporabil 3 sestavne dele. CPE oz. centralna procesna enota oz. enostavneje procesor, pomnilnik in vhodno-izhodni sistem. Procesor je središče računalnika in on je tisti, ki skrbi za delovanje računalnika in izvajanje procesov. V CPE najdemo krmilno enoto, ALE oz. aritmetično logično enoto in registre. Namen pomnilnika je shranjevanje podatkov in procesov, ki se izvajajo. Vhodno-izhodni sistem pa se deli na vhodne in izhodne naprave. Vhodne naprave so tiste s katerimi uporabnik vnaša podatke v računalnik (miška, tipkovnica, kamera, mikrofon, optični bralnik, …), izhodne pa tiste, ki podatke iz računalnika prinesejo do uporabnika (zaslon, zvočnik, tiskalnik, …)

**Zapis podatkov in logični konstrukti**

Na 3. predavanju smo se pogovarjali o zapisovanju podatkov v računalnik oz. komunikaciji med uporabnikom in računalnikom. Govorili smo tudi o 2 vrstah signalov; analogni in digitalni.

Računalniki ne razumejo slovenskega, angleškega ali pa kateregakoli drugega »človeškega« jezika, zato mu moramo podatke in informacije vnašati v njemu razumljivem jeziku. Tu pride v poštev dvojiški ali binarni številski sistem. Ljudje poznamo desetiški sistem, ki ga uporabljamo za štetje in ima znake 0, 1, 2, 3 ,4, 5, 6, 7, 8 in 9. Dvojiški sistem pa ima samo 2 znaka, to sta 0 in 1. Računalnik to razume kot DA in NE, 1 pomeni da, 0 pomeni ne. Torej kako računalnik ve katere tipke jaz trenutno klikam, ko pišem tole poročilo. Vsaka črka ima svojo unikatno kodo oz. šifro, ki je zapisana v dvojiškem sistemu. Šifre za posamezne črke lahko najdemo v ASCII tabeli, to je standardizirana tabela, kjer je vsaka črka, znak, simbol, številka, zapisana s svojo šifro. Prav tako lahko z dvojiškimi števili štejemo, saj je proces štetja enak, vendar ljudje smo desetiški sistem zbrali, ker imamo 10 prstov in desetiški sistem je vsekakor enostavnejši in preglednejši od dvojiškega, za računalnik pa velja ravno nasprotno. Števila lahko pretvarjamo med desetiškim in dvojiškim sistemom. Za našo stroko je pomemben tudi heksadecimalni oz. šestnajstiški sistem, ki ima znake od 0 do 9, potem pa še črke A, B, C, D, E in F, ki zamenjajo števila od 10 do vključno 15.

Primer pretvarjanja iz desetiškega v dvojiški sistem (hitrejši način):

·         Poznati moramo potence števila 2 (1, 2, 4, 8, 16, 32, 64, 128, …)

·         Število npr. 45 pretvorimo v dvojiški sistem tako da preprosto pod potencami pišemo 1 in 0 dokler vsota ne nanese 45

·         128 in 64 sta večji od 45, torej ju lahko ignoriramo, pri 32 bo 1, če 32 prištejemo 16 dobimo 48, kar je ponovno več kot 45, zato bo tu 0, potem ko 32 prištejemo  8 dobimo 40 in nato rabimo še 5 to pa dobimo iz 4 in 1

·         Število 45 dvojiško izgleda takole: 00101101 oz. 101101

·         Pretvarjanje v drugo smer poteka tako da samo seštejemo potence nad 1, logično

Govorili smo tudi o negativnih in decimalnih dvojiških številih, sam to recimo da obvladam, saj smo se o tem učili že v srednji šoli. Za negativna števila uporabljam dvojiški komplement, za decimalke pa gledamo ponovne potence 2, le da imamo tokrat ulomke ½, ¼, ¹⁄₈ , ¹⁄₁₆, …

Pogovarjali smo se tudi o logičnih tabelah IN, ALI, XALI, NE ter o vezjih, ki jih gradijo na podlagi teh tabel. Na srečo sem tudi o tej snovi že seznanjen iz srednje šole.

**Računalniški modeli in Turingov stroj**

Na 4. predavanju pri predmetu UR smo govorili o Turingovem stroju in na splošno o programiranju strojev. Človeški jezik je stroju nerazumljiv, zato lahko določene probleme rešuje le, če mu je podan v strojnem jeziku. Obstajajo tudi problemi, za katere ne obstaja algoritmične rešitve. Govorili smo tudi o modelu računskega agenta, s katerim ohranimo le najbolj pomembne lastnosti oz. operacije. V modelu zajamemo pomembne lastnosti pravega agenta, model je podan v različnem merilu in nima vseh funkcionalnosti. Z modeli lahko tudi napovedujemo glede na obnašanje modela, s čimer je delo z njim lažje, cenejše in varneje, omogoča pa nam tudi testiranje brez agenta.

Model računskega agenta mora brati vhodne podatke, shranjevati in brati podatke iz pomnilnika, izvajati ukaze v odvisnosti od vhoda in/ali stanja, na koncu pa mora proizvesti rezultat. Poznamo tudi matematične modele računalnikov.

Turingov stroj je sestavljen iz neskončnega traku. Na traku so polja, v katerih se hranijo simboli (1 polje, 1 simbol), na trak se zapisuje tudi abeceda, vhodni podatki in trak služi kot pomnilnik. Turingov stroj programiramo z ukazi kot so »zapiši nov simbol«, »spremeni stanje«, »premakni se za eno polje naprej/nazaj«, …

Turingov stroj je vedno najprej postavljen na 1 in branje začne na najbolj levem nepraznem polju. Vsa pravila morajo biti različna ali pa imeti različen vhod/izhod. Turingov stroj torej bere vhodne podatke, ima trak, ki služi kot pomnilnik za branje in pisanje podatkov, izvaja različne akcije in proizvede rezultat. Lastnosti Turingovega stroja ga kvalificirajo kot dober model algoritma. Turingov stroj deluje na različne načine, npr. invertiranje bitov, paritetni bit, eniški inkrement, eniško seštevanje, … Omenili smo tudi Church-Turingovo tezo. Ta teza ni dokazana in se je tudi ne more dokazati, je pa vseeno skoraj gotovo resnična, ker ni bil nikoli najden nasproten primer oz. dokaz. Teza temelji na povezovanju stroja, kodiranja in algoritma, s katerim bere in piše na vhod ali izhod. Našteli smo tudi nekaj nerešljivih problemov in kako jih prepoznamo.

**Jeziki in prevajalniki**

Na 5. predavanju smo govorili o jezikih in prevajalnikih v računalniku. Med drugim smo omenili hierarhijo po Chomskemu, to je le ena od mnogih računalniških gramatik. Računalniška gramatika temelji na znakih, ki jih vsakodnevno srečujemo – črkah. Velike črke so neterminali, male črke so terminali, velika črka S pa je začetni simbol. Govorili smo tudi o Backus-Naurovi obliki gramatike.

**Programski jeziki**

Jezik v zbirniku temelji na ukazih, ki se preslikajo neposredno v strojne ukaze, kode ukazov in naslovi niso binarni, ampak simbolični, delovanje pa temelji na psevdo ukazih, ki so uporabniku prijaznejši. Poznamo nizko nivojske in visoko nivojske programske jezike. Nizko nivojska sta strojni in zbirni jezik. Visoko nivojski pa Java, C, Python, … Poznamo tudi jezik za opisovanje pomena kot je na primer eXtensible Markup Language, jezik semantičnega spleta na primer Web Ontology Language, skriptni jeziki za generiranje formul in simbolični jeziki.

**Prevajalniki**

V procesorju se izvajajo samo strojni jezik, zato se mora vsak program prevesti v strojni jezik. Prevajanje iz zbirnega v strojni jezik je trivialno oz. enostavno, medtem ko je prevajanje iz visoko nivojskih jezikov dosti bolj kompleksno oz. zapleteno. Prevajalniki prevajajo programe iz visoko nivojskih jezikov. Mora pa prevajalnik poskrbeti, da strojni ukazi naredijo natančno to, kar pomenijo visoko nivojski ukazi, obenem pa morajo biti učinkoviti in jedrnati, kar pomeni, da mora biti koda optimizirana za čim hitrejše izvajanje. Proces prevajanje poteka v 4 korakih; *leksikalna analiza*(združevanje znakov v lekseme), *sintaksna analiza*(preverjanje sintakse in gradnja notranje predstavitve programa), *semantična analiza in generiranje kode* (analiza pomena in generiranje strojnih ukazov) in *optimizacija kode*(izboljševanje časovne in prostorske učinkovitosti kode).

**Jeziki in algoritmi**

**Jeziki**

V svetu računalništva poznamo veliko različnih programskih jezikov, ki so si bolj ali manj podobni. Vsak jezik ima svoje posebnosti, po katerih se loči od drugih in je zato bolj primeren za reševanje določenega problema, imajo pa vsi jeziki tudi določene skupne lastnosti. Izbira jezika je torej odvisna od zahtev naloge, odvisna pa je tudi od samega uporabnika, saj se lahko on v nekaterih jezikih bolje znajde oz. jih bolj obvlada, kot v drugih. Te jezike delimo na **proceduralne jezike**in na **namenske jezike**. Proceduralni jeziki se izvajajo v procedurah oz. temeljijo na nekem zaporedju ukazov, med njih spadajo FORTRAN, COBOL, C, C++, Java, C# in Python. Sam sem se v srednji šoli učil C++ in C#, zdaj pa spoznavam še Python. Vsi našteti jeziki so splošno-namenski, namenski jeziki pa so bili razviti za učinkovito reševanje točno določenih nalog. Poznamo SQL, HTML, XML, OWL in JavaScript. V srednji šoli sem uporabljal HTML in JavaScript pri izdelovanju spletnih strani in SQL pri učenju dela s podatkovnimi bazami. Poznamo še 3 alternativne programske paradigme. To so funkcijsko programiranje, logično programiranje in paralelno programiranje. Funkcijsko programiranje temelji na programu, katerega glavni gradniki so matematične in druge funkcije. Pri logičnem programiranju opisujemo dejstva in pravila, s katerimi povemo, kaj točno želimo in nas ne zanima, kako pridemo do rešitve. Programiranje torej poteka z logičnim sklepanjem, največkrat v umetni inteligenci. Paralelno programiranje pa izvajajo programski jeziki, ki omogočajo izvajanje programov na večih procesorjih. Delimo jih na SIMD in MIMD. SIMD ima eno krmilno enoto in veliko aritmetičnih logičnih enot. MIMD pa ima tudi veliko krmilnih enot in tako lahko izvaja večje število programov in operira z veliko količino podatkov. Govorili smo tudi o priljubljenosti programskih jezikov. Dandanes so v samem vrhu Java, C in Python.

**Algoritmi**

Algoritem je urejeno zaporedje nedvoumnih in izračunljivih operacij, ki proizvede rezultat in se ustavi v končnem času oz. procedura reševanja problemov v končnem številu korakov. Algoritemske operacije delimo na 3 kategorije. Zaporedne​, pogojne in iterativne operacije. Izvora besede algoritem nisem poznal, mi je pa vsekakor zanimivo, da izhaja iz imena perzijskega matematika, ki je postavil algoritme kot  osnovne matematične operacije.  Njegova knjiga je bila osnova, za uveljavitev arabskih številk v evropski matematiki, ki so še dandanes aktualne. Primeri algoritmov se delijo na enostavne in zahtevne ter seveda tiste vmes. Algoritem lahko napišemo prav za vsako stvar. Od peke palačink, do zidanje hiše in vsega bolj ali manj zapletenega. Za algoritmično reševanje problemov moramo izpolniti nekaj pogojev. Imeti moramo računskega agenta oz. izvajalca programa npr. stroj, program, osebo. Ta agent mora kreirati pravilne in učinkovite algoritme, preučiti njihove lastnosti, načrtovati programske jezike in graditi računalniške sisteme.

**Psevdokoda in načrtovanje algoritmov**

Algoritem lahko predstavimo na 3 načine. Z **naravnim** **jezikom**, s **programskim** **jezikom** in z **psevdokodo**. Naravni jezik je zelo ekspresiven, bogat, enostaven, pogosto uporabljan, redundanten, nestrukturiran, dvoumen in odvisen od konteksta. Programski jezik je strukturiran, namenjen računalnikom, ni redundanten, preveč natančen in vsebuje težko gramatiko. Psevdokoda je nekje vmes med naravnim in programskim jezikom. Je programski jezik brez podrobnosti, enostavna, berljiva, strukturirana, brez strogih pravil. Vizualizacija in prevod v programske jezike sta enostavna.

V vseh vrstah jezikov imamo tri glavne zaporedne operacije in sicer **računska**, **vhodna** in **izhodna**. Računska operacija kot že ime pove, računa določen izračun, na vhod se podatki vnašajo v algoritem, na izhodu pa vrača pridobljene podatke. Jeziki vsebujejo tudi **spremenljivke**, to so poljubno poimenovane lokacije, na katerih so shranjene izbrane vrednosti. Zaporedne operacije sestavljajo zaporedni oz. premočrtni algoritem. Algoritmi lahko vsebujejo tudi **nadzorne** **operacije**, ki so namenjene nadzorovanju poteka algoritma. Delijo se na **pogojne** **stavke** oz. **vejitve**in na **iteracije**oz. **zanke**. Pogojni stavek uporabimo, kadar želimo preveriti, če spremenljivka v algoritmu izpolnjuje določeni pogoj, npr. preveri, če je neko število sodo ali liho ali pa če je nek pogoj resničen ali neresničen (True/False). Zanke, so del programa, ki se izvajajo, če je in dokler je izpolnjen določen pogoj, primer zanke sta npr. while in do while zanki. Uporabnik se mora izogibati pisanju neskončnih zank oz. poskrbeti, da nam zanke nekaj vrnejo. V tem predavanju sem se naučil oz. ponovil snov, o tem kaj so glavne komponente programskega jezika. To so torej vhod, računske operacije, pogoji in zanke ter izhod.

**Algoritmi in kompleksnost**

Predavanja se še vedno nanašajo na algoritme. V tokratnem predavanju smo poslušali o atributih algoritmov, o analizi učinkovitosti, se naučili opisati, ilustrirati in uporabiti algoritme, govorili o redu časovne kompleksnosti in ponovno govorilo o nerešljivih problemih in o vsaj delnem reševanju le teh.

Omenili smo tudi, da za isti problem oz. nalogo obstaja več rešitev oz. več različnih poti reševanja.

**Atributi algoritmov**

Atributi oz. lastnosti so tako rekoč pridevniki, s katerimi opišemo vsakodnevne stvari, v računalništvu pa so poleg tega tudi vrednosti, s katerimi sestavljamo algoritme. Najpomembnejši atributi so pravilnost, razumljivost, eleganca in učinkovitost. Če je algoritem napisan nepravilno, nam praktično ne služi pri ničemer. Algoritem mora biti razumljiv in eleganten, saj s tem olajša vzdrževanje, nadgradnjo, preverjanje in ga lahko tudi lažje delimo z drugimi. Algoritmi so učinkoviti na 2 načina, ali so časovno hitri s čimer je delo uporabnika hitrejše, ali pa imajo dovolj prostora in s tem možnost shranjevanja večje količine informacij.

**Analiza algoritmov**

Med sabo primerjamo različne algoritme, ki pa opravljajo enako nalogo in vrnejo isti rezultat. Primerjamo jih glede na že prej naštete atribute, torej časovna in prostorska učinkovitost, razumljivost in eleganca, pravilna pa sta že, glede na to, da vrneta rezultat.

**Kompleksnost algoritmov**

Med samo smo primerjali različne redove velikosti. Za primer smo med sabo primerjali naraščanje različnih funkcij, kot so linearna, kvadratna, logaritemska, eksponenta, v računalništvu pa poznamo še druge razrede kompleksnosti kot so konstanta, kubična in polinomska. Za določanje kompleksnosti algoritma preštejemo vse operacije, ki jih vsebuje. Omenili smo tudi približne algoritme, ki pa podan problem rešijo le delno oz. rešitev ni optimizirana, vendar pa je rešitev vseeno dovolj dobra.

**Programsko inženirstvo**

V tem predavanju smo govorili predvsem o razliki med zbirnim in visoko-nivojskim programskim jezikom ter o razlikah med posameznimi programskimi jeziki. **Zbirni jezik** je bolj razumljiv in lažji za uporabo kot strojni jezik, ker se ukazi preslikajo neposredno v strojne ukaze in ker vsebuje simbolične kode ukazov, naslove in podatke ter temelji na psevdo ukazih. Ima pa seveda tudi zbirni jezik nekaj pomanjkljivosti. Pri pisanju kode v zbirnem jeziku mora programer sam poskrbeti za premikanje podatkov med pomnilnikom in registri, v zbirnem jeziku je pogled na naloge mikroskopski in je daleč od naravnega jezika. **Visoko-nivojski jezik** pa je bolj razumljiv in z njim je lažje delate kot z zbirnim jezikom. V visoko-nivojskem jeziku programerju ni treba skrbeti za pomikanje podatkov med pomnilnikom in registri, ima makroskopski pogled na naloge, programi so prenosljivi med računalniki in jezik je bolj podoben naravnemu. Na kratko smo tudi govorili o **prevajanju v visoko-nivojskih jezikih**. Za ta proces potrebujemo 4 komponente. To so **prevajalnik**, ki kodo prevede v nižje-nivojski jezik, **zbirnik**, ki kodo pretvori v objektno kodo, **programske** **knjižnice**, ki vsebujejo preverjeno kodo in **povezovalnik**, ki vse skupaj poveže v en izvršljiv modul.

Hotel sem samo omeniti, da mi je bila beseda **makroskopski** neznana. Za mikroskopski pogled sem seveda lahko sklepal, da gre za nekaj, kar lahko vidimo pod mikroskopom oz. ne moremo videti s prostim očesom. Za makroskopski pogled pa sem šel preverit na elektronski SSKJ in ugotovil, da gre za stvari, ki jih lahko vidimo s prostim očesom, kar je torej nasprotje mikroskopskega pogleda.

Našteli smo kar nekaj različnih programskih jezikov. Sam sem pred tem že iz srednje šole kar dobro poznal C++ in C#, nekaj malega smo delali tudi v Javi, slišal pa sem tudi že za Python, ki pa so ga v učni načrt v 2. letniku uvedli, ko sem bil že v 3. letniku, tako da mi je Python bil znan samo po imenu, sem ga pa seveda že kar dobro spoznal na predavanjih pri Programiranju 1. Poleg tega od prej poznam tudi HTML, CSS in osnove Jave script ter PHP, ki smo jih spoznali pri izdelovanju spletnih strani in SQL pri podatkovnih bazah. Uporabljal pa sem tudi Kotlin pri 4. predmetu tj. izdelek ali storitev z zagovorom, kjer sem izdelal enostavno aplikacijo. Na predavanju smo torej primerjali C++, Javo in Python. Samo kodiranje oz. pravila in ukazi se mi zdijo pri vseh naštetih dokaj podobni, razlikujejo se v majhnih podrobnostih, seveda pa imajo nekateri več funkcij kot drugi. Sam bi rekel, da če znaš enega, se lahko potem hitro naučiš tudi drugega, saj poznaš pravila npr. if stavkov, zank, izpisa nizov, spremenljivk, tako da se moraš naučiti samo pravila zapisa posameznega jezika, npr. v if stavku v C++ pišemo pogoj1 && pogoj2, v Pythonu pa pogoj1 and pogoj2, stvar je dobesedno enaka.