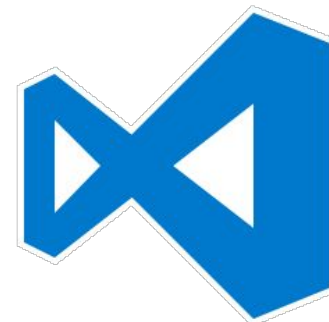




VS Code használata C és Python környezetben

```
def  
return :+5  
  
def dotwrite(ast):  
    nodename = get_nodename()  
    label=symbol_table.get(ast[1],ast[0])  
    print ' %s [%s]' % (nodename, label)  
    if isinstance(ast[1], str):  
        if ast[1].strip():  
            print '= %s:' % ast[1]  
        else:  
            print '['  
    else:  
        print '];'  
        children = []  
        for n, child in enumerate(ast[1:]):  
            children.append(dotwrite(child))  
        print ', %s -> {' % nodename  
        for i, name in enumerate(children):  
            print '%s' % name,
```

c-ben írt python modulon keresztül



Felhasznált irodalom és internetes tartalom

- A szerkesztő bemutatása a hivatalos Visual Studio Code dokumentáció alapján lett létrehozva
<https://code.visualstudio.com/docs>
- Példakódhoz vett forrásanyag
<https://learn.microsoft.com/en-us/visualstudio/python/working-with-c-cpp-python-in-visual-studio?view=vs-2022>
- A példakódok a <https://highlightjs.org/> használatával lettek színezve a base16-atelier-heath-light témát használva

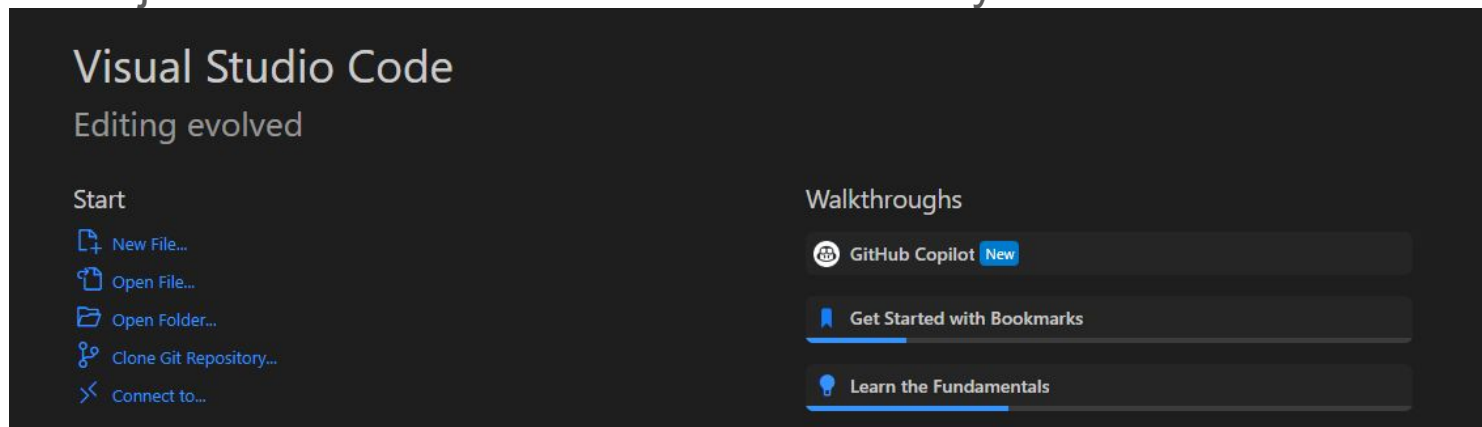
Visual Studio Code bemutatása

- A Visual Studio Code egy nyílt forráskódú több platformos kódszerkesztő amit 2015-ös bejelentése óta a Microsoft fejleszt. Alapja az Electron framework, ami egy NodeJs környezetben futó böngésző alapú asztali alkalmazásfejlesztő keretrendszer. A fő kódok Javascript-ben és C-ben íródnak a megjelenítésért HTML CSS és Javascript felel.
- GitHub-on elérhető a kód és a bináris telepítő is több platformra <https://github.com/microsoft/vscode>
- Több programnyelvhez beépített IntelliSense támogatást tartalmaz, ami szintaktikai kiemeléseket és ellenőrzést és kódkiegészítést végez
- Saját bővítmény könyvtárral rendelkezik (Visual Studio Code Marketplace)
- A beállítások online fiók összekapcsolásával szinkronizálhatóak több gép használata esetén (Github vagy Microsoft fiók használatával)

- Több füles és osztott képernyős elrendezésű, megnyitható egy fájl is, de hatékonyabb a Munkaterület kezelését használni ami GUI-n keresztül és Json szerkesztéssel is testre szabható. Kinézete témák segítségével alakítható
- A parancsok többsége menüből vagy gyorsbillentyűkkel elérhető, de az összes parancs egy parancskészletből előhívható (ctrl + shift + p)
- Beépített verziókövető támogatást tartalmaz ami az oldalsávban elérhető és alapértelmezetten a GIT
- beépített Debugger támogatással rendelkezik, ami több nyomkövetési módot támogat és paraméterezhető
- Beépített tesztelő oldalpanelen keresztül szintén több tesztelési módot támogat
- Általában véve testreszabható ezáltal több keretrendszer, fejlesztési platform alapja lett a megjelenése óta pl.: PlatformIO
- Online is elérhető <https://vscode.dev>

VS code telepítése windows környezetben

- A telepítő letölthető a <https://code.visualstudio.com/> oldalról
- A letöltött VSCodeUserSetup-x64-1.82.0.exe elindításával a program a szerződés elfogadása után gyorsan telepíthető (a fájl neve a verziószámot és a platformot is tartalmazza ezért ez a későbbiekben eltérhet)
- A telepítés végeztével a szerkesztő elindul egy üdvözlő képernyővel amin egy funkció bemutató is elérhető (angol nyelven) Get Started with VS Code vagy kezdhetjük a kódszerkesztést a Start alatt elhelyezkedő Linkekkel is.



Program részei

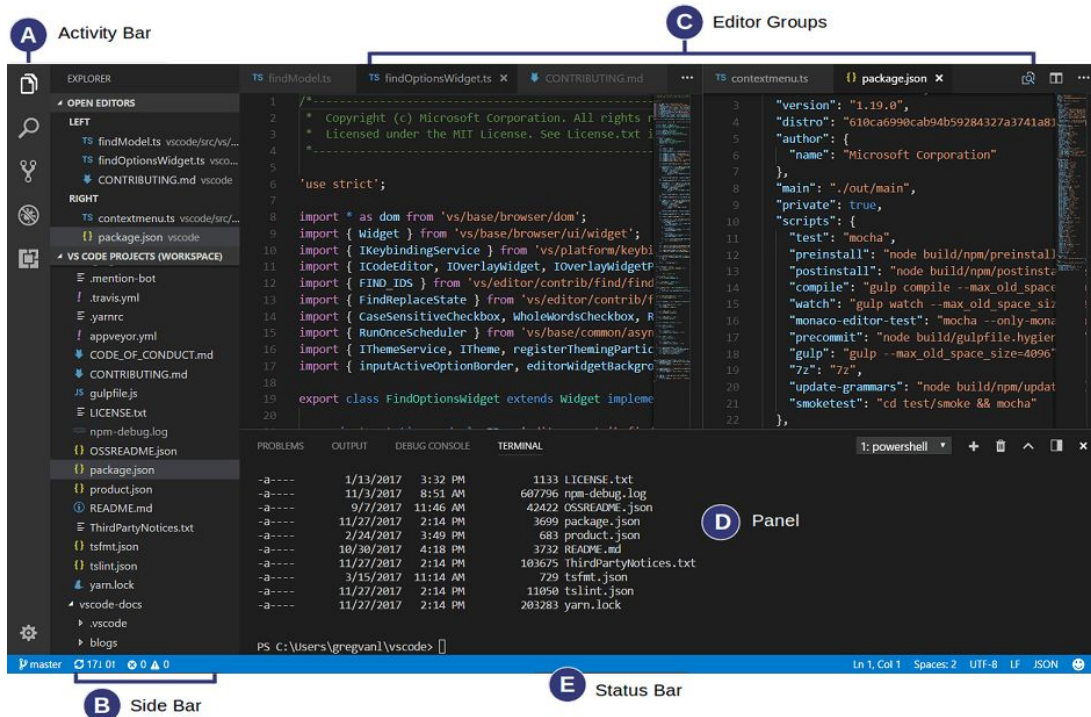
A.) Tevékenység sáv. A bal szélen található, lehetővé teszi a nézetek közötti váltást,

B.) Oldalsáv. Különböző nézeteket tartalmaz, például az Intézőt, hogy segítsen a projekten való munka során.

C.) Szerkesztők. A fő terület a fájlok szerkesztéséhez. Tetszőleges számú szerkesztőt nyithat meg egymás mellett függőlegesen és vízszintesen.

D.) Panel. Alapértelmezés szerint kimenetet, hibakeresési információkat, hibákat és figyelmeztetéseket, valamint egy integrált terminált tartalmaz. A panel balra vagy jobbra is mozgatható.

E.) Állapotsor. Információk a megnyitott projektről és a szerkesztett fájlokról.



Egy másodlagos oldalsáv is elérhető az elsődleges oldalsávval szembeni nézetek megjelenítéséhez. A Ctrl+Alt+B billentyűkombinációval lehet megjeleníteni.

A VS Code minden egyes indításakor ugyanabban az állapotban nyílik meg, mint amikor legutóbb bezárta. A mappa, az elrendezés és a megnyitott fájlok megmaradnak.

Szerkesztő felület és Minimap

Tetszőleges számú szerkesztőt nyitható meg egymás mellett függőlegesen és vízszintesen. A képernyő tetején Tabokban (lapokban) is lehet váltani a megnyitott fájlok között.

