The Nordic Prior Knowledge Test in Programming

Author: Sondre Sæther Bolland

Institution: Department of Informatics, University of Bergen

Email: sondre.bolland@uib.no

Introduction

The Nordic Prior Knowledge Test in Programming is a tool for assessing students' programming skill. The test covers the fundamental elements of introductory programming taught at different universities and university colleges in Norway and Sweden. By testing the students in the concepts found in CS1 we aim for instructors to be better able to develop and adapt their courses to this new found prior knowledge.

This Notebook is a dynamic report of the results from 2024, designed to effectivly convey the findings of the test at the start of the semester. The (static) written report will be published sometime fall 2024 here: https://programmeringstesten.no/

Imports

In [12]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import PercentFormatter
from scipy import stats
```

Data

Before using this notebook two python scripts need to be run. A requirement to run these scripts is to have the csv file with the original dataset: `total.csv`. This must be placed in the `data` folder in the main directory (where this Notebook is located).

`clean_data.py` cleans certain columns for ambigious data and renames columns for analysis.

`grade_submissions.py` grades each question based on the rubric (`rubric.json`).

In [88]:

```
#!python clean_data.py
#!python grade_submissions.py
```

Dataset

In [89]:
```
# Specify the path of the data file
path = "data/"
filename = path + "results.csv"
df = pd.read_csv(filename, on_bad_lines="skip", delimiter=";", encoding="utf8")

# Remove all students who have taken a university level course
df = df[(df['UniversityExperience'] == 'No')]
```

Demographics

See the background of the students. Note that there are a large number of submissions that are blank for some of the following items. This is due to the students submitting ambigious answers, which have not been labled correctly by `clean_data.py`.

In [54]:
```
# Total number of students
print(f"The dataset has {len(df['Total'])} student submissions.")
```

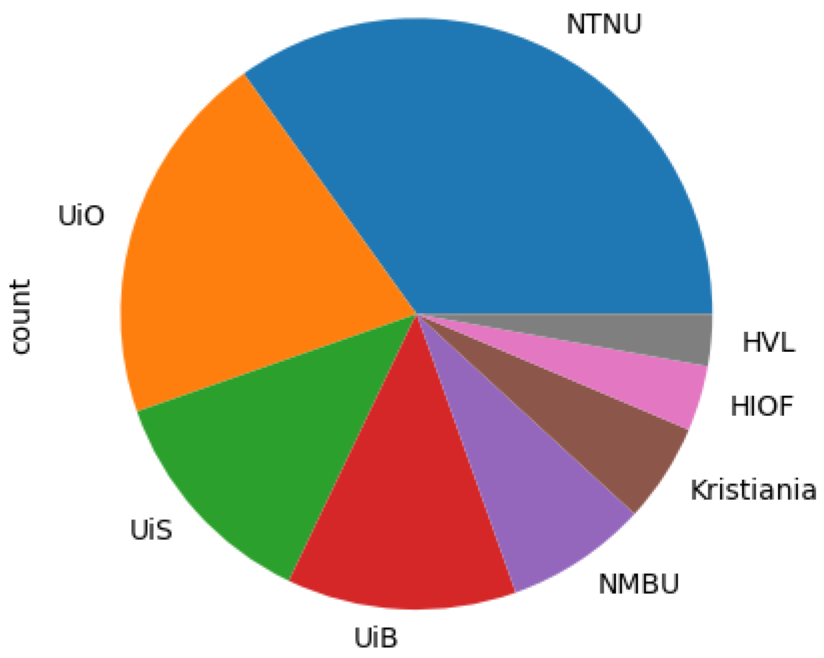
```
The dataset has 2661 student submissions.
```

Institutions

In [55]:
```
institutions = df.Institution.unique()

institution_column = "Institution"
print(df[institution_column].value_counts())
df[institution_column].value_counts().plot.pie()
```

```
Institution
NTNU          928
UiO           543
UiS           334
UiB           333
NMBU          206
Kristiania    144
HIOF           95
HVL            75
Name: count, dtype: int64
```

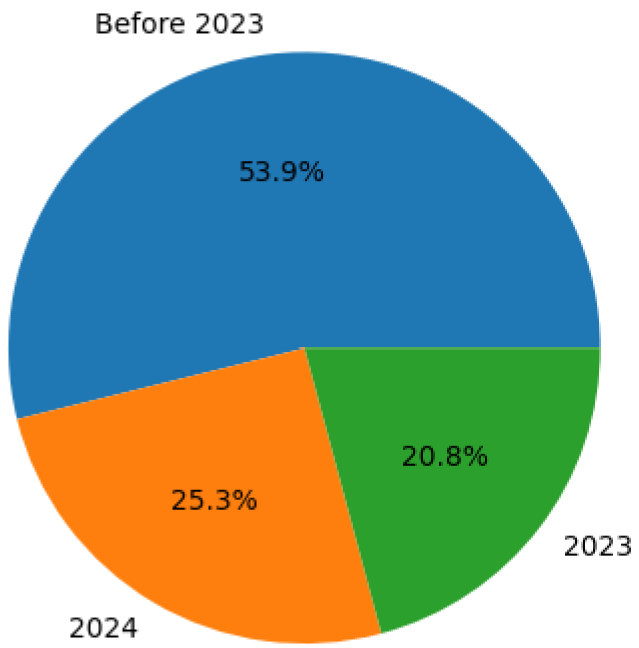
Out[55]:
```
<Axes: ylabel='count'>
```



Graduate Year

In [56]:
```
years = df.GraduateYear.unique()

graduateyear_column = "GraduateYear"
print(df[graduateyear_column].value_counts())
df[graduateyear_column].value_counts().plot.pie(autopct='%1.1f%%', ylabel='')
fig = plt.gcf()
plt.show()
fig.savefig('plots/graduateYear.png',dpi=300, bbox_inches='tight')
```

