Podstawy Programowania

Michał Bujacz

bujaczm@p.lodz.pl

B9 „Lodex” 207

godziny przyjęć: środy i czwartki 10:00-11:00

http://www.eletel.p.lodz.pl/bujacz/

Podział zajęć

karta ECTS: http://www.programy.p.lodz.pl/

40 godzin laboratoriów (13 x 3h)

20 godzin wykładów (10 x 2h)

Zaliczenie wykładu:

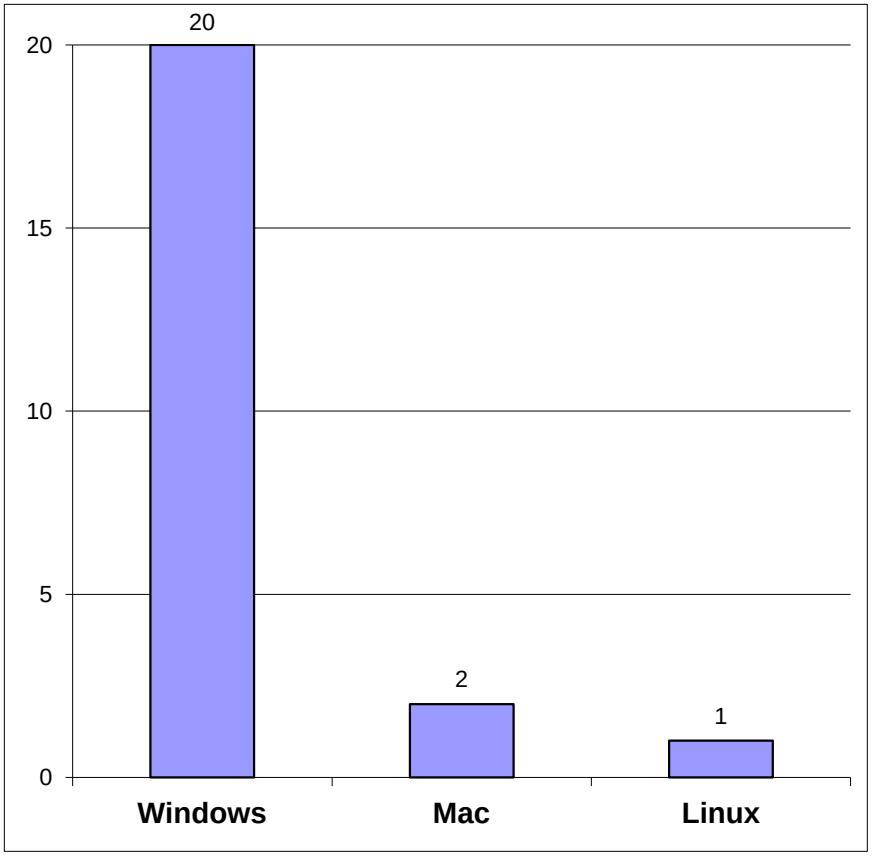
- odpytywanie na laboratoriach (pytania na koniec wykładu)
- prezentacje w parach

Podniesie oceny:

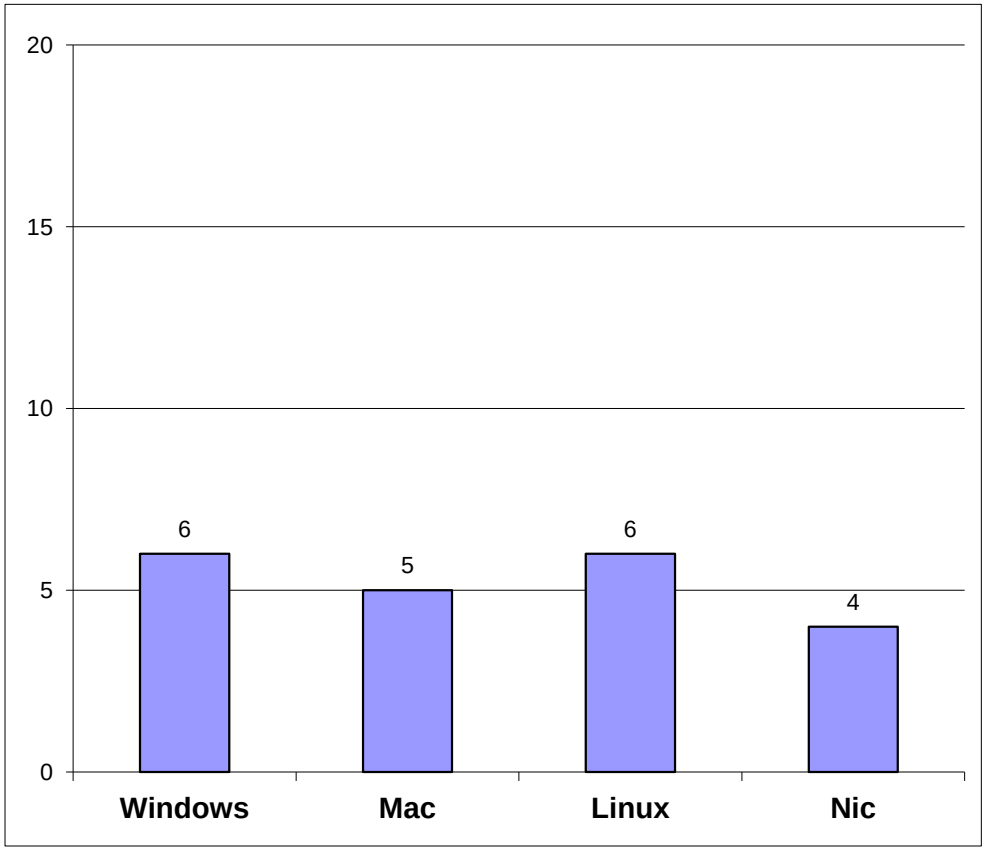
- obecności
- egzamin ustny

Ankieta wstępna: systemy operacyjne

Używam i/lub znam się:

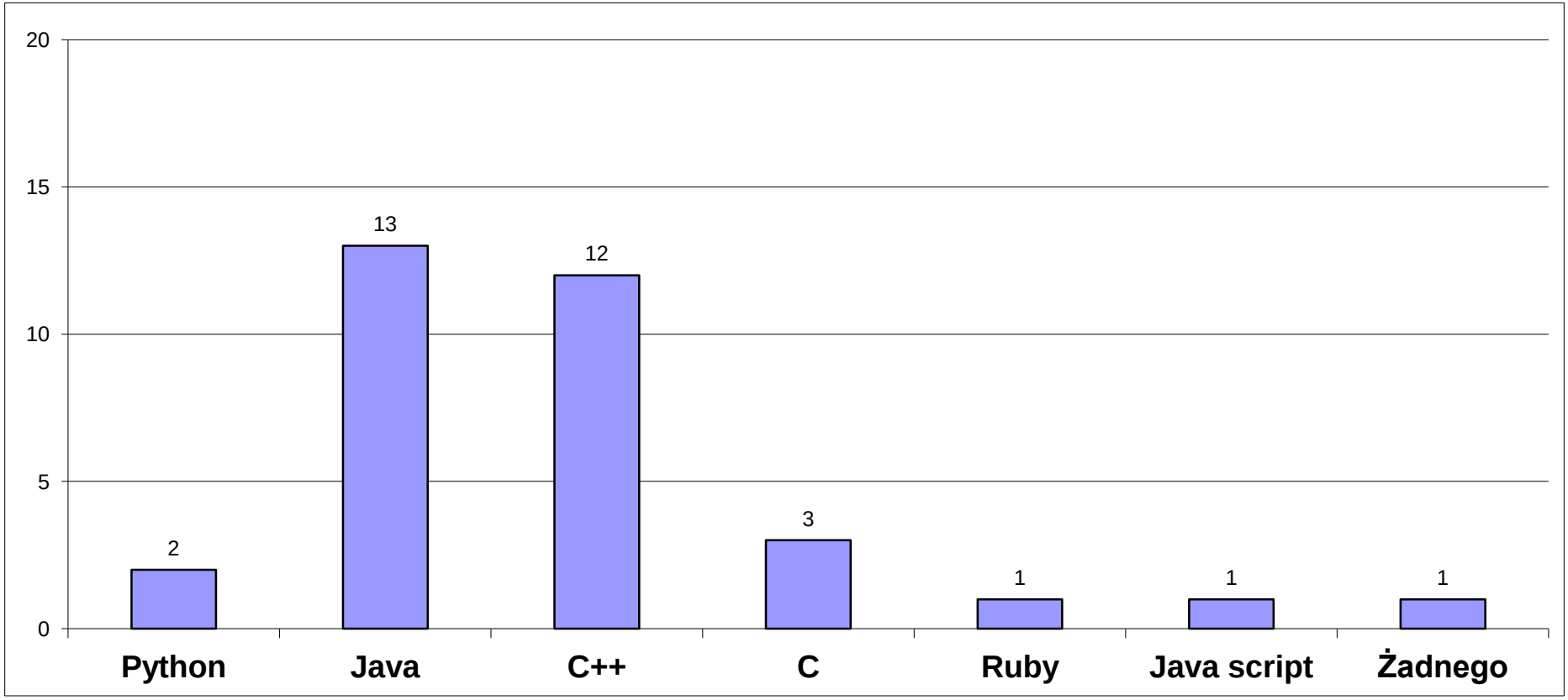


Chcę się nauczyć:



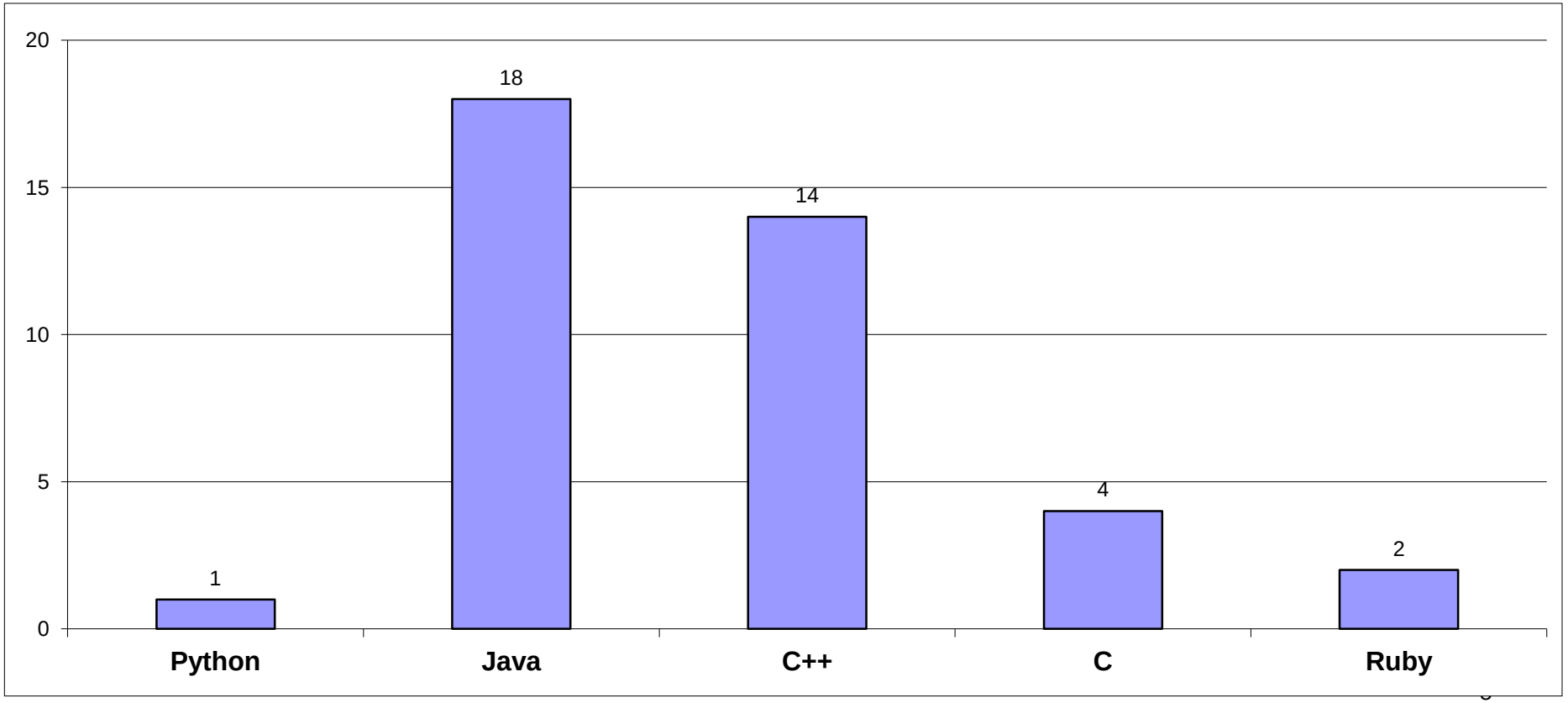
Ankieta wstępna: języki programowania

- 19/20 uważa że zna się na Pythonie
- Języki które chcielibyście opanować:

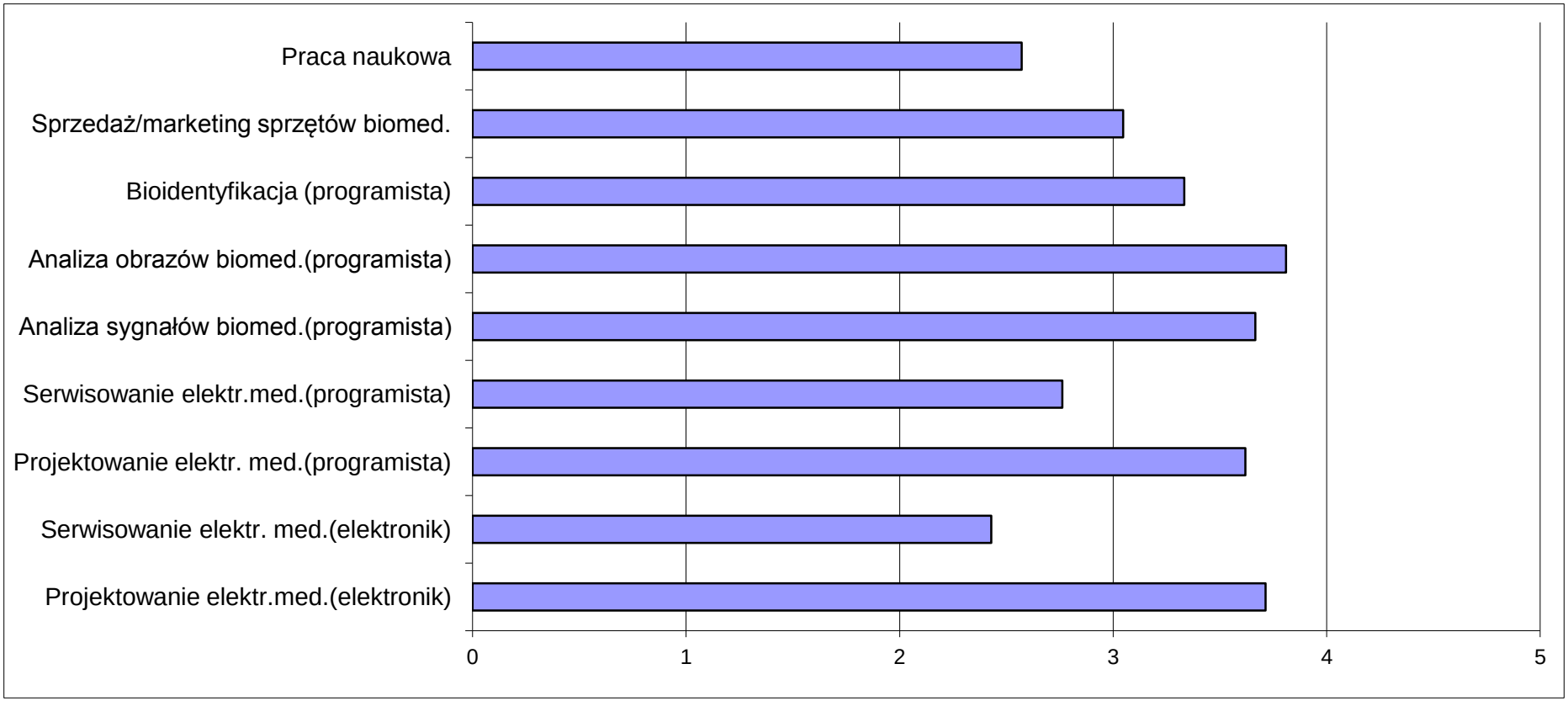


Ankieta wstępna: języki programowania

- Języki które uważacie że powinien znać programista:

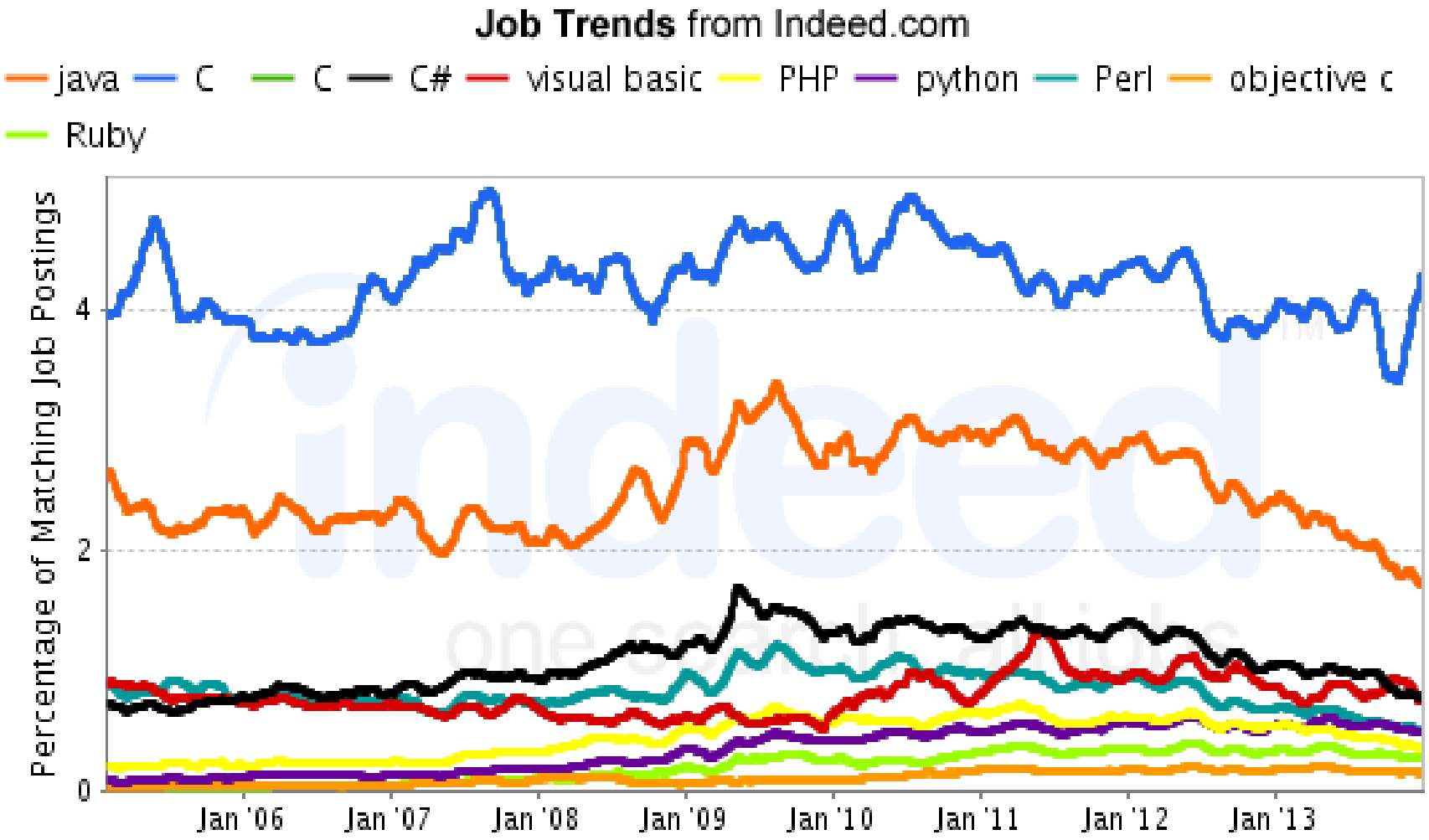


Ankieta wstępna: plany kariery / zainteresowania

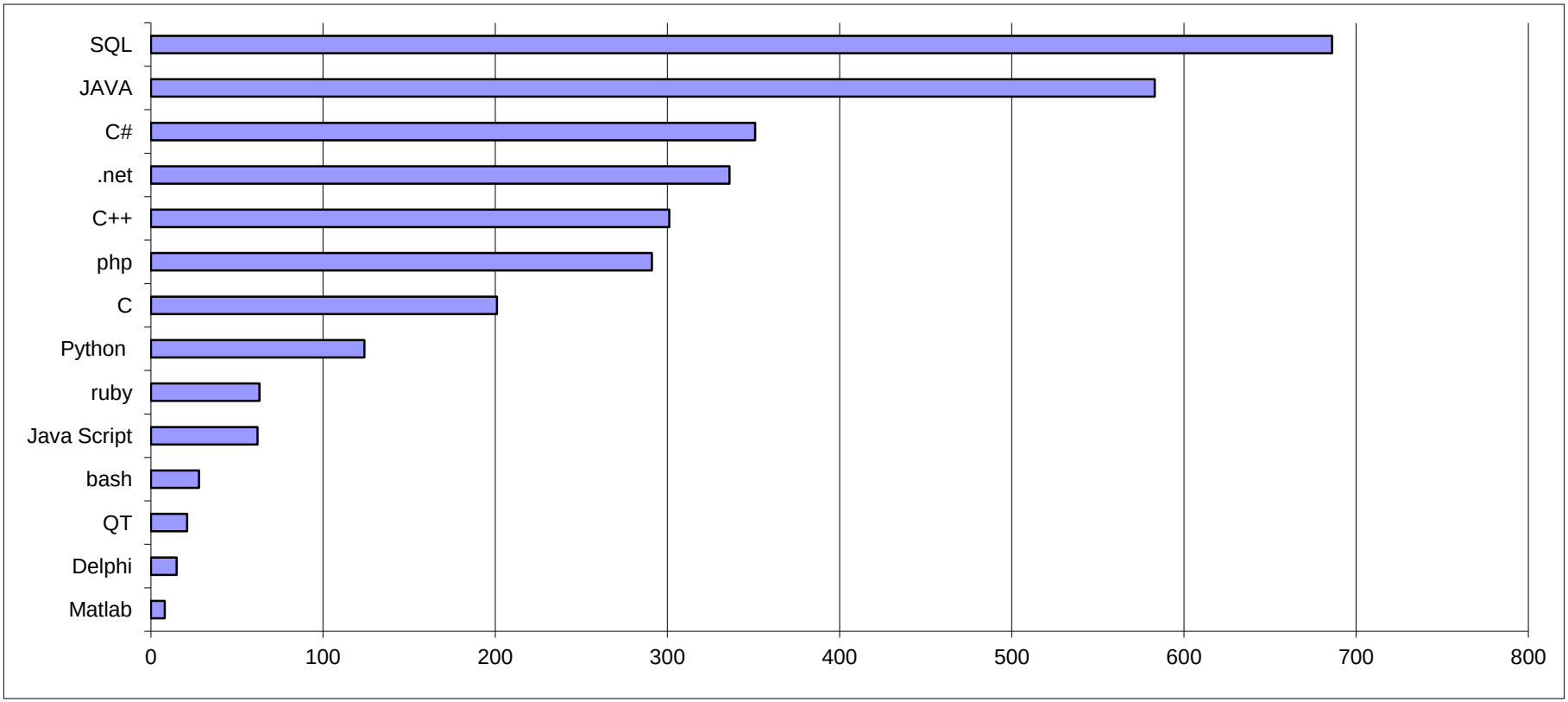


- Materiały biomed. Implanty. Sztuczne tkanki.

Poszukiwani programiści (ogłoszenia z indeed.com)



Poszukiwani programiści (ogłoszenia z praca.pl)



Myśleć jak programista

- Wiedza deklaratywna
  - *pierwiastek z x to liczba która spełnia równanie y * y = x*
- Wiedza imperatywna (proceduralna/praktyczna)
  - *zgadujemy jakieś g < x*
  - *sprawdzamy czy g * g – x < limit*
  - *jeżeli nie to nowe g = (g + x/g) / 2*
  - *zgadujemy aż uzyskamy zadowalające g*

6 kroków pracy programisty

1. Specyfikacja / definicja problemu
2. Projekt programu (design)
3. Programowanie (coding)
4. Testowanie (debugging)
5. Dokumentacja (documentation)
6. Pielęgnacja (maintenance)

Sposoby projektowania programów

- Pseudokod
- Schemat blokowy (Flowchart)
- Schemat modularny (Top-down)
- Programowanie organiczne (Bottom-up)

Zadanie z algorytmów 1

Napisz prosty algorytm w pseudokodzie lub schemacie blokowym wyliczający jak wydać resztę za pomocą najmniejszej liczby monet.

Do dyspozycji mamy monety:

1,2,5,10,20,50 gr

1,2,5 zł

Zadanie z algorytmów 2

Zaproponuj jak ułożyć 4 liczby a,b,c,d w rosnącej kolejności:

- sprawdzić wszystkie możliwe kombinacje („a ≥ b ≥ c ≥ d” ? „a ≥ c ≥ b ≥ d” … etc.)
- znaleźć najmniejszą/największą, przerzucić na początek/koniec
- porównywać po dwie sąsiadujące i zamieniać jeżeli są w złej kolejności

Klasyfikacje języków programowania:

- **poziom/generacja** (bliskość do maszyny/człowieka)
- **kompilator/interpreter** (sposób wykonywania programów)
- **paradygmat programowania** (np. imperatywne / obiektowe)

Generacje języków programistycznych

I – poziom maszynowy

```
0100111
```

II – poziom niski (assemblerowy)

```
MOV A1, A2
```

III – języki wysokiego poziomu

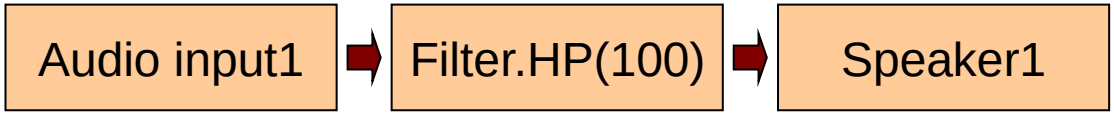
```
for(int i=0;i<n;i++)
```

IV – języki zadaniowe (SQL, Excel, LabView)

```
SELECT id FROM objects WHERE x = TRUE
```

V – języki naturalne / graficzne

(czy już istnieją? czy są tylko interfejsami do 3G and 4G?)



„Filter this audio input for me. Remove all frequencies below 100Hz.”

Kompilator vs. interpreter

Interpreter – tłumaczy jedną instrukcję na język maszynowy, czeka aż procesor ją wykona, tłumaczy następną (np. BASIC, języki skryptowe)

- powolne, wymaga interpretera na danym komputerze
+ duża przenośność i wykrywalność błędów

Kompilator – tłumaczy cały program, zazwyczaj tworząc „plik wykonywalny” (np. C, C++)

+ szybsze i nie wymagają kompilatora do uruchomienia
- słaba przenośność, większe ryzyko błędów

Hybryda: JAVA – posiada zarówno interpreter i kompilator – wszystkie programy kompiluje się na „wirtualną maszynę JAVA” która zainstalowana na dowolnym komputerze służy za interpreter skompilowanego już kodu (efektem jest nieporównywalna przenośność i względnie dobra szybkość)

Paradygmaty programowania (1)

- **Imperatywne**
  - Skupienie na instrukcjach, oddzielnie definiowane dane
  - Program wykonuje kolejno komendy manipulujące danymi

- **Obiektowe**
  - Połączenie stanów (danych) i zachowań (instrukcji, procedur, metod)
  - Większość instrukcji polega na komunikacji między obiektami

Paradygmaty programowania (2)

- **Strukturalne** – łączenie instrukcji w hierarchiczne bloki
- **Proceduralne** - grupowanie instrukcji w procedury
- **Modularne** –nadrzędność modułów nad blokami i procedurami
- **Funkcyjne** - skupienie na funkcjach, nie instrukcjach
- **Uogólnione** (generic) – nie wymaga twardego definiowania typów danych
- **Sterowane zdarzeniami** (event based) – mocno związene z wieloprocesowością i obiektowymi GUI
- **Deklaratywne (logiczne)** – liczy się wynik końcowy nie instrukcje potrzebne by go osiągnąć (języki 4G)
- **Agentowe** – wyższy poziom abstrakcji od obiektowego –niezależnie działające ale współpracujące mini-programy (software agents)
- **Hybrydowe** – większość istniejących języków programistycznych to połączenia powyższych paradygmatów

Pytania weryfikacyjne:

- Wiedza deklaratywna vs imperatywna
- 6 kroków pracy programisty
- Sposoby projektowania programów
- Klasyfikacje języków programowania
- Poziom/generacja języka programowania
- Kompilator vs. interpreter
- Paradygmaty programowania