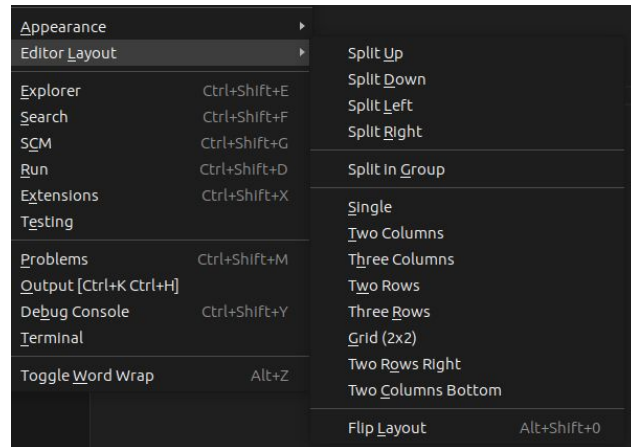
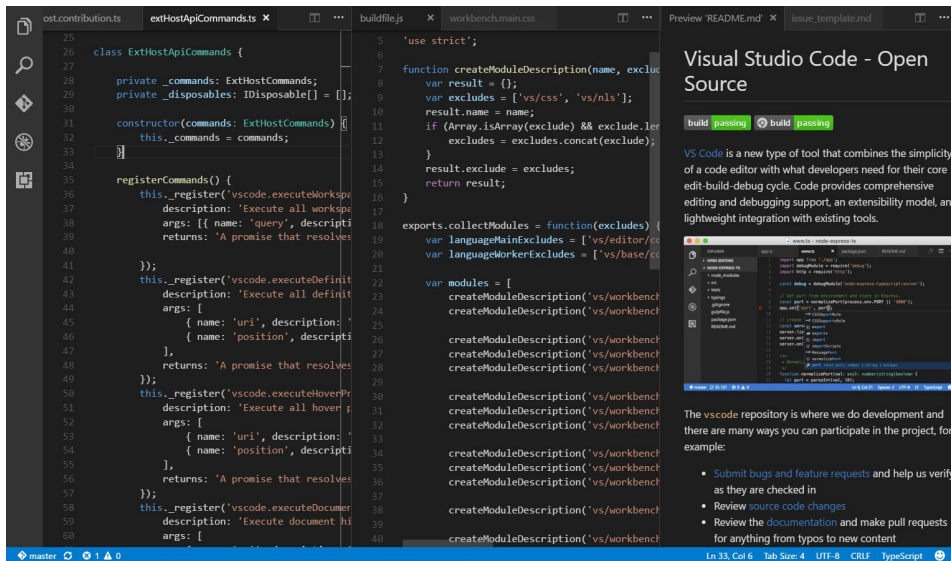
A fájlok közötti váltáshoz billentyűzetten a Ctrl + Tab al lehet váltani. Az osztott szerkesztők között pedig a Ctrl + 1 2 ...

Menüben a View -> Editor Layout pontban még több elrendezés választható (később még bemutatásra kerül)

A minimap egy átfogóbb betekintést nyújt az épp aktuális kódra. Megkönnyíti a kódban való mozgást. Szürke területtel jelöli az egy ablakban megjeleníthető kódrészletet és a szürke sávot húzva mozoghatunk a kódban.

A szintaktikai hibás sorok pirossal kiemelve jelennek meg és egyébként is követi a kód színezését.

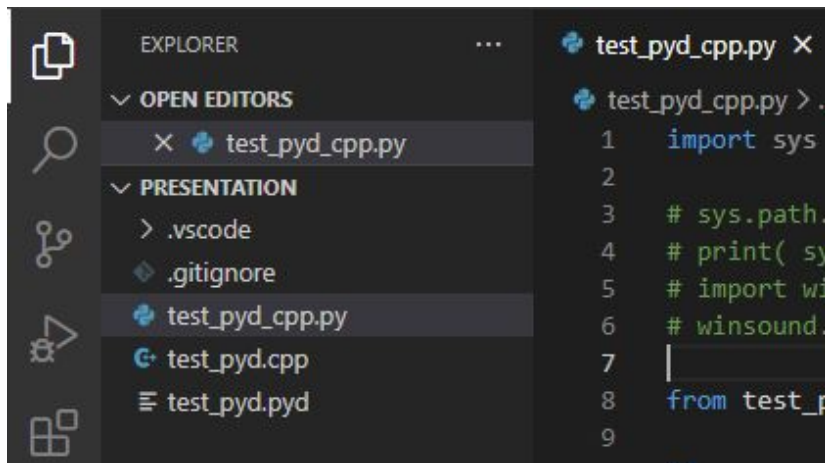
A minimap kikapcsolható vagy áthelyezhető.



Fájlkezelő

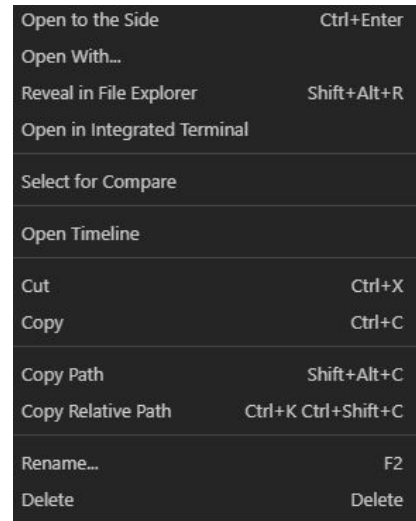
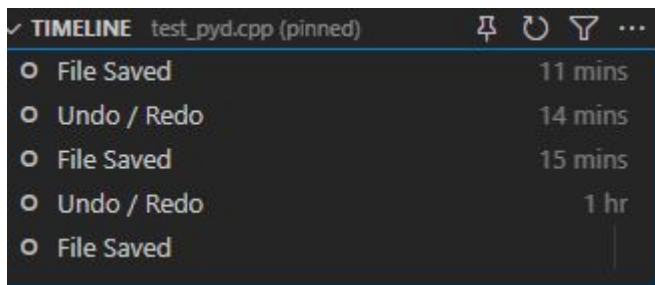
A tevékenység sáv legelső eleme a fájlkezelő

Felső rész a megnyitott fájlok felsorolása
Lentebb a munkaterület fájljai és mappái,
a mappák kinyithatóak és bezárhatóak a
könnyebb átláthatóság érdekében



A fájl jobb gombra menü nyílik meg alap fájl műveletek végezhetőek (átnevezés, törlés). Hasznos lehet a fájl nevének másolása (Copy path és Copy Relative Path)

Open Timeline, a fájlon elvégzett műveletek listáját kapjuk időrendben



Breadcrumbs (zsemlemorzsa)

A szerkesztő felett található egy navigációs sáv ahol egérekattintásra gyorsan lehet mozogni mappák és fájlok között.

Összetettebb mappa struktúránál lehet hasznos

Az utolsó bejegyzés az aktuális kódban az aktuális függvény neve amennyiben egy függvényben áll a cursor. Ilyenkor az elérhető függvények között is lehet mozogni

```
test_pyd.cpp > [🔍] superfastcode_methods
1  #include <stdio.h>
2  #include <math.h>
3  #include <stdbool.h>
4  #include <Python.h>
5  /**
```

```
C abstract.h ×
py3114 > include > cpython > C abstract.h > ...
1  #ifndef Py_PYTHON_ABSTRACTOBJECT_H
2  # error
3  #endif
4
5  /* === 0
6
```

- > cpython
- > greenlet
- > internal
- > pygame

```
C test_pyd.cpp > [🔍] superfastcode_methods
1  #include <stdio.h>
2  #include
3  #include
4  #include
5  /**
6  * python38 eseten
7  */
8  PyObject* square_impl(PyObject* /* unused module reference */, PyObject* o) {
9
```

- 🔍 square_impl(PyObject *, PyObject *)
- [🔍] superfastcode_methods
- [🔍] test_pyd_module
- 🔍 PyInit_test_pyd()

Keresés és csere a tevékenység sáv második pontja

A megnyitott fájlok vagy ha munkaterülettel dolgozunk a munkaterületen lévő fájlok tartalmában lehet keresni. A keresőmező melletti ikonok:

Aa - Kis nagybetű különböző

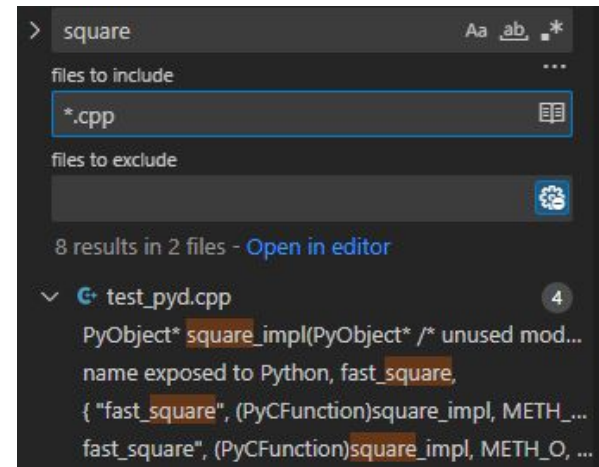
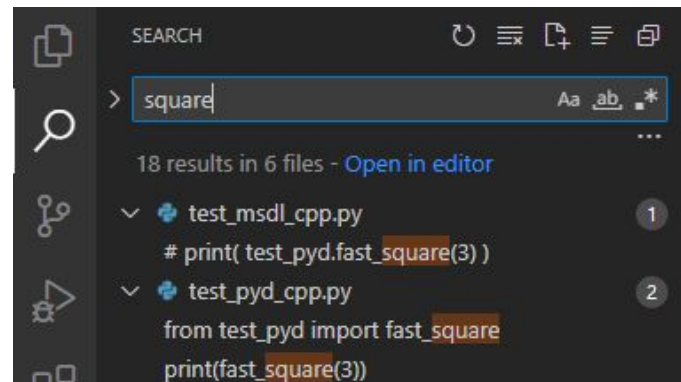
ab - Szóra keres

_* - reguláris kifejezés is megadható

... - További szűrések megnyitása

files to include - csak a megadott feltétel szerinti fájlok szűrése

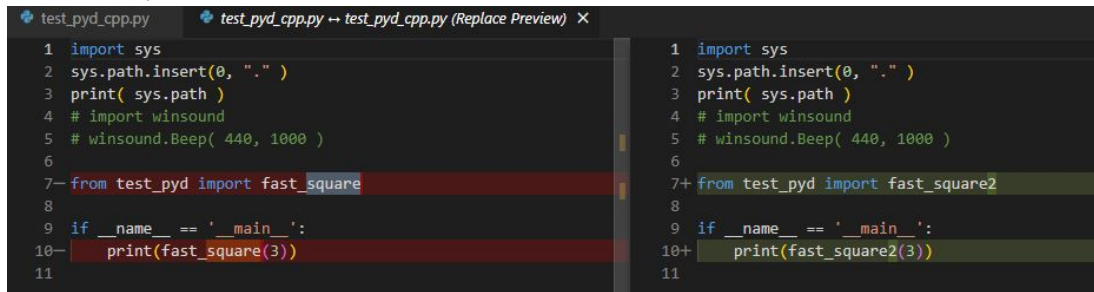
files to exclude - a megadott fájlok kizárása



Keresés és csere a tevékenység sáv második pontja

A keresett értékek cserélhetőek is.