```
GraduateYear
Before 2023    1433
2024            674
2023            554
Name: count, dtype: int64
```



Educational Background

What experiences with programming do these students have?

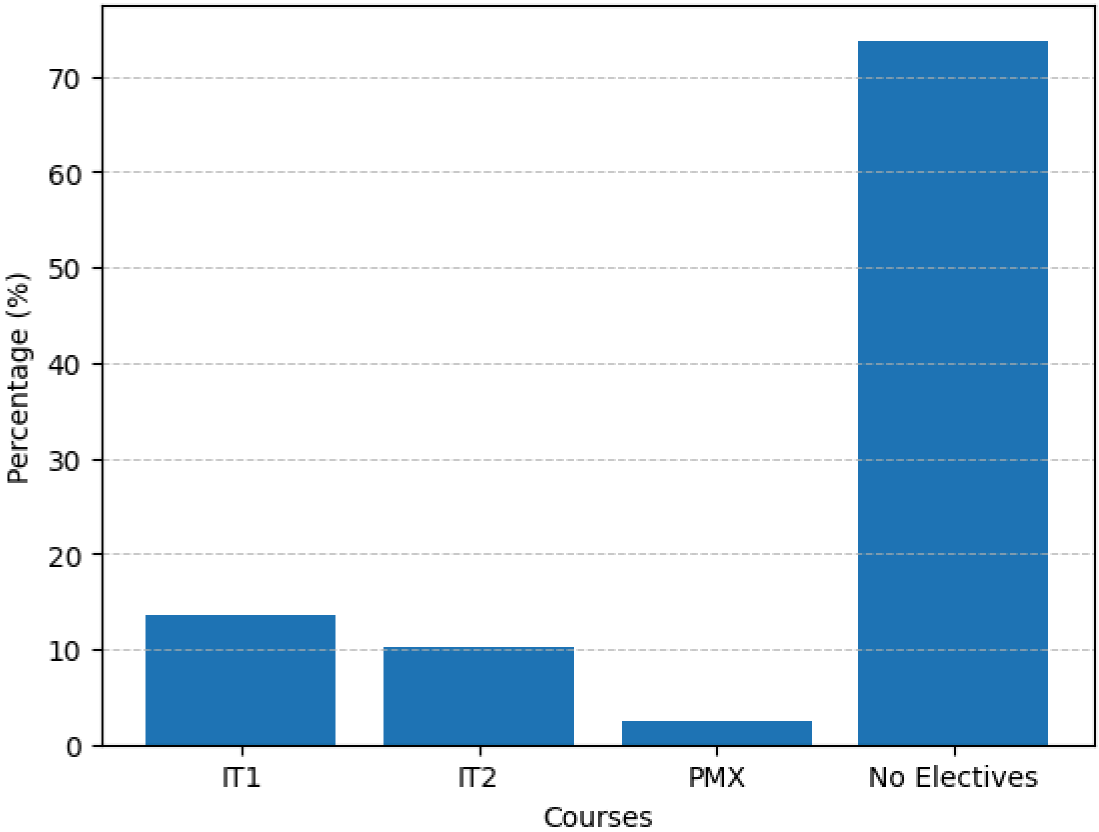
Elective Programming Courses

During the secondary school phase, students have the option to take three elective courses in programming: *Information Technology 1* (IT1), *Information Technology 2* (IT2), and *Programming and Modelling X* (PMX).

In [57]:
```
# Make new column for NO elective programming course
courses = ['ElectiveCourse.IT1', 'ElectiveCourse.IT2', 'ElectiveCourse.PMX']
df_temp = df[(~df['ElectiveCourse.IT1'].isin(courses))]
df_temp = df_temp[(~df_temp['ElectiveCourse.IT2'].isin(courses))]
df_temp = df_temp[(~df_temp['ElectiveCourse.PMX'].isin(courses))]
df_temp['NoElective'] = ~df_temp['ElectiveCourse.IT1'].isin(courses)

df['NoElective'] = df_temp['NoElective']
```

In [84]:
```
# Set up figure and axis
fig, ax = plt.subplots()

# Define courses and counts
courses = ['IT1', 'IT2', 'PMX', 'No Electives']
n_it1 = df['ElectiveCourse.IT1'].value_counts()[1]
n_it2 = df['ElectiveCourse.IT2'].value_counts()[1]
n_pmx = df['ElectiveCourse.PMX'].value_counts()[1]
n_noelectives = df['NoElective'].value_counts()[True]
counts = [n_it1, n_it2, n_pmx, n_noelectives]

# Calculate percentages
total_students = sum(counts)
```

```
percentages = [(count / total_students) * 100 for count in counts]

# Plot the bar chart using percentages
ax.bar(courses, percentages)
ax.set_ylabel('Percentage (%)')
ax.set_xlabel('Courses')