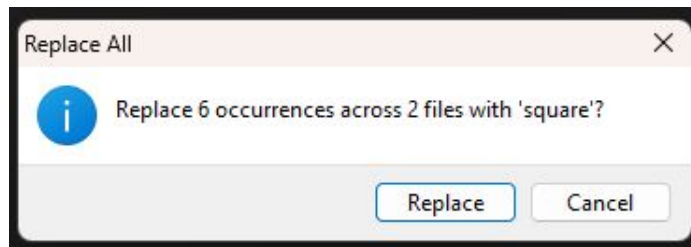
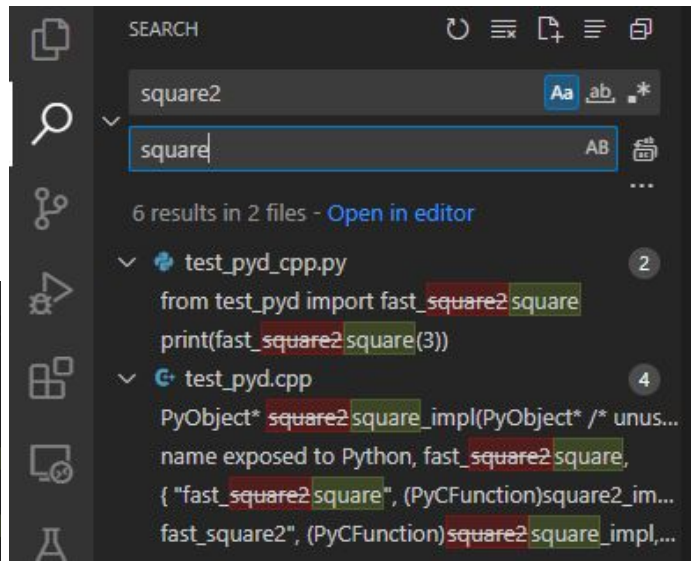
Ha a találati listában rákattintunk egy értékre
Akkor osztott nézetben megjelenik az adott fájl
előtte, utána nézete



```
test_pyd_cpp.py | test_pyd_cpp.py ↔ test_pyd_cpp.py (Replace Preview) X
1 import sys
2 sys.path.insert(0, ".")
3 print( sys.path )
4 # import winsound
5 # winsound.Beep( 440, 1000 )
6
7- from test_pyd import fast_square
8
9 if __name__ == '__main__':
10- print(fast_square(3))
11
```

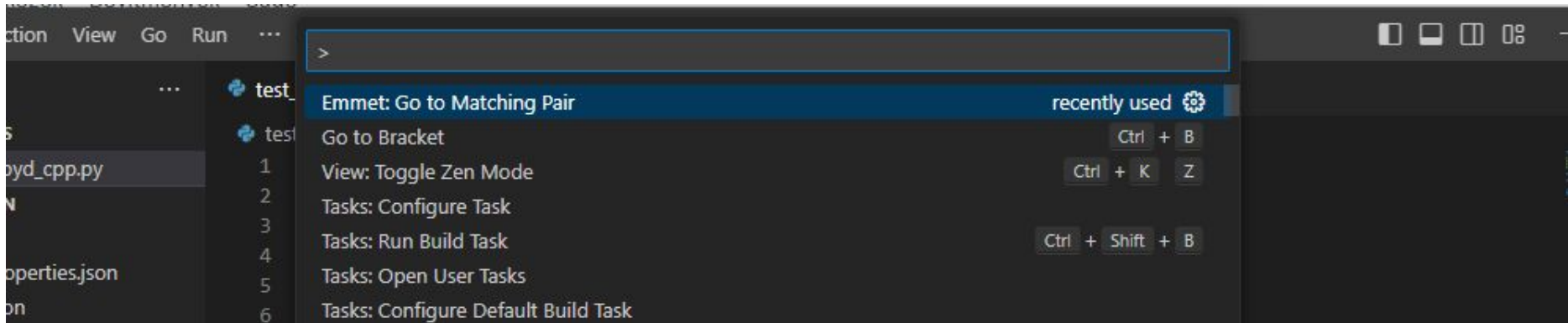
```
1 import sys
2 sys.path.insert(0, ".")
3 print( sys.path )
4 # import winsound
5 # winsound.Beep( 440, 1000 )
6
7+ from test_pyd import fast_square2
8
9 if __name__ == '__main__':
10+ print(fast_square2(3))
11
```

Választható az összes cseréje, ilyenkor egy megerősítés kérés után a munkaterület összes fájljában cserére kerül a megadott kifejezés (alkalmazása körültekintést igényel)



Parancskészlet

A parancskészlet tartalmazza az összes elérhető utasítást amit a szerkesztő ismer. A Ctrl + Alt + P billentyű kombinációval előhívható és a szerkesztő felső sávjában közepén jelenik meg (később erre a területre felső sávként hivatkozom) Elkezdve a gépelést a találatok szótöredékre szűrve jelennek meg. Az eredmények a felhasználás gyakorisága szerint rendeződnek, így a gyakran használt funkciókat előrébb találjuk, a gyorsbillentyűk is megjelennek jobb oldalt.



Nézetek és Zen mód

A rengeteg panel és nézet típus között el lehet vésni, erre megoldás, ha tudjuk, hogy melyik képernyőt szeretnénk elérni akkor billentyűzetről is el tudunk navigálni az épp kellő nézethez. Ctrl + P -re megnyílik a felső kereső és oda beírva a view (nézet) szót megjelennek az elérhető nézetek. Menüből is elérhető,

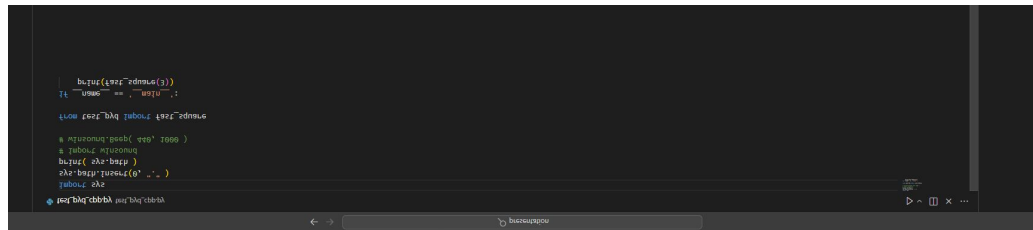
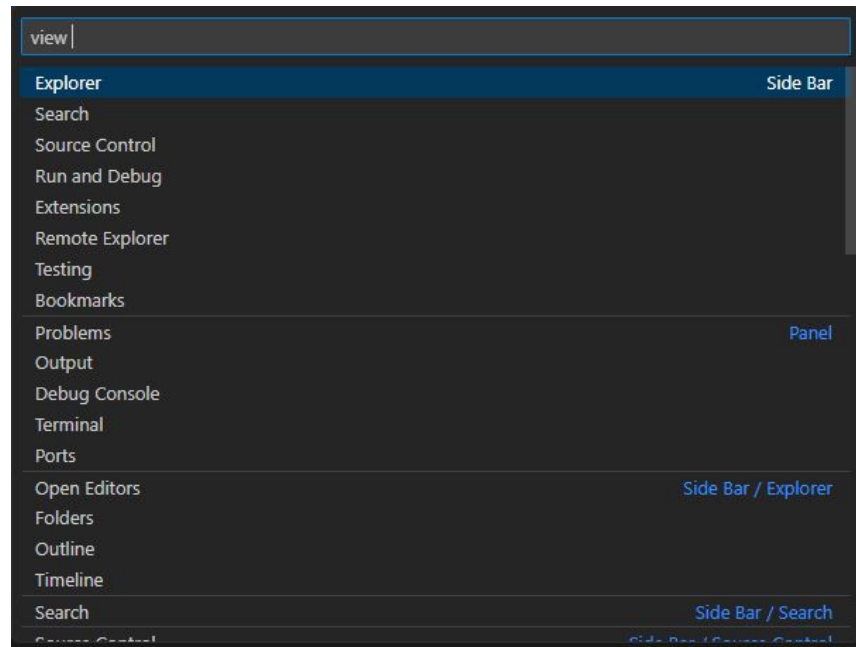
View -> Open View...

Zen mód

Ebben a nézetben csak az aktuális fájl jelenik meg, az összes egyéb panel el lesz rejtve.

Ctrl + Alt + P után toggle zen mode

13



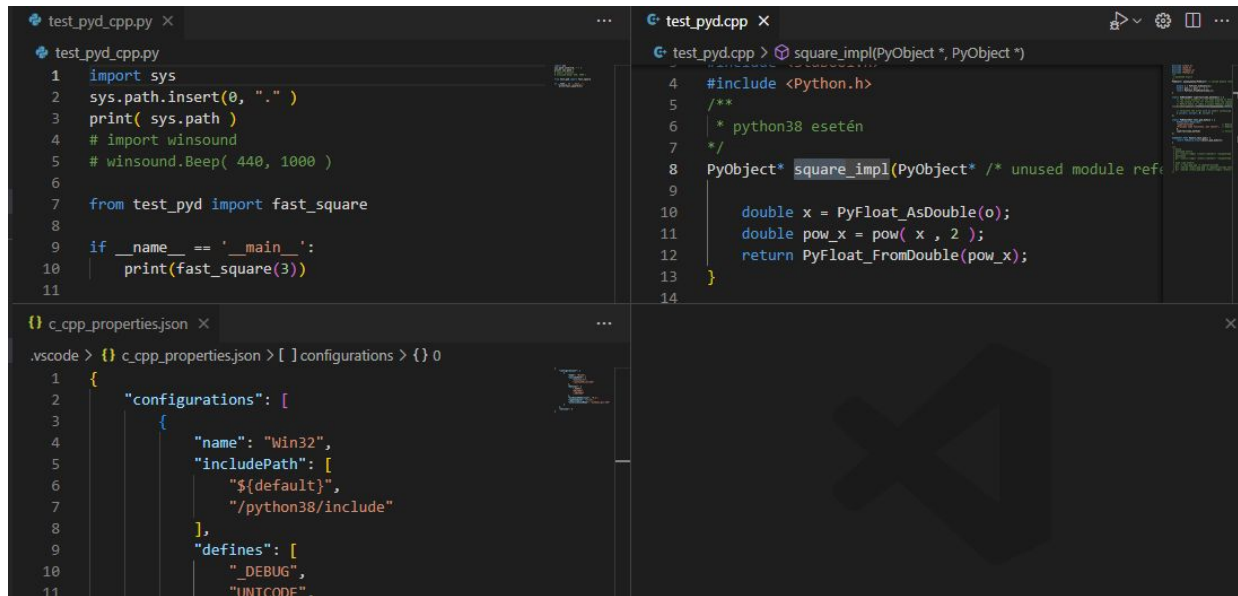
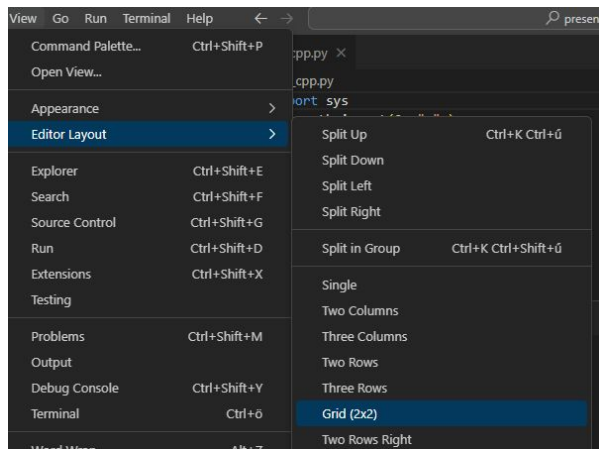
Rácsos elrendezés - Grid mode

Nem csak oszlopokban rendezhetők a megnyitott fájlok, hanem rácsos módban is.

Menüből elérhető View -> Editor Layout -> Grid (2x2)

Parancskészletből > Grid Editor Layout (2x2)

A megnyitott fájlokat behúzhatjuk az általunk kívánt részére a rácsnak



Témák, sötét téma és fájl ikonok

Több színséma (téma) választható alapértelmezetten vagy a VS Code Marketplace-ről is telepíthetők témák. Világos és sötét témák közül választhatunk. Első indításkor a rendszer beállítások szerint indul.

Windows és Linux (disztibuciónként változhat) alatt is már beállítható az alap rendszer téma, hogy világos vagy sötét legyen.

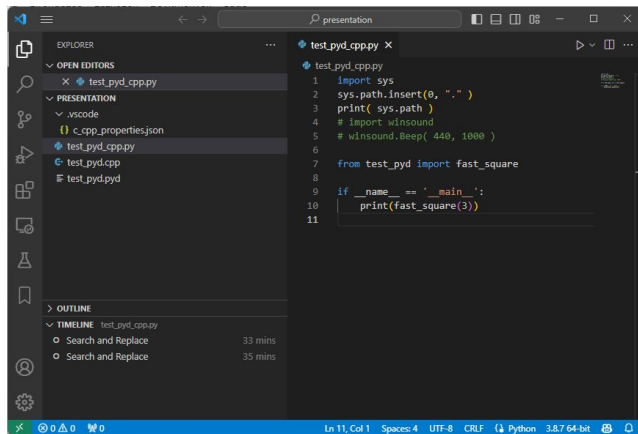
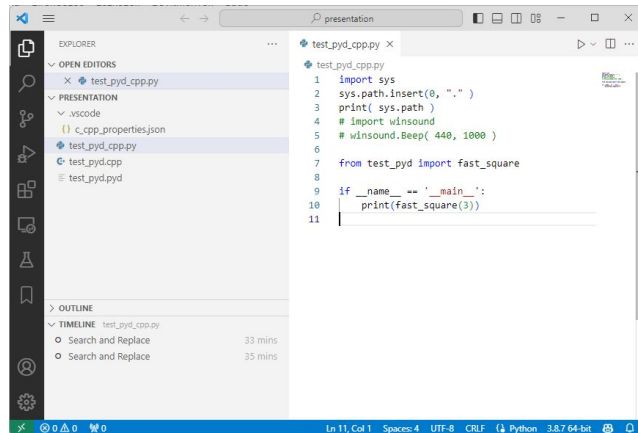
Menüből File -> Preferences -> Theme -> Color Theme

A színsémákkal a syntax highlight is változik

Megváltoztatható a fájlok ikonjai is, menüből ugyanitt

File Icon Theme

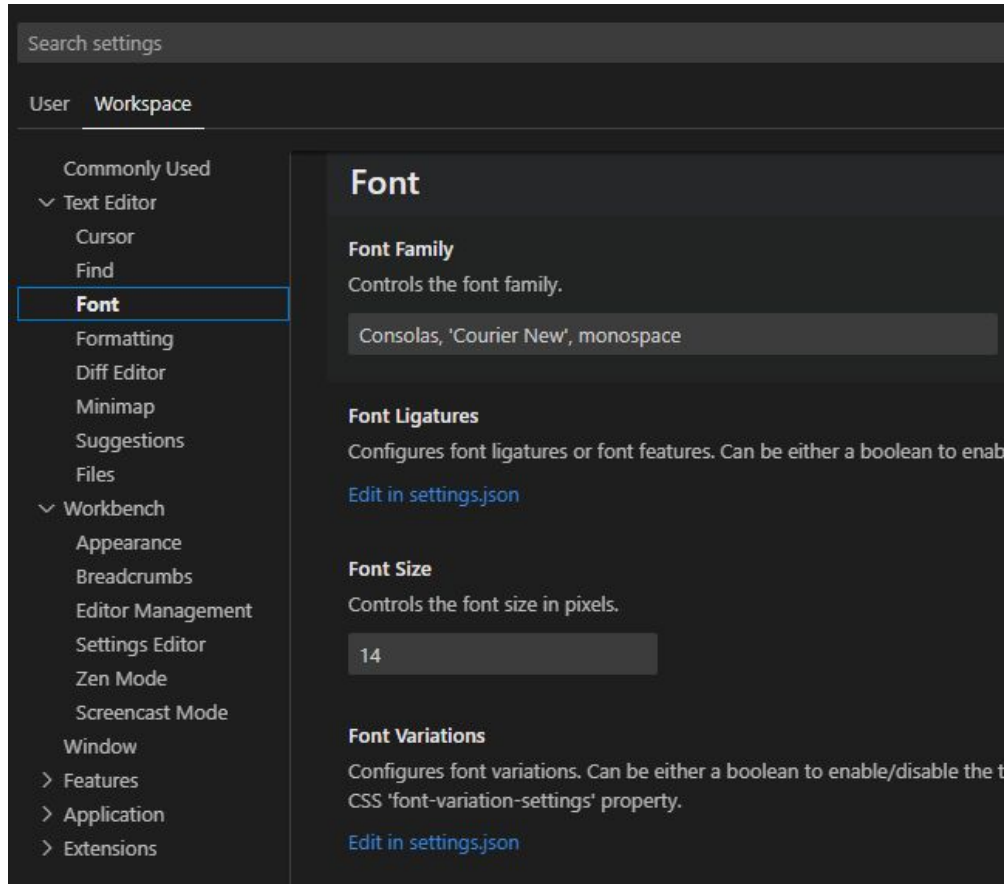
Minden téma testreszabható



Munkaterületek beállításai

A megnyitott fájlok vagy egy mappa fájljai munkaterületbe rendezhetőek.

A munkaterület külön testre szabható. Akár külön témát rendelhetünk az adott munkaterülethez, a szerkesztő beállításait módosíthatjuk munkaterületenként. Adott munkaterület adott fájljára is megadhatunk külön beállítást (pl.: karakterkódolás)



Billentyűzet parancsok

A billentyűzet parancsok meggyorsítják a munkát. VS Code esetén minden parancs gyorsbillentyűhöz rendelhető, de bővítményként komplett gyorsbillentyűzet kiosztás is telepíthető. Számomra nagyon hasznos a Notepad++ keymap

Hasznos és általában egységes billentyűzet kombinációk:

Ctrl + P fájlhoz ugrás

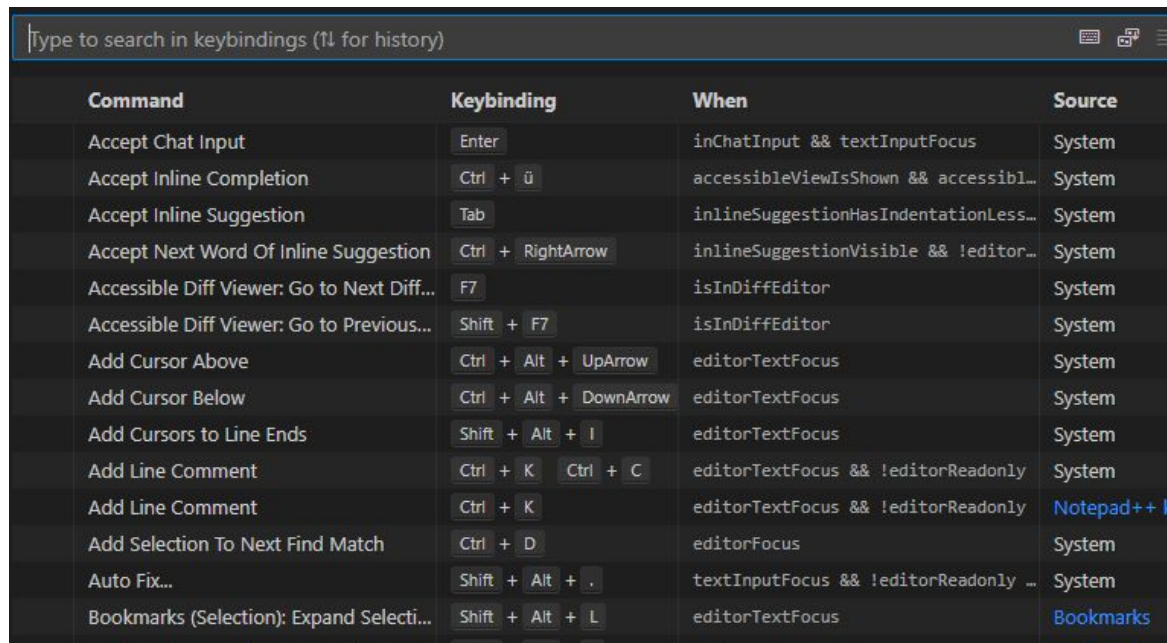
Ctrl + Shift + P parancskészlet

Menüből elérhető a gyorsbillentyű hozzárendelő felület

File -> Preferences ->

Keyboard Shortcuts

Ezen a felületen minden testreszabható



Command	Keybinding	When	Source
Accept Chat Input	Enter	inChatInput && textInputFocus	System
Accept Inline Completion	Ctrl + ü	accessibleViewIsShown && accessibl...	System
Accept Inline Suggestion	Tab	inlineSuggestionHasIndentationLess...	System
Accept Next Word Of Inline Suggestion	Ctrl + RightArrow	inlineSuggestionVisible && !editor...	System
Accessible Diff Viewer: Go to Next Diff...	F7	isInDiffEditor	System
Accessible Diff Viewer: Go to Previous...	Shift + F7	isInDiffEditor	System
Add Cursor Above	Ctrl + Alt + UpArrow	editorTextFocus	System
Add Cursor Below	Ctrl + Alt + DownArrow	editorTextFocus	System
Add Cursors to Line Ends	Shift + Alt + I	editorTextFocus	System
Add Line Comment	Ctrl + K Ctrl + C	editorTextFocus && !editorReadOnly	System
Add Line Comment	Ctrl + K	editorTextFocus && !editorReadOnly	Notepad++ + k
Add Selection To Next Find Match	Ctrl + D	editorFocus	System
Auto Fix...	Shift + Alt + .	textInputFocus && !editorReadOnly ...	System
Bookmarks (Selection): Expand Selecti...	Shift + Alt + L	editorTextFocus	Bookmarks

Összecsukhatóság Folding

A különböző kódblokkok tetszőlegesen összecsukhatóak, így növelve az átláthatóságot a kódban. A kódblokkok nyelvenként eltérőek lehetnek.

A nyilak abban az esetben jelennek meg ha az egeret a sorszámok utáni üres mezőre húzzuk és itt kattintással lehet ki és becsukni az adott kódblokkot.

Az összes kódblokk is ki és becsukható a parancskészletben a `Fold All` és `Unfold All` parancsokkal.

```
>fold all|
Unfold All
Fold All
Fold All Block Comments
Fold All Except Selected
Fold All Regions
Unfold All Except Selected
Unfold All Regions
```

```
24
25  v static PyModuleDef test_pyd_module = {
26      PyModuleDef_HEAD_INIT,
27      "super-fastcode",           // Module name to use with Pyt
28      "Provides some functions, but faster", // Module description
29      0,
30      superfastcode_methods      // Structure that defines the
31  };
32
33  > PyMODINIT_FUNC PyInit_test_pyd() { ...
36
```

Oszlopos és több kurzoros kijelölés

Lehetőség van több kurzoros kijelölésre és akár adatbevitelre és törlésre is.

Billentyűzettel az Alt + Shift és fel / le nyíl billentyűvel lehet több sorra kiterjeszteni a kijelölést. Több soron Visszatörlés és Törlés is alkalmazható valamint gépelve minden kijelölt soron megjelenik a begépelte szöveg.

Alt + Shift + Bal egér gomb megnyomásra pedig a kezdeti pozíciótól a kattintott területig lesz kijelölve doboz szerűen az adott részlet

Airi Satou	Accountant
Angelica Ramos	Chief Executive Officer (CEO)
Ashton Cox	Junior Technical Author
Bradley Greer	Software Engineer
Brenden Wagner	Software Engineer
Brielle Williamson	Integration Specialist
Bruno Nash	Software Engineer
Caesar Vance	PreSales Support
Cara Stevens	Sales Assistant
Cedric Kelly	Senior Javascript Developer

Airi Satou	Accountant	Tokyo	33	20081128	\$162,700
Angelica Ramos	Chief Executive Officer (CEO)	London	47	20091009	\$1,200,000
Ashton Cox	Junior Technical Author	San Francisco	66	20090112	\$86,000
Bradley Greer	Software Engineer	London	41	20121013	\$132,000
Brenden Wagner	Software Engineer	San Francisco	28	20110607	\$206,850
Brielle Williamson	Integration Specialist	New York	61	20121202	\$372,000
Bruno Nash	Software Engineer	London	38	20110503	\$163,500
Caesar Vance	PreSales Support	New York	21	20111212	\$106,450
Cara Stevens	Sales Assistant	New York	46	20111206	\$145,600
Cedric Kelly	Senior Javascript Developer	Edinburgh	22	20120329	\$433,060

Airi Satou	Accountant	11
Angelica Ramos	Chief Executive Officer (CEO)	11
Ashton Cox	Junior Technical Author	11
Bradley Greer	Software Engineer	11
Brenden Wagner	Software Engineer	11
Brielle Williamson	Integration Specialist	11
Bruno Nash	Software Engineer	11
Caesar Vance	PreSales Support	11
Cara Stevens	Sales Assistant	11
Cedric Kelly	Senior Javascript Developer	11

Kiegészítők telepítése, VS Code Marketplace

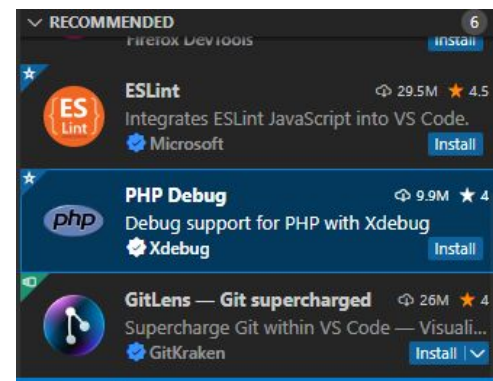
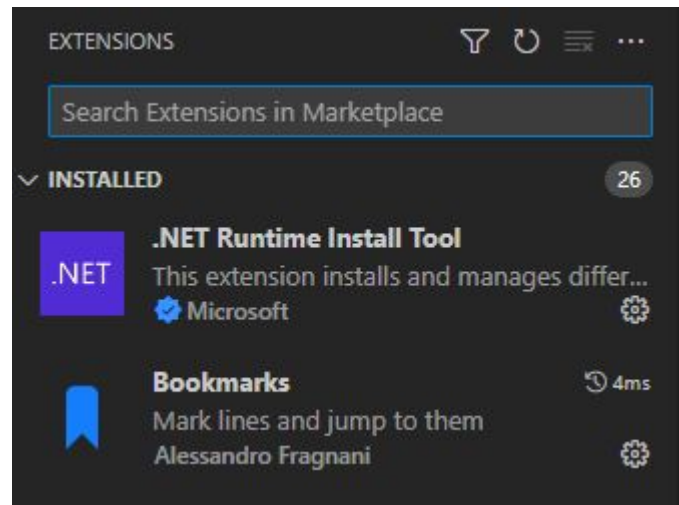
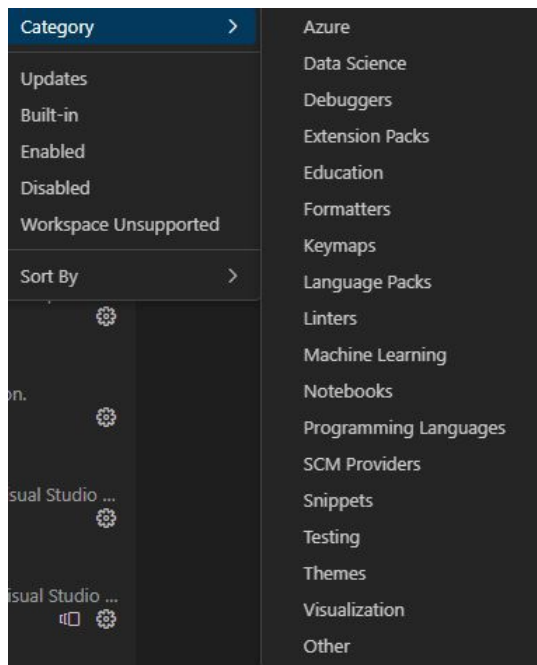
A bővítményekkel kiegészítheti az eleve bő lehetőségeit a szerkesztőnek.

A kiegészítők között kereshet szöveg alapon, az eredmények szűrhetők kategóriánként

A keresési eredmények között a kiegészítőre kattintva egy leíró oldal jelenik meg.

A bővítmény az Install gombra kattintva telepíthető. Ha nincs rá szükség akkor az uninstallal eltávolítható

20



IntelliSense kódkiegészítés

Az adott programnyelvhez tartozó függvények, osztályok névkiegészítését végzi. Ezen felül a kódban definiált változók neveire és osztályaira is a gépelést követően felajánlja a lehetséges értékeket. Ezen felül megadja függvények esetén a paraméterek neveit.

```
PyObject* square_impl(PyObject* /* unused module reference */, PyObject* o) {  
  
    double x = PyFloat_AsDouble(o);  
    double pow_x = pow( x , 2 );  
    return PyFloat_FromDouble(pow_x);  
}
```

PyFloat_FromDoub... PyObject *PyFloat_FromDouble(doubl...

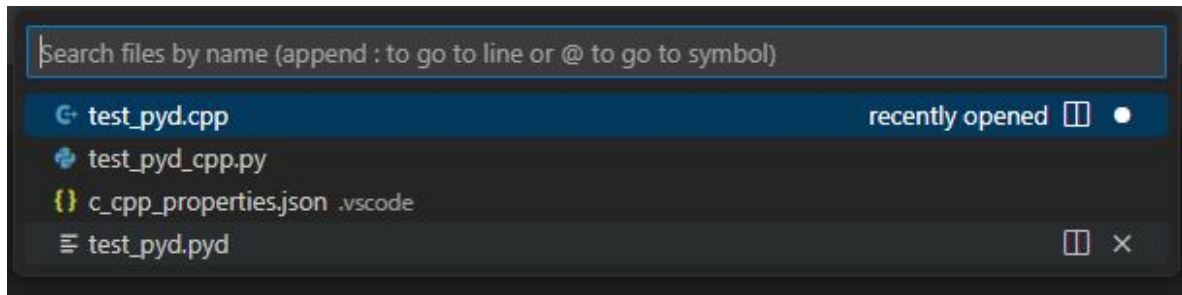
Ha használunk kommenteket a saját változókra vagy Függvényekre azok is megjelennek

```
MODINIT_FUNC PyInit_test_pyd() {  
    sq  
    re sqrt  
    sqrtf  
    sqrtl  
    square_impl  
    M_SQRT1_2  
    M_SQRT2  
    Py_sq_ass_item  
    Py_sq_concat  
    Py_sq_contains  
    Py_sq_inplace_concat  
    Py_sq_inplace_repeat
```

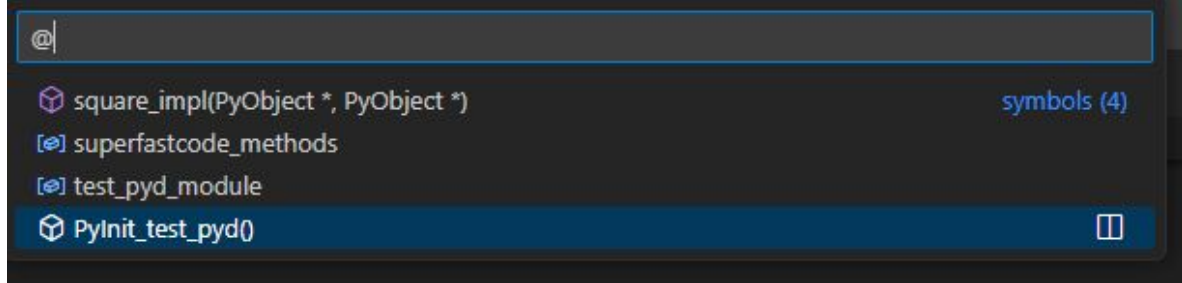
PyObject *square_impl(PyObject *, PyObject *o)
négyzetre emelés
Parameters:
nem használt objektum referencia a modulra
o – maga a paraméter
Returns:
a négyzetre emelt érték

Navigáció a fájlok között és a kódon belül

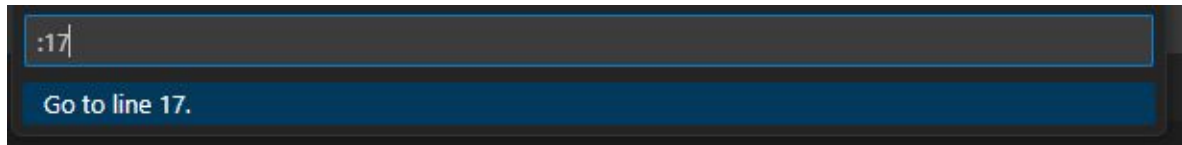
A munkaterület fájllai között gyors navigációt tesz lehetővé a Ctrl + P kereső



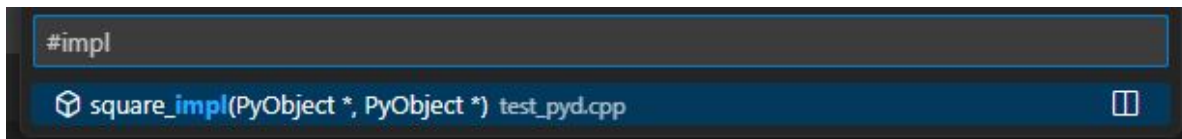
Kódon belül a funkciókra ugrás Ctrl + Shift + O



Adott sorra ugrás
Ctrl + G

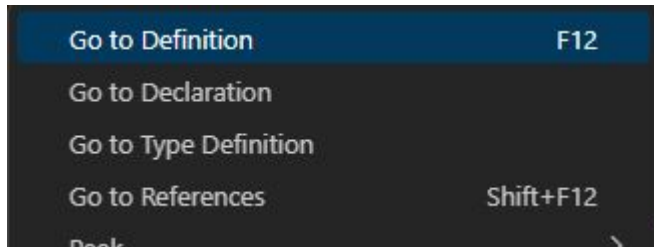


Szimbólum keresése
a munkaterületen



Navigáció a fájlok között és a kódon belül

Ugrás a kurzor pozícióban lévő változó definíciójára F12



Ugrás a kurzor alatt lévő változó hivatkozásaira. Ilyenkor egy plusz ablak jelenik meg a pozíció alatt ahol felsorolásra kerül az összes előfordulás

```
16 static PyMethodDef superfastcode_methods[] = {
test_pyd.cpp C:\munkak\egyeb\py_c_compile\presentation - References (2)
8 // @param o maga a parameter
9 // @return a négyzetre emelt érték
10 PyObject* square_impl(PyObject* /* unused module reference */, PyObject* o) {
11     double x = PyFloat_AsDouble(o);
12     double pow_x = pow( x , 2 );
13     return PyFloat_FromDouble(pow_x);
14 }
15
16 static PyMethodDef superfastcode_methods[] = {
17     // The first property is the name exposed to Python, fast_square,
18     // The second is the C++ function with the implementation
```

Ugrás a zárójel párjára Ctrl + B
Egyéb blokkok esetén is működik

```
PyMODINIT_FUNC PyInit_test_pyd() {
    return PyModule_Create(&test_pyd_module);
}
```

Változó átnevezése és szabványos komment blokk generálás

Változó átnevezése Rename symbol

```
left = bitonic_sort(arr[:n // 2], True)
right = bitonic_sort(arr[n // 2:], False)

arr = bitonic_merge(left + right, reverse)

return arr
```

right2
Enter to Rename, Shift+Enter to Preview

```
left = bitonic_sort(arr[:n // 2], True)
right2 = bitonic_sort(arr[n // 2:], False)

arr = bitonic_merge(left + right2, reverse)

return arr
```

Komment blokk generálása

Generate Doxygen Comment

```
/// @brief
/// @param type
/// @param args
/// @param kwds
/// @return
PyObject *MyClass_new(PyTypeObject *type, PyObject *args, PyObject *kwds){
```


Copilot, mesterséges intelligencia

Fizetős a Copilot, aktív github előfizetés szükséges hozzá.

Az AI több módszer szerint dolgozik, figyeli az adott kódon belül alkalmazott sajátosságokat, de a munkaterület többi fájlja alapján is dolgozik.

Van hogy olyan technikát ajánl amit korábban nem használtam és nagyon hasznos, hogy a commentekben is tesz ajánlásokat mindezt magyar nyelven.

```
led_sample.cpp > ...
189     int ledcnt = 1;
190     if (!PyArg_ParseTuple(args, "i", &ledcnt)) {
191         PyErr_SetString(PyExc_RuntimeError, "nem megfelelő paraméter" );
192         return NULL;
193     }
194     initLeds(ledcnt);
195     Py_RETURN_NONE;
196 }
197
198 static PyObject* setLedColor_py(PyObject* self, PyObject* args) {
    int idx = 0;
    int r = 0;
    int g = 0;
    int b = 0;
    if (!PyArg_ParseTuple(args, "iii", &idx, &r, &g, &b)) {
        PyErr_SetString(PyExc_RuntimeError, "nem megfelelő paraméter" );
        return NULL;
    }
    setLedColor(idx, r, g, b);
    Py_RETURN_NONE;
}
199
200 static PyObject* setLedLight_py(PyObject* self, PyObject* args) {
201     int idx = 0;
202     int light = 0;
203     if (!PyArg_ParseTuple(args, "ii", &idx, &light)) {
204         PyErr_SetString(PyExc_RuntimeError, "nem megfelelő paraméter" );
```

Copilot Chat, MI csetelés

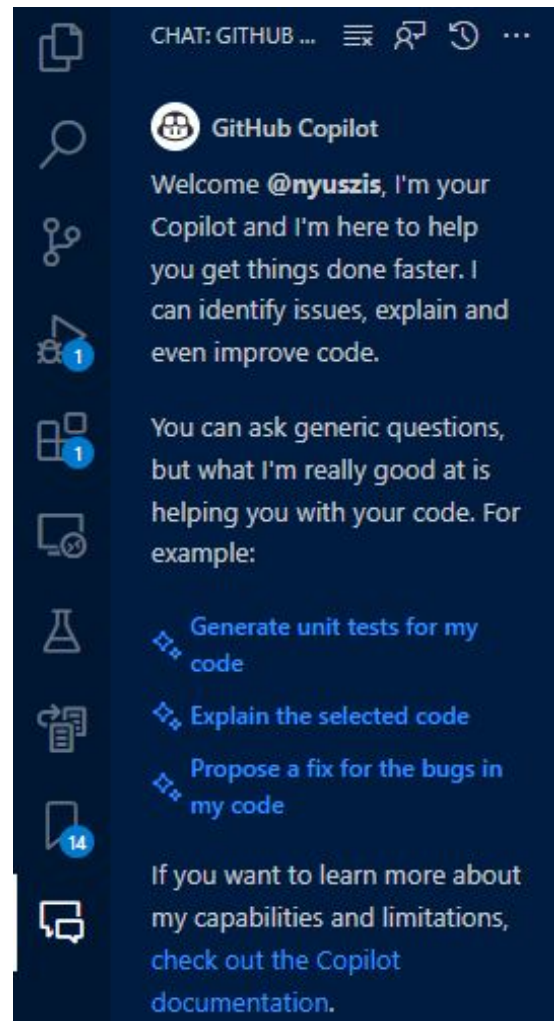
Megjelent egy új lehetőség ahol cseveghetünk az AI-val.

A tevékenység sávban megjelenik a Chat funkció.

A kívánt feladatokat szövegesen megfogalmazva kódrészleteket vagy akár teljes kódot jelenít meg az oldalsávban. Működik magyar és angol nyelven is, de magyarul nagyobb eséllyel kapunk kitérő választ mint kódot.

A magyarul megfogalmazott kérésre magyar válasz érkezik, az angolra angol (más nyelvet nem próbáltam).

A chat az aktív ablakban lévő kód környezetét felhasználva alakítja ki a választ igazodva az aktív kód programnyelvéhez.



Copilot Chat, MI csetelés

Angol nyelven az “add new function double” utasításra az oldalsávban megjelenítette a kódot. Ebben az esetben az új funkciót megjelenítette, de a korábban ott lévő (négyzetre emelés) nem, de utána hivatkozott a “fast_square” függvényre. Azaz ez a javaslat csak szerkesztéssel eredményezett fordítható kódot. Az include-ok közül is kihagyta a math.h -t, ezt a későbbiekben is következetesen kihagyta.



CHAT: GITHUB COPILOT

Sure, here's an example of how you can add a new Python function that doubles a given number using the existing `square_impl` function as a template:

```
#include <Python.h>

PyObject* double_impl(PyObject* /* unused module reference */, Py
double x = PyFloat_AsDouble(o);
double doubled_x = x * 2;
return PyFloat_FromDouble(doubled_x);
}

static PyMethodDef superfastcode_methods[] = {
    { "fast_square", (PyCFunction)square_impl, METH_O, nullptr },
    { "fast_double", (PyCFunction)double_impl, METH_O, nullptr },
    { nullptr, nullptr, 0, nullptr }
};

static PyModuleDef test_pyd_module = {
    PyModuleDef_HEAD_INIT,
    "superfastcode",
    "Provides some functions, but faster",
    0,
    superfastcode_methods
};

PyMODINIT_FUNC PyInit_test_pyd() {
    return PyModule_Create(&test_pyd_module);
}
```

This adds a new function called `fast_double` that takes a single argument and returns its double.

[How can I compile this Python module?](#)

Ask Copilot or type '/' for commands

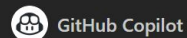
Copilot Chat, MI csetelés

A következő utasítást magyar nyelven fogalmaztam meg és egy nehezebb feladatot kértem tőle, ahol használni kellett olyan függvényeket amelyek nem voltak az előző kódban és azokat be is kellett regisztrálni, hogy működő kódot kapjunk. Ezt a feladatot nagyon jól megoldotta, egy fordítható működő kódot adott a válaszban. Most minden korábbi funkciót is felsorolt egyedül az include-ot hagyta ki. A fordításra is adott használható javaslatot.



nyuszis

adj hozzá egy funkciót ami több paramétert összeszorzoz



Természetesen! Az alábbi kódrészletben hozzáadtam egy új függvényt, amely több paramétert szoroz össze a `square_impl` és `double_impl` függvények mintájára:

```
#include <Python.h>
```

```
PyObject* square_impl(PyObject* /* unused module reference */, Py  
double x = PyFloat_AsDouble(o);  
double pow_x = pow(x, 2);  
return PyFloat_FromDouble(pow_x);  
}
```

```
PyObject* multiply_impl(PyObject* /* unused module reference */, |  
double result = 1.0;  
int arg_count = PyTuple_Size(args);  
for (int i = 0; i < arg_count; ++i) {  
    PyObject* o = PyTuple_GetItem(args, i);  
    double x = PyFloat_AsDouble(o);  
    result *= x;  
}  
return PyFloat_FromDouble(result);  
}
```

```
static PyMethodDef superfastcode_methods[] = {  
    { "fast_square", (PyCFunction)square_impl, METH_O, nullptr },  
    { "fast_double", (PyCFunction)double_impl, METH_O, nullptr },  
    { "fast_multiply", (PyCFunction)multiply_impl, METH_VARARGS, |
```

Copilot Chat, MI csetelés

A következő utasítást csak angolul volt hajlandó végrehajtani. Itt a python kódban kértem tesztelő funkciókat. Egy működő és használható tesztelő kódot adott válaszban. Minden eredményre helyes választ adott.

```
c:\munkak\egyeb\py_c_compile>pytest -v test_pyd_cpp.py
===== test session starts =====
platform win32 -- Python 3.8.7, pytest-7.4.0, pluggy-1.2.0 -- C:\python38\python.exe
cachedir: .pytest_cache
rootdir: c:\munkak\egyeb\py_c_compile
plugins: anyio-3.6.2, Faker-16.6.0
collected 3 items

test_pyd_cpp.py::test_fast_square PASSED [ 33%]
test_pyd_cpp.py::test_fast_double PASSED [ 66%]
test_pyd_cpp.py::test_fast_multiply PASSED [100%]

===== 3 passed in 0.14s =====
```



nyuszis

add pytest functions



GitHub Copilot

Sure! Here's an example of how you can add `pytest` functions to your Python code:

```
from test_pyd import fast_square, fast_double, fast_multiply

def test_fast_square():
    ... assert fast_square(3) == 9.0
    ... assert fast_square(4) == 16.0
    ... assert fast_square(5) == 25.0


def test_fast_double():
    ... assert fast_double(3) == 6.0
    ... assert fast_double(4) == 8.0
    ... assert fast_double(5) == 10.0


def test_fast_multiply():
    ... assert fast_multiply(3, 4) == 12.0
    ... assert fast_multiply(2, 3, 4) == 24.0
    ... assert fast_multiply(5, 6, 7, 8) == 1680.0
```

This adds three `pytest` test functions that test the `fast_square`, `fast_double`, and `fast_multiply` functions respectively. You can run the tests by running the `pytest` command in the terminal.

Copilot Chat, MI csetelés

Végül arra kértem egy mondatban angolul, hogy készítsen egy programot, amit én internetes keresésekből és saját ötletből készítettem ehhez az előadáshoz. Ezt hibátlanul megoldotta, még átadott paraméter ellenőrzést is tett bele ami az én megoldásomból kimaradt.

 nyuszis
create program c python pyd one function multiply

 GitHub Copilot
Sure! Here's an example of how you can create a C++ program that exports a `multiply` function to Python using a `.pyd` file:

```
#include <Python.h>

double multiply(double x, double y) {
    return x * y;
}

PyObject* multiply_impl(PyObject* /* unused module reference */, PyObject* args) {
    double x, y;
    if (!PyArg_ParseTuple(args, "dd", &x, &y)) {
        return nullptr;
    }
    double result = multiply(x, y);
    return PyFloat_FromDouble(result);
}

static PyMethodDef test_pyd_methods[] = {
    { "multiply", (PyCFunction)multiply_impl, METH_VARARGS, nullptr },
    { nullptr, nullptr, 0, nullptr }
};

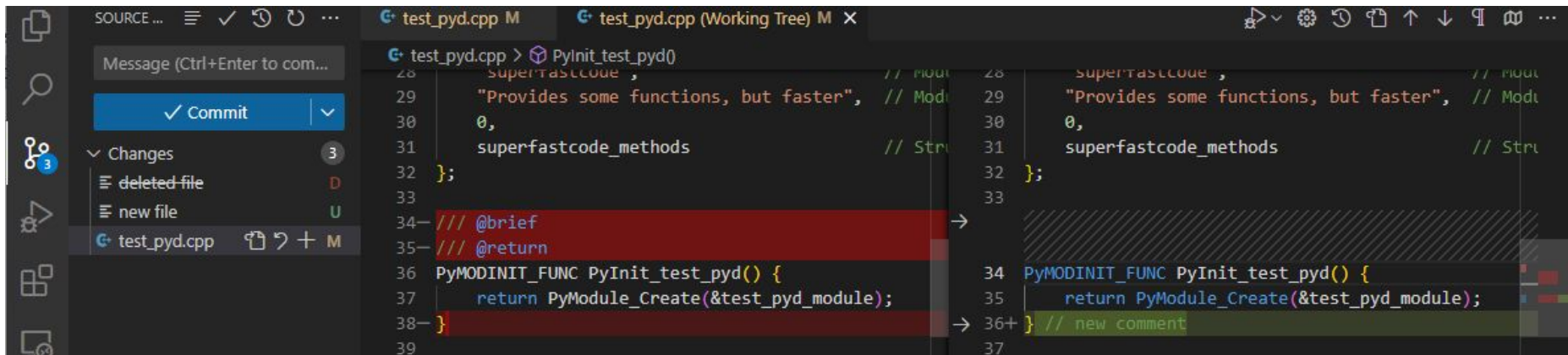
static PyModuleDef test_pyd_module = {
    PyModuleDef_HEAD_INIT
```


Tevékenység sáv, verziókövetés GIT

Az oldalsávban megjelennek a módosított fájlok. A fájlra kattintva megjeleníti a korábbi és a jelenlegi verzió közötti változásokat. Színezéssel jelöli a törölt (piros) és az új részeket (zöld)

A commithoz a fájl változás esetén hozzáadható + vagy visszavonható egy változtatás ↻. A fájl neve mellett M jelöli a változást, D a törölt, U az új fájlt

Az új commit és a remote repository szinkronizálás is elvégezhető a sávban.



```
test_pyd.cpp M
test_pyd.cpp (Working Tree) M X

Message (Ctrl+Enter to com...
Commit
Changes
deleted-file D
new file U
test_pyd.cpp ↻ ↶ + M

test_pyd.cpp > PylNit_test_pyd()
28     superfastcode , // modul
29     "Provides some functions, but faster", // Modul
30     0,
31     superfastcode_methods // Stru
32 };
33
34- /// @brief →
35- /// @return →
36 PyMODINIT_FUNC PyInit_test_pyd() {
37     return PyModule_Create(&test_pyd_module);
38- } →
39
34     superfastcode , // modul
35     "Provides some functions, but faster", // Modul
36     0,
37     superfastcode_methods // Stru
38 };
39
34 PyMODINIT_FUNC PyInit_test_pyd() {
35     return PyModule_Create(&test_pyd_module);
36+ } // new comment
37
```

Snippets kódtörédek

Nyelvenként változó kódrészletek hozhatók létre, miután elkezdjük beírni az utasítást és Ctrl + Space paranccsal a Code snippets -el kezdődő sort választjuk

```
PyMODINIT_FUNC PyInit_test_pyd() {
    return PyModule_Create(&test_pyd_module);
    for (size_t i = 0; i < count; i++)
    {
        /* code */
    }
} // new comment
```

```
for
/ ne for
for
for
```

Code snippet for for loop

További snippets-ek telepíthetők a Marketplace-ről vagy magunk is hozhatunk létre.

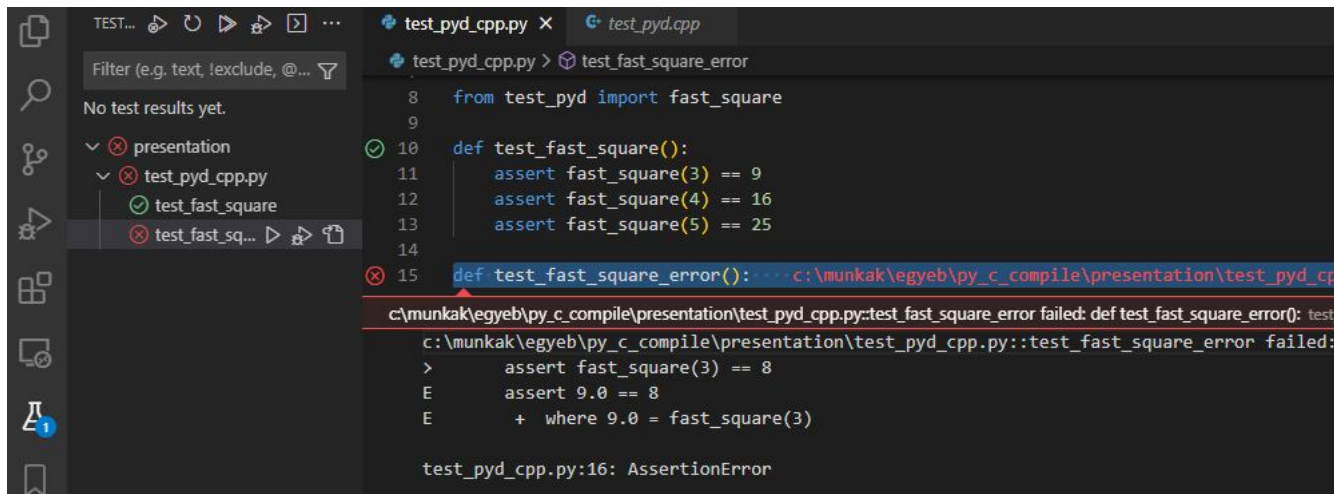
A példa egy javascript for ciklust tartalmaz

```
// in file 'Code/User/snippets/javascript.json'
{
  "For Loop": {
    "prefix": ["for", "for-const"],
    "body": ["for (const ${2:element} of ${1:array}) {", "\t$0", "}"],
    "description": "A for loop."
  }
}
```


Tesztelő, python tesztelő beállítása

A tevékenységsávban a tesztelést választva az oldalsávjában válasszuk a Configure Python Tests pontot. Megjelenik egy kérdés a felső sávban ahol a pyTest -et válasszuk.

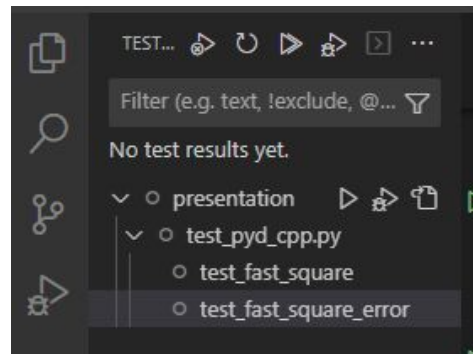
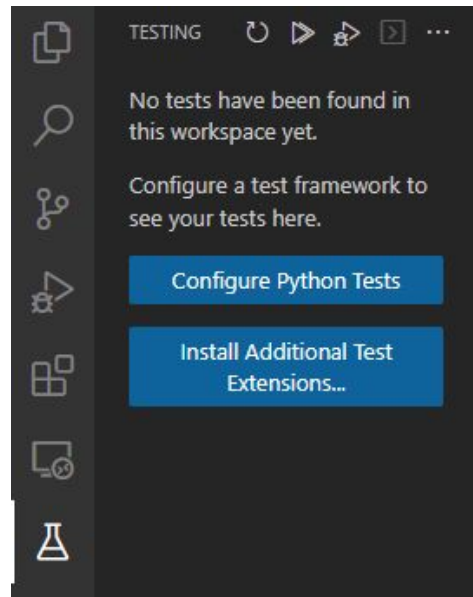
A fájlokban megkeresi a tesztekét és a háromszögre kattintva futtatja azokat. Az eredményeket hiba esetén a szerkesztőben jeleníti meg



```
TEST... [refresh] [play] [stop] [close] [more]
test_pyd_cpp.py x test_pyd.cpp
test_pyd_cpp.py > test_fast_square_error
8 from test_pyd import fast_square
9
10 def test_fast_square():
11     assert fast_square(3) == 9
12     assert fast_square(4) == 16
13     assert fast_square(5) == 25
14
15 def test_fast_square_error():
16     assert fast_square(3) == 8
17     assert 9.0 == 8
18     + where 9.0 = fast_square(3)

c:\munkak\egyeb\py_c_compile\presentation\test_pyd_cp
c:\munkak\egyeb\py_c_compile\presentation\test_pyd_cpp.py::test_fast_square_error failed: def test_fast_square_error(): test
> assert fast_square(3) == 8
E assert 9.0 == 8
E + where 9.0 = fast_square(3)

test_pyd_cpp.py:16: AssertionError
```

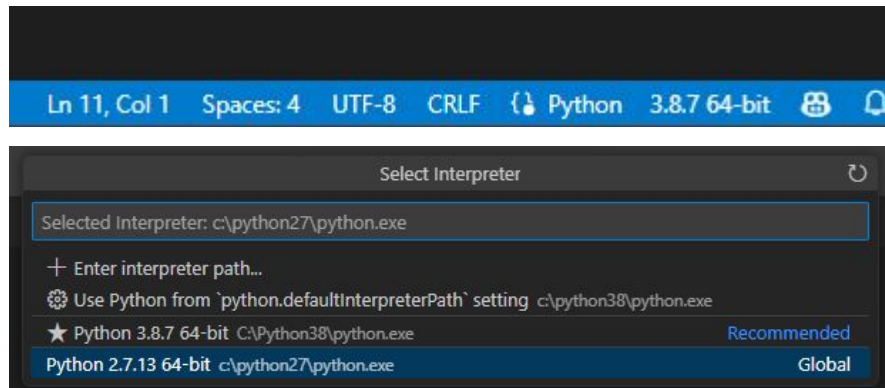


Python nyelvi munkaterület létrehozása

Python esetén a futtatókörnyezetet lehet meghatározni, a verzióra kattintva a fenti sávban jelennek meg a választható opciók.

Enter interpreter path... pontban új futtatókörnyezet is gyorsan hozzáadható.

A kiválasztott interpreter nincs hatással a linter működésére, a python 2 szintaktika nincs támogatva, így hibát fog mutatni az amúgy 2-es pythonban lefutó kód.

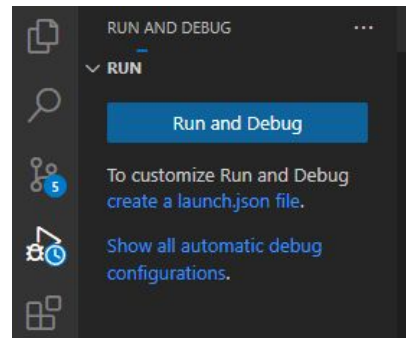


Python hibakereső (debug) környezet beállítása

A tevékenység sávban található a hibakereső panel.

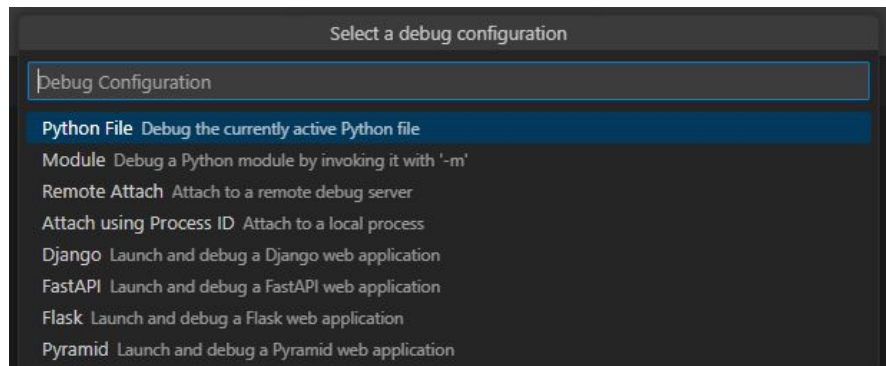
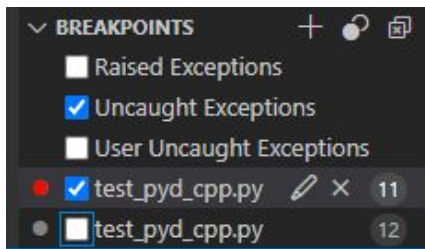
A Run and Debug gombra kattintva a felső sávban választhatunk több környezet közül, mi most a Python File-t válasszuk

Töréspontot a sorszám melletti területre kattintva vehetünk fel



```
10 if __name__ == '__main__':
11     a = 10
12     a = a + fast_square(3)
13     print(a)
14     print(fast_square(3))
15     print(fast_square(4))
16     print(fast_square(5))
```

Az oldalsávban karbantarthatóak az aktív töréspontok



Python hibakeresés

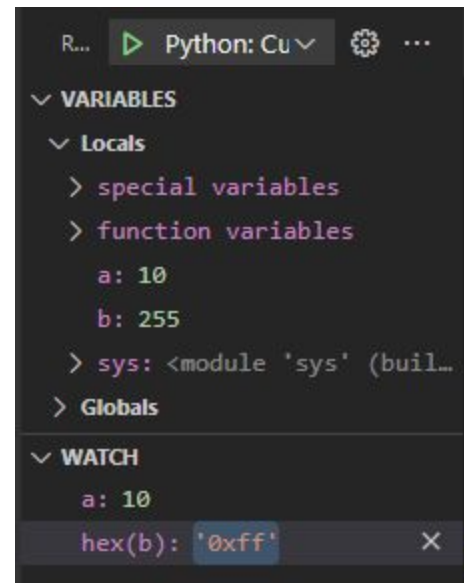
A hibakeresés menüből Run -> Start Debugging vagy F5 billentyűvel indítható

A töréspontnál az oldalsávban megjelennek az aktuális változók az épp aktuális értékekkel.

Megfigyeltethetők a változók a WATCH panelen is, itt megadható függvény is (a példában a b változó hexában)

A Run menüpontban láthatóak az elérhető funkciók, amik gyors billentyűvel is elérhetőek.

A felső sávban megjelenő gombokkal is vezérelhetjük a futást



C nyelvi munkaterület létrehozása

Beállítható a fordítási környezet
Én a gcc 64 bites verzióját
használok

Megadható saját definíció

```
{
  "configurations": [
    {
      "name": "Win32",
      "includePath": [
        "${default}",
        "c:\\python38\\include\\"
      ],
      "defines": [
        "_DEBUG",
        "UNICODE",
        "_UNICODE"
      ],
      "windowsSdkVersion": "8.1",
      "cppStandard": "c++17",
      "intelliSenseMode": "windows-gcc-x64",
      "cStandard": "c17"
    }
  ],
  "version": 4
}
```

IntelliSense mode

The IntelliSense mode to use that maps to a platform and architecture variant of MSVC, gcc, or Clang. If not set or if set to `${default}`, the extension will choose the default for that platform. Windows defaults to `windows-msvc-x64`, Linux defaults to `linux-gcc-x64`, and macOS defaults to `macos-clang-x64`. Select a specific IntelliSense mode to override the `${default}` mode. IntelliSense modes that only specify `<compiler>-<architecture>` variants (e.g. `gcc-x64`) are legacy modes and are converted automatically to the `<platform>-<compiler>-<architecture>` variants based on the host platform.

windows-gcc-x64

Defines

A list of preprocessor definitions for the IntelliSense engine to use while parsing files. Optionally, use `=` to set a value, e.g. `VERSION=1`.

One definition per line.

```
_DEBUG
UNICODE
_UNICODE
```

A végeredmény egy JSON fájl, ez akár kézzel is létrehozható, módosítható.

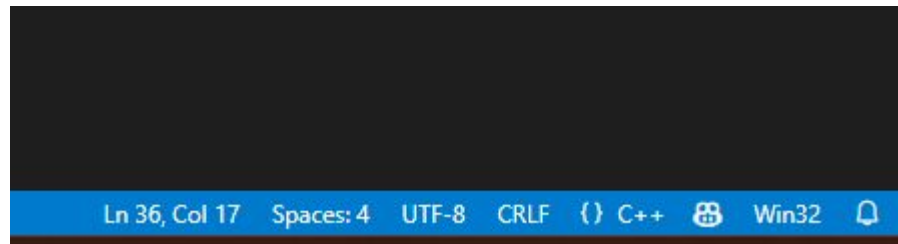
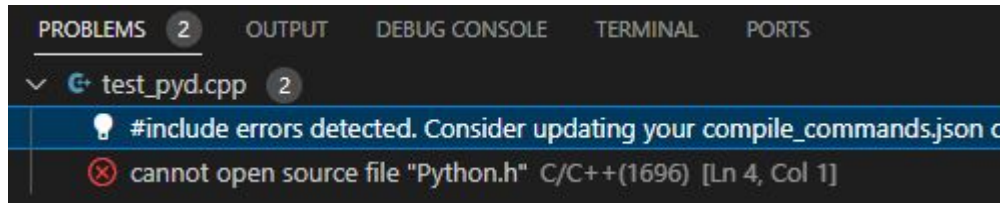
További fordítók is hozzáadhatóak a Marketplace segítségével.

C nyelvi munkaterület létrehozása

A mintapéldánk hibát jelez, mert nem találja a “Python.h” header fájlt. Ezt javíthatjuk, ha beállítjuk a munkaterületünkre vonatkozó include keresési könyvtárakat

A win32-re kattintva megjelenik a felső sávban egy menü, itt érdemes a UI konfigurációval elvégezni a beállítást.

Az include path részben felvéve a könyvtárat, megszűnik a hiba.



Include path

An include path is a folder that contains header files (such as `#include "myHeaderFile.h"`) that are included in a source file. Specify a list of paths for the IntelliSense engine to use while searching for included header files. Searching on these paths is not recursive. Specify `**` to indicate recursive search. For example, `${workspaceFolder}/**` will search through all subdirectories while `${workspaceFolder}` will not. If on Windows with Visual Studio installed, or if a compiler is specified in the `compilerPath` setting, it is not necessary to list the system include paths in this list.

One include path per line.

```

${default}
c:\python38\include\
```

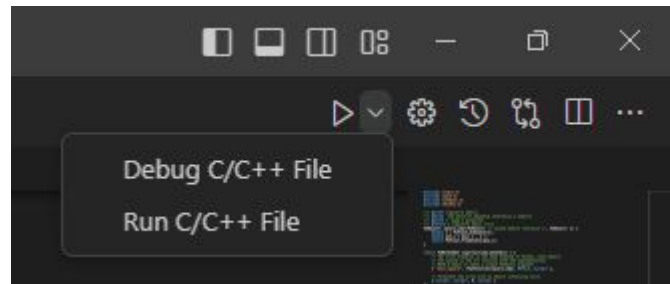
Win32

Edit Configurations (UI)

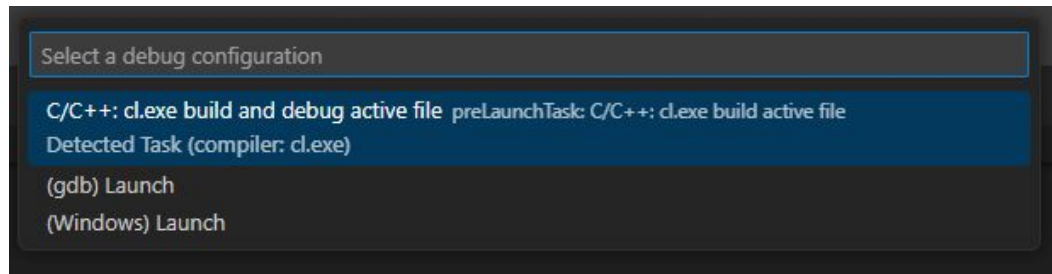
Edit Configurations (JSON)

C fordítás és hibakeresés beállítása

Amikor cpp fájl az aktív a szerkesztőben a jobb sarokban a nyíl ikon melletti lenyíló ikonra kattintva válasszuk a Run C/C++ File pontot



A felső sávban válasszuk a build and debug active file -t tartalmazó sort.



Ez a művelet létrehoz egy tasks.json fájlt és új bejegyzést tesz a launch.json fájlba. A fájlok a .vscode mappába kerülnek. A tasks.json a fordításért felel, a launch.json pedig a debug futtatásért.

A launch fájl már korábban létrejött amikor a python debug-ot futtattuk.

C fordítás és hibakeresés beállítása

Amiket módosítani kell:

label - bekerült, hogy DLL-t fordítunk

args - tömbként a fordítási paraméterek

-static-* - ne használjon runtime -ot

-shared - DLL-t fordítunk

-I/python38/include - a python.h helye

/python38/python38.dll - a dll-ben lévő

függvényeket használjuk, nem statikust

-o utáni paraméterben a .exe -t .pyd-re

átírjuk

```
{
  "tasks": [
    {
      "type": "cppbuild",
      "label": "C/C++: g++.exe build DLL file",
      "command": "C:/Program Files/mingw64/bin/g++.exe",
      "args": [
        "-static-libgcc",
        "-static-libstdc++",
        "-lwinpthread",
        "-shared",
        "-fdiagnostics-color=always",
        "-g",
        "${file}",
        "-I/python38/include",
        "/python38/python38.dll",
        "-o",
        "${fileDirname}\\${fileBasenameNoExtension}.pyd"
      ],
      "options": {
        "cwd": "C:/Program Files/mingw64/bin"
      },
      "problemMatcher": [
        "$gcc"
      ],
      "group": "build",
      "detail": "Task generated by Debugger."
    }
  ],
  "version": "2.0.0"
}
```


C fordítás és hibakeresés beállítása

Amiket módosítani kell:

name - hogy felismerhető legyen, DLL kerül bele

program - itt a python.exe, mert azt indítjuk

args - átadjuk a py fájlt, hogy azt futtassa

cwd - \${fileDirname} a fájluk könyvtára

environment - ezt csak akkor kell

beállítani, ha a környezeti változók nem

tartalmazzák