ax.yaxis.grid(True, linestyle='--', linewidth=0.7, alpha=0.7)  # Horizontal grid

# Show the plot
plt.show()
```



Math Courses

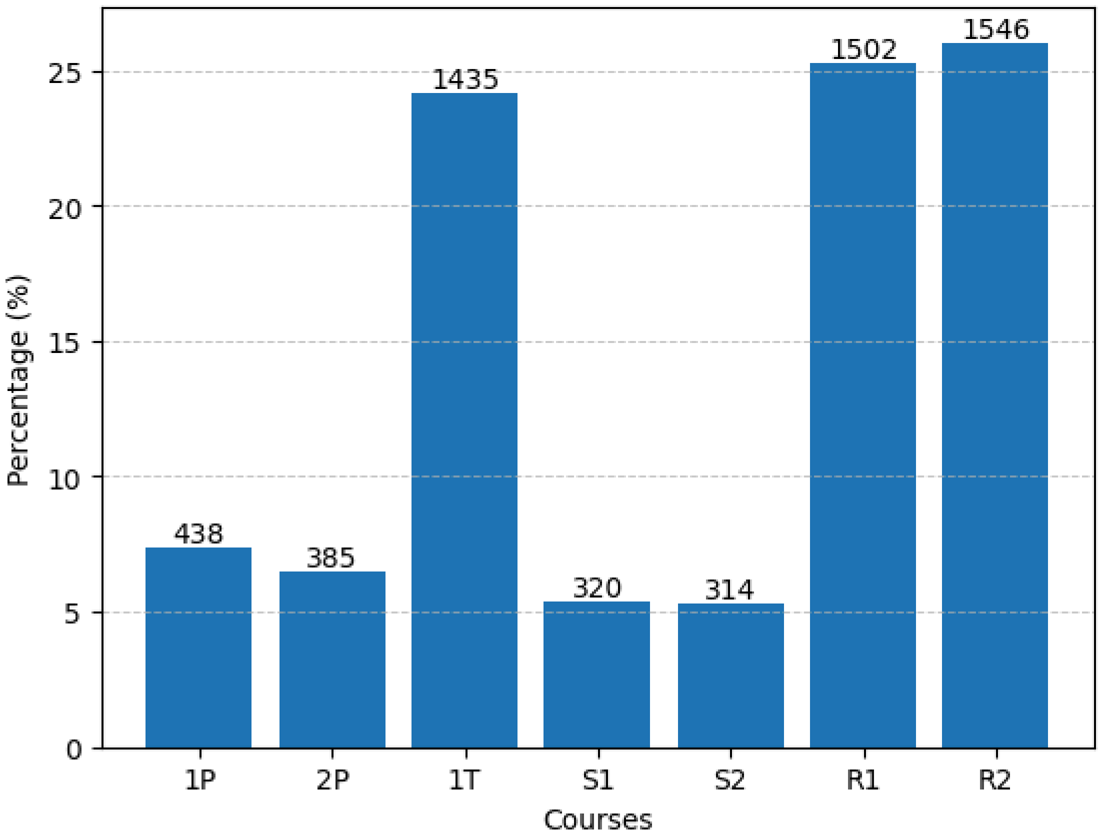
The most common math courses available in secondary school are:

- Practical Math 1 (1P)
- Practical Math 2 (2P)
- Theoretical Math 1 (1T)
- Social Science Math 1 (S1)
- Social Science Math 2 (S2)
- Natural Science Math 1 (R1)
- Natural Science Math 2 (R2)

The majority of students who took the test belonged to STEM fields, where the typical admission requirement includes S1 and S2 or R1 mathematics. Certain math-intensive study programs may also demand R2 mathematics. Notably, most students had completed the Natural Science Math courses (see plot below), which is the most advanced option.

In [85]:
```
# Set up figure and axis
fig, ax = plt.subplots()

# Define courses
courses = ['MathCourse.1P', 'MathCourse.2P', 'MathCourse.1T', 'MathCourse.S1', '
course_titles = ['1P', '2P', '1T', 'S1', 'S2', 'R1', 'R2']


# Initialize lists to store course data
gotten_courses = []
counts = []

# Collect counts for each course
for course in courses:
    try:
        count = df[course].value_counts()[1]
        counts.append(count)
        gotten_courses.append(course)
    except KeyError:
        continue

# Calculate percentages
total_students = sum(counts)
percentages = [(count / total_students) * 100 for count in counts]

# Plot the bar chart using percentages
bars = ax.bar(course_titles, percentages)
ax.set_ylabel('Percentage (%)')
ax.set_xlabel('Courses')

# Add grid lines
ax.yaxis.grid(True, linestyle='--', linewidth=0.7, alpha=0.7)

# Add student counts on top of each bar
for bar, count in zip(bars, counts):
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width() / 2, height, f'{count}', ha='center',

# Show the plot
print(counts)
plt.show()

# Save the figure
fig.savefig('plots/courses_percentage.png', dpi=300, bbox_inches='tight')
```

```
[438, 385, 1435, 320, 314, 1502, 1546]
```



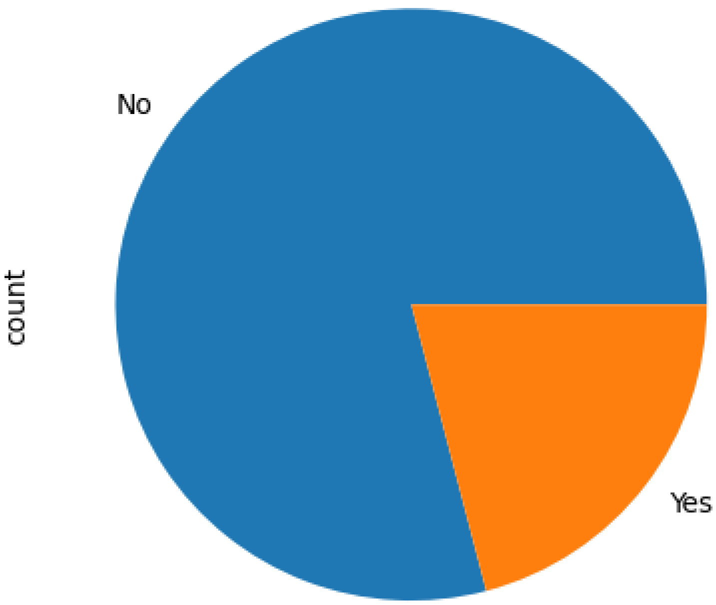
Experience outside of school

A source of programming knowledge is self-directed learning outside of formal education, where individuals independently explore the field, using resources like books and online materials. The following plot shows the distribution of students who report having at least 30 hours of outside experience with either block based or text based programming.

In [60]:
```
outside_column = "OutsideExperience"

print(df[outside_column].value_counts())
print(df[outside_column].value_counts(normalize=True))
df[outside_column].value_counts().plot.pie()
```

```
OutsideExperience
No     2098
Yes     555
Name: count, dtype: int64
OutsideExperience
No     0.790803
Yes    0.209197
Name: proportion, dtype: float64
```

Out[60]:
```
<Axes: ylabel='count'>
```



Results

In this section of the report, the main results are presented before a review of the background the students have on the various study paths and what connection there is between background and results. Later we take a close look at how well they performed in specific programming tasks to understand their grasp of the different concepts.

In [61]:
```
# Configurations
bins = 20
max_points = 40.6 # i.e. range
edgecolor = 'black'
alpha = 0.5
```

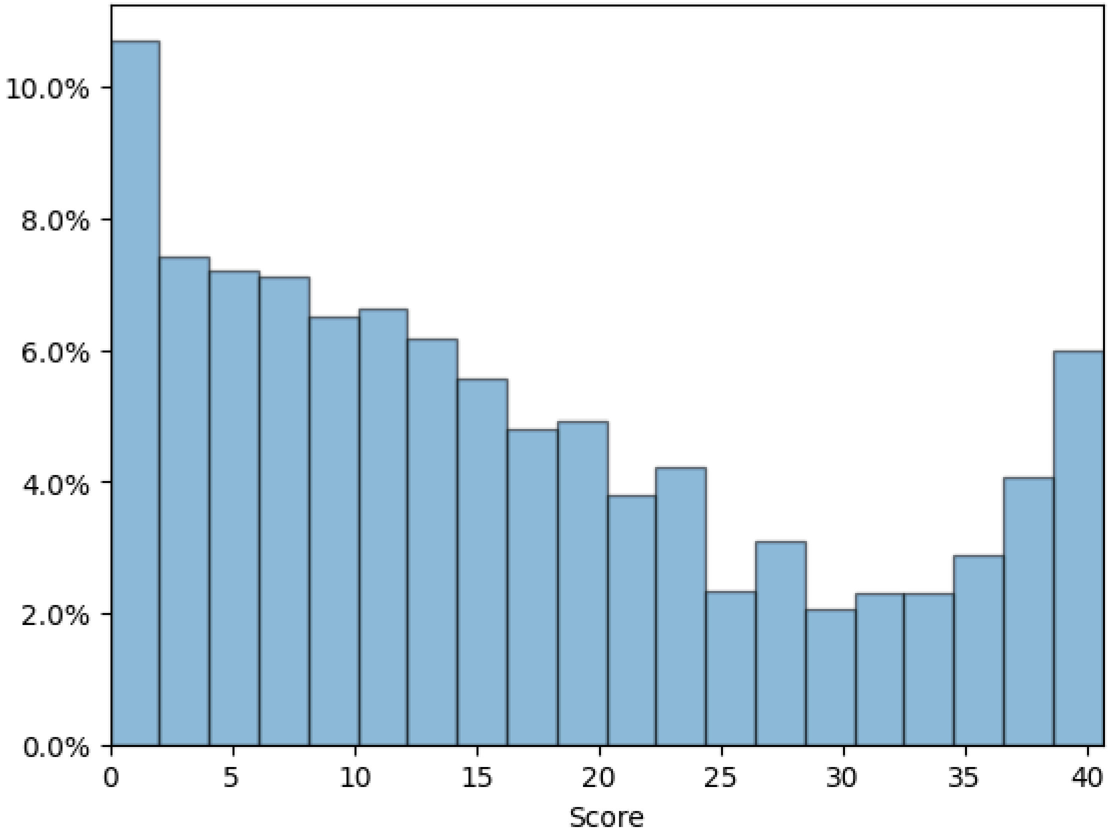
Main Result

A histogram of the total score of the students. Maximum score: 71.

In [82]:
```
x = df['Total']
x_label = 'Score'

mean = df.loc[:, 'Total'].mean()
sd = df.loc[:, 'Total'].std()
print(f'Mean: {round(mean, 3)}')
print(f'Standard deviation: {round(sd, 3)}')

plt.hist(x, bins=bins, alpha=alpha, edgecolor=edgecolor, label=x_label,
         weights=np.ones(len(x)) / len(x), range=[0, max_points])
plt.gca().yaxis.set_major_formatter(PercentFormatter(1))

# Add x and y axis labels
```

```
plt.xlabel('Score')  # Label for the x-axis
#plt.ylabel('Percentage of students')  # Label for the y-axis

plt.margins(x=0.0001)
fig = plt.gcf()
plt.show()
fig.savefig('plots/allHist.png', dpi=300, bbox_inches='tight')
```

```
Mean: 16.189
Standard deviation: 12.074
```

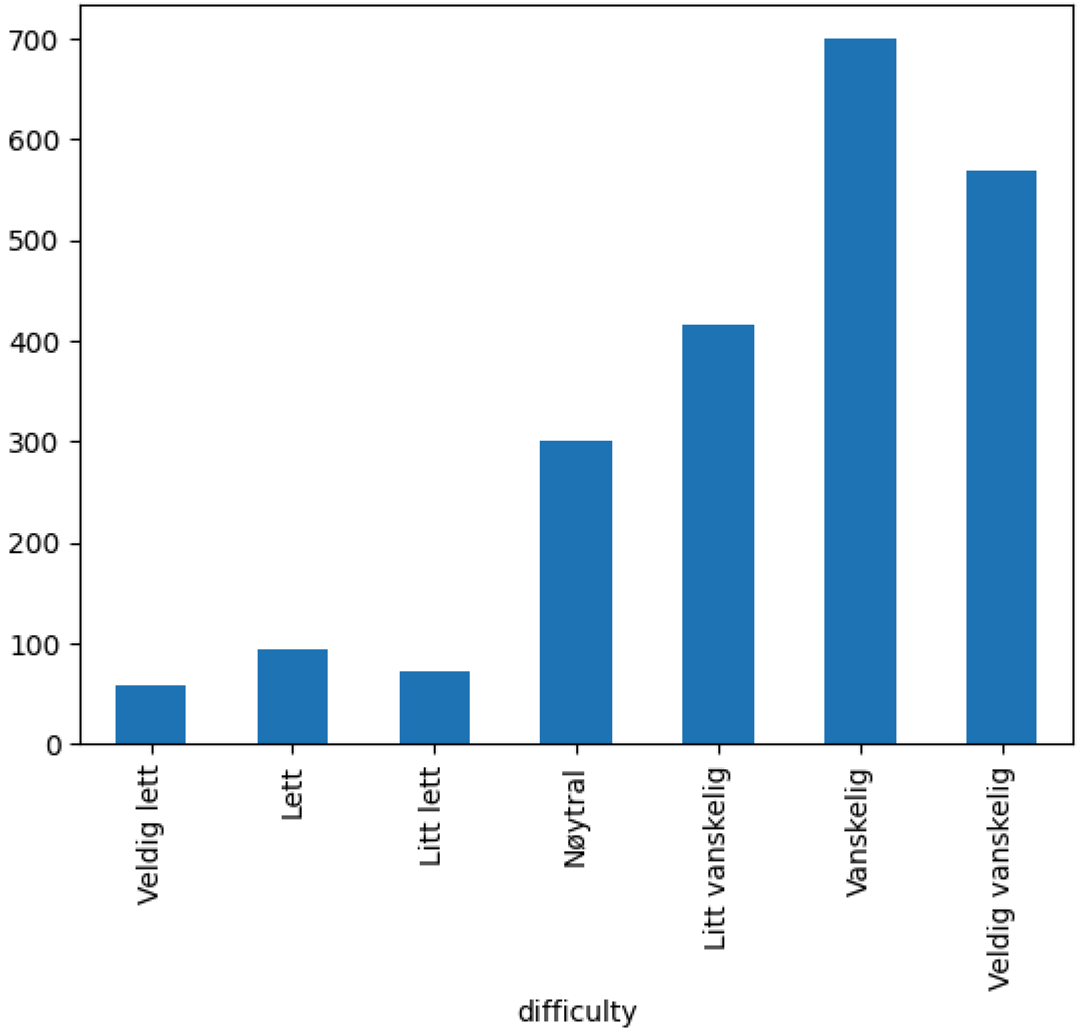


Did the students find the test hard?

We asked the students to rate the test's difficulty on a Likert scale from 1 (very easy) to 7 (very hard)

In [63]:
```
df['difficulty'].value_counts().loc[['Veldig lett', 'Lett', 'Litt lett', 'Nøytra
```

Out[63]:
```
<Axes: xlabel='difficulty'>
```



In [79]:
```
def compare_distributions(x, y, x_label, y_label, save_figure_name=''):
    n = len(x)
    mean = x.mean()
    sd = x.std()
    print(f'N students in {x_label}: {n}')
    print(f'Mean of {x_label}: {round(mean, 3)}')
    print(f'Standard deviation of {x_label}: {round(sd, 3)}')
    print()

    n = len(y)
    mean = y.mean()
    sd = y.std()
    print(f'N students in {y_label}: {n}')
    print(f'Mean of {y_label}: {round(mean, 3)}')
    print(f'Standard deviation of {y_label}: {round(sd, 3)}')

    plt.hist(x, bins=bins, alpha=alpha, edgecolor=edgecolor, label=x_label, weig
    plt.hist(y, bins=bins, alpha=alpha, edgecolor=edgecolor, label=y_label, weig
    plt.gca().yaxis.set_major_formatter(PercentFormatter(1))

    # Add x and y axis labels
    plt.xlabel('Score')  # Label for the x-axis
    #plt.ylabel('Percentage of students')  # Label for the y-axis

    plt.margins(x=0.0001)
    plt.legend(loc='upper right')
    fig = plt.gcf()
    plt.show()
```

```
    if save_figure_name != '':
        fig.savefig('plots/'+save_figure_name+'.png', dpi=300, bbox_inches='tigh
```

Prior Programming Experience in Secondary School

In the initial segment of the test, we inquired with the students regarding their prior exposure to programming before to commencing their higher education studies.

Graduation Year

The educational reforms outlined in LK20 were introduced in the year 2020, resulting in programming becoming a compulsory component solely for those students who graduated in 2023 and onward.

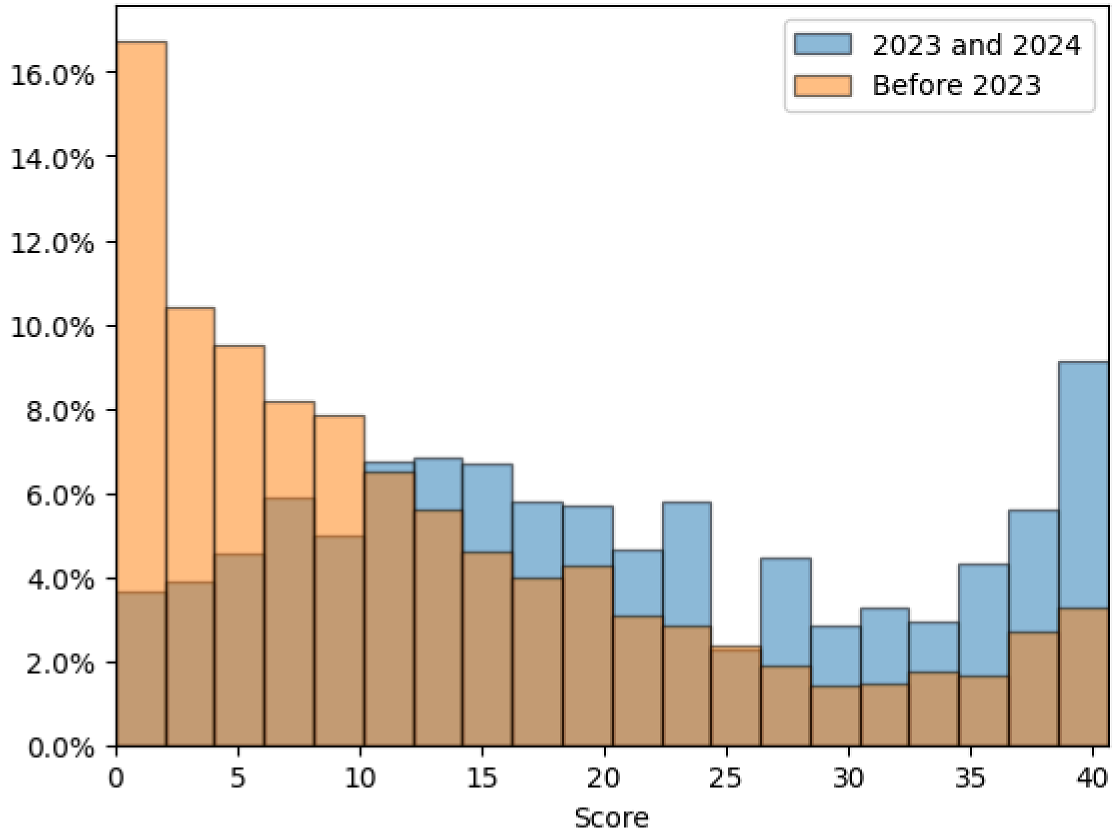
In [80]:

```
# Significance test
def significane(group1, group2):
    # Perform Mann-Whitney U Test
    stat, p_value = stats.ranksums(group1, group2)

    # Print the test statistic and p-value
    print(f"Wilcox Rank-sum test: {stat}")
    print(f"P-value: {p_value}")

    # Interpretation of p-value
    a = 0.05
    if p_value < a:
        print("Reject the null hypothesis: There is a significant difference bet
    else:
        print("Fail to reject the null hypothesis: There is no significant diffe
```

In [81]:

```
years = df['GraduateYear']
x = df[(years.isin(['2023', '2024']))]['Total']
y = df[(~years.isin(['2023', '2024']))]['Total']
y = y[~np.isnan(y)]

x_label = '2023 and 2024'
y_label = 'Before 2023'

compare_distributions(x, y, x_label, y_label, 'graduateYearHist')
significane(x, y)
```

```
N students in 2023 and 2024: 1228
Mean of 2023 and 2024: 20.253
Standard deviation of 2023 and 2024: 11.809

N students in Before 2023: 1433
Mean of Before 2023: 12.707
Standard deviation of Before 2023: 11.18
```

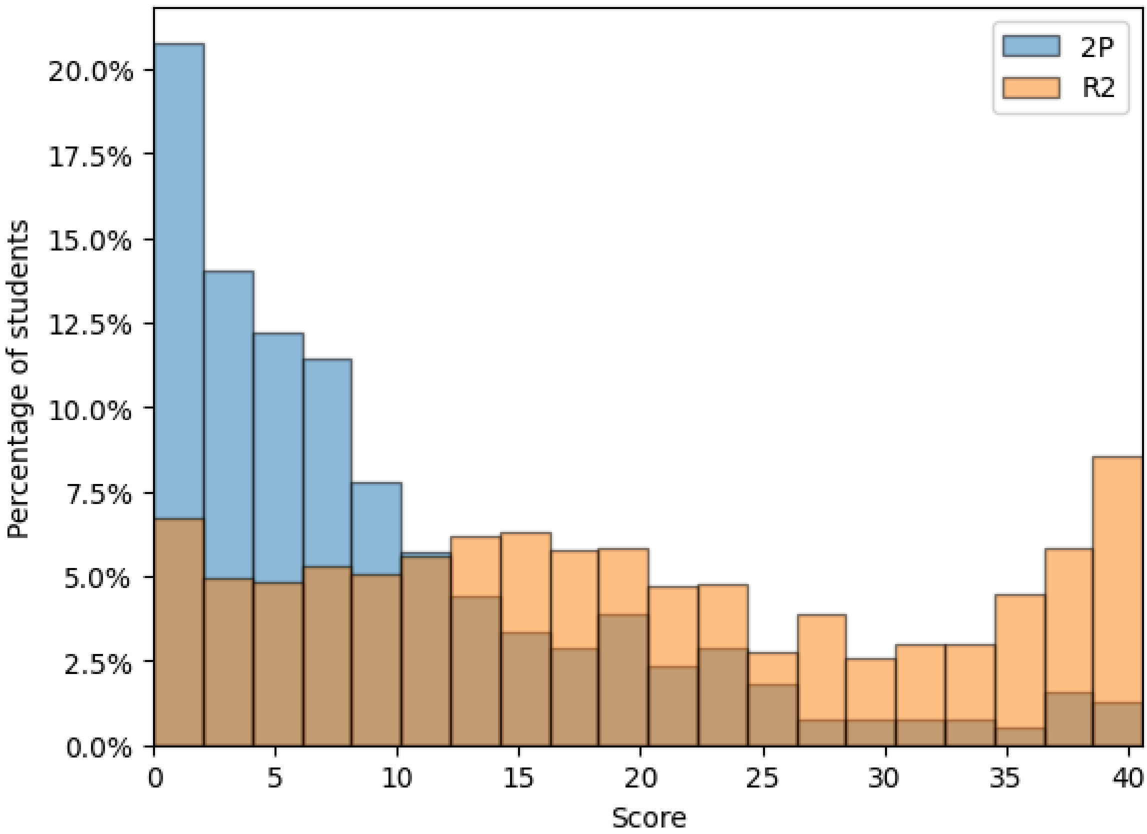



```
Wilcox Rank-sum test: 16.994743599138264
P-value: 8.9822258328567e-65
Reject the null hypothesis: There is a significant difference between the two gro
ups.
```

Mathematics Courses

In [67]:
```
x = df[(df['MathCourse.2P'].isin([1]))]['Total']
y = df[(df['MathCourse.R2'].isin([1]))]['Total']
y = y[~np.isnan(y)]

x_label = '2P'
y_label = 'R2'

compare_distributions(x, y, x_label, y_label)
significane(x, y)
```

```
N students in 2P: 385
Mean of 2P: 9.574
Standard deviation of 2P: 9.339

N students in R2: 1546
Mean of R2: 19.518
Standard deviation of R2: 12.26
```

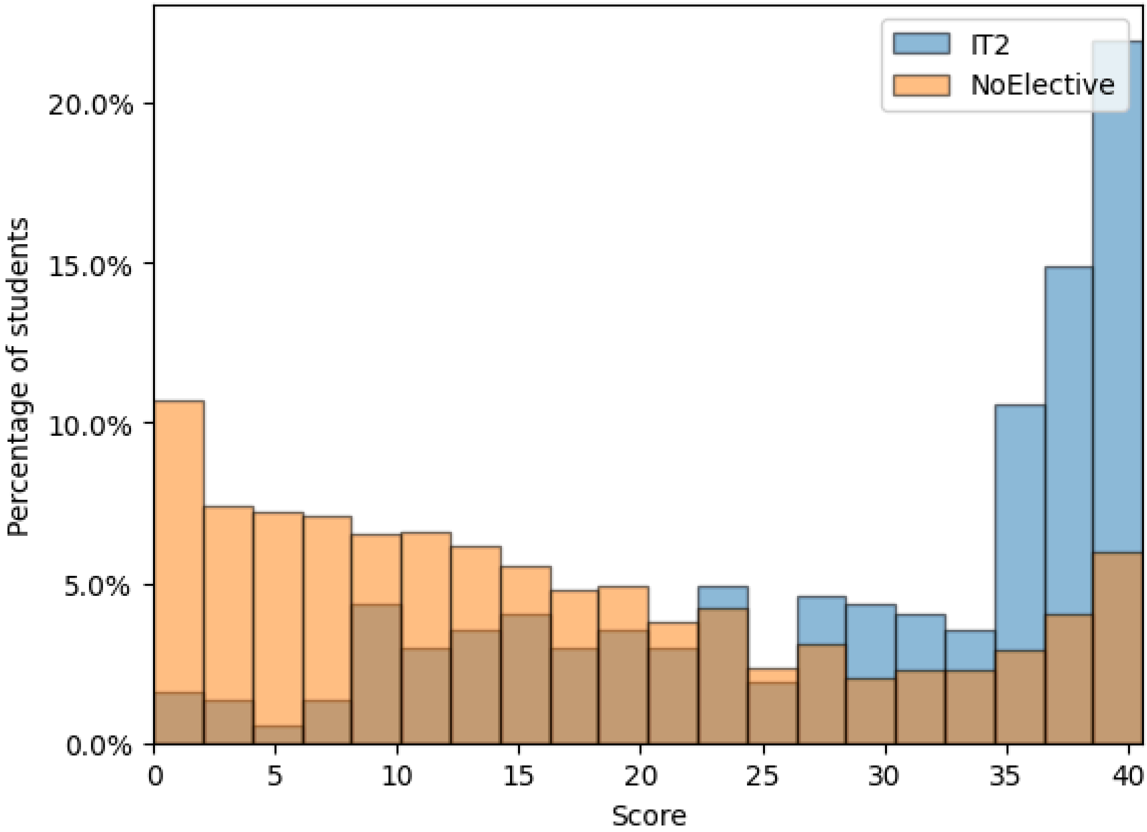



```
Wilcox Rank-sum test: -14.741318890551712
P-value: 3.498843585414536e-49
Reject the null hypothesis: There is a significant difference between the two gro
ups.
```

Programming Electives

In [68]:
```
x = df[(df['ElectiveCourse.IT2'].isin([1]))]['Total']
y = df[(df['NoElective'].isin([True]))]['Total']
y = y[~np.isnan(y)]

x_label = 'IT2'
y_label = 'NoElective'

compare_distributions(x, y, x_label, y_label)
significane(x, y)
```

```
N students in IT2: 369
Mean of IT2: 28.501
Standard deviation of IT2: 11.296

N students in NoElective: 2661
Mean of NoElective: 16.189
Standard deviation of NoElective: 12.074
```

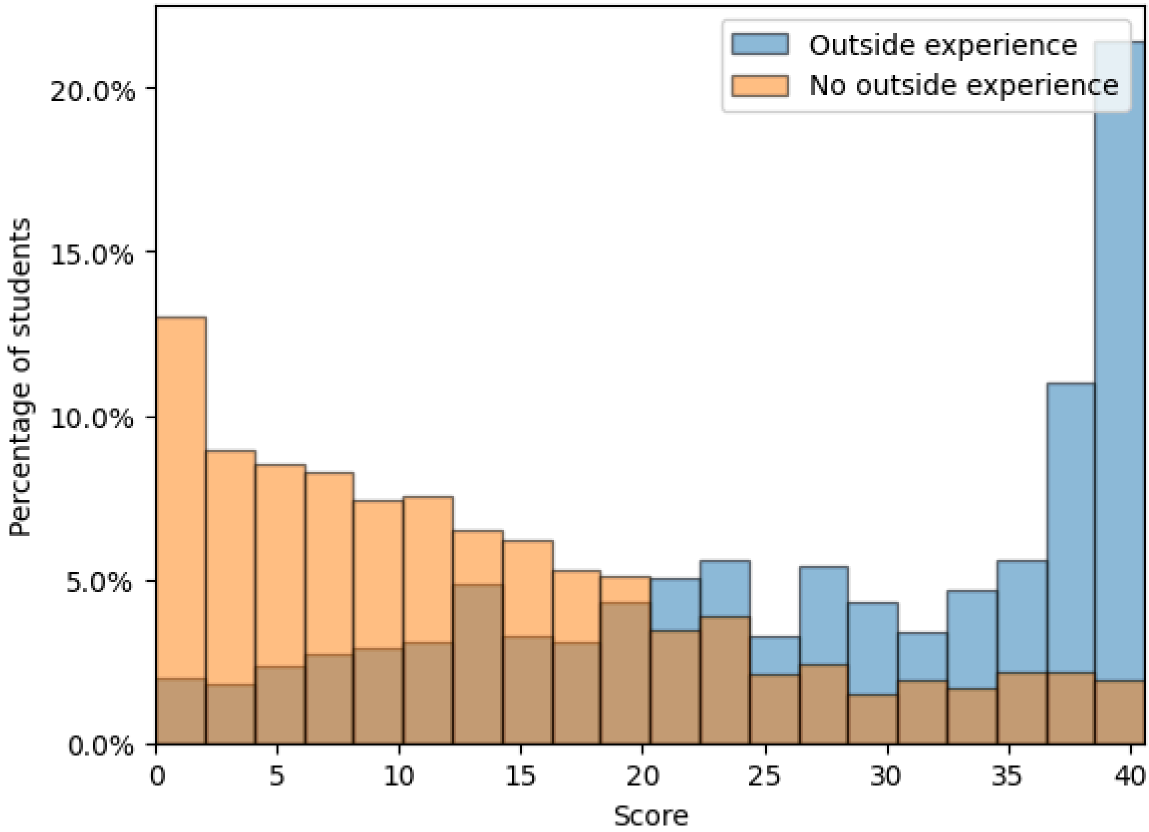



```
Wilcox Rank-sum test: 16.60270756439821
P-value: 6.661694809416955e-62
Reject the null hypothesis: There is a significant difference between the two gro
ups.
```

Outside Experience

In [69]:
```
x = df[(df['OutsideExperience'].isin(['Yes']))]['Total']
y = df[(df['OutsideExperience'].isin(['No']))]['Total']
y = y[~np.isnan(y)]

x_label = 'Outside experience'
y_label = 'No outside experience'

compare_distributions(x, y, x_label, y_label)
significane(x, y)
```

```
N students in Outside experience: 555
Mean of Outside experience: 26.681
Standard deviation of Outside experience: 11.716

N students in No outside experience: 2098
Mean of No outside experience: 13.402
Standard deviation of No outside experience: 10.543
```




```
Wilcox Rank-sum test: 21.433379979201103
P-value: 6.526591735513856e-102
Reject the null hypothesis: There is a significant difference between the two gro
ups.
```

How many students do not need introductory programming?

The students who perform very well on this test might not need CS1. If we set an (arbitrary) point threshold we can get an indication of how many students can perhaps move on to more advanced courses.

In [87]:
```python
threshold = round(max_points * 0.9, 2) # 90%
x = df[df['Total'] > threshold]
n = len(x)
percentage = round((n/len(df))*100, 2)

print(f'Number of students with a score over {threshold}: {n} ({percentage}%)')
```

```
Number of students with a score over 36.54: 267 (10.03%)
```

How many students have not learned much?

In [86]:
```python
threshold = round(max_points * 0.2, 2) # 20%
x = df[df['Total'] < threshold]
n = len(x)
percentage = round((n/len(df))*100, 2)

print(f'Number of students with a score under {threshold}: {n} ({percentage}%)')
```

```
Number of students with a score under 8.12: 863 (32.43%)
```

Programming Tasks

Each task featured in the test pertained to a designated concept category. The following cells show the number of correct answers and the most common answers for each task.

Correctness rate for each concept category

Below you can find the percentage of accurate responses achieved by the students for each concept.

Note that performance within each category may not exclusively reflect the students' mastery of that programming concept. Variability in task difficulty plays a substantial role, with some tasks naturally being easier than others, irrespective of the underlying concept.

In [75]:
```
def correctnes_rate(columns, topic, task_weight=1):
    correct_answer_rate = 0
    for column in columns:
        points_column = column + "_points"
        task_mean = df.loc[:, points_column].mean()
        correct_answer_rate += task_mean
    correct_answer_rate /= len(columns)*task_weight
    print(f'{topic:15s} {correct_answer_rate*100:3.2f} %')
```

In [76]:
```
print("The students have the following average correctness rate for each program

# Datatypes
datatype_columns = ['Datatypes1', 'Datatypes2', 'Datatypes3', 'Datatypes4']
correctnes_rate(datatype_columns, "Datatypes")

# Operators
operator_columns = ['Operators1', 'Operators2', 'Operators3', 'Operators4', 'Ope
correctnes_rate(operator_columns, "Operators")

# Booleans
boolean_columns = ["100 == 100", "2 > 7", "(10 + 3) >= 13", "(10*2) < 9", "(1 +
correctnes_rate(boolean_columns, "Booleans", task_weight=0.2)

# Variables
variable_columns = ["Variables1a", "Variables1b", "Variables2a", "Variables2b",
correctnes_rate(variable_columns, "Variables", task_weight=0.2)

# Conditionals
conditional_columns = ["Conditionals1", "Conditionals2", "Conditionals3", "Condi
correctnes_rate(conditional_columns, "Conditionals")

# Loops
loop_columns = ["Loops1", "Loops2", "Loops3", "Loops4", "Loops5", "Loops6", "Loc
correctnes_rate(loop_columns, "Loops")

# Lists
list_columns = ["Lists1", "Lists2", "Lists3a", "Lists3b"]
correctnes_rate(list_columns, "Lists")
```

```
# Functions
function_columns = ["Functions1", "Functions2", "Functions3", "Functions4", "Fun
correctnes_rate(function_columns, "Functions")

all_columns = datatype_columns + operator_columns + boolean_columns + variable_c
```

```
The students have the following average correctness rate for each programming top
ic:
Datatypes       51.46 %
Operators       53.25 %
Booleans        60.67 %
Variables       49.47 %
Conditionals    55.06 %
Loops           27.01 %
Lists           31.79 %
Functions       20.07 %
```

In [ ]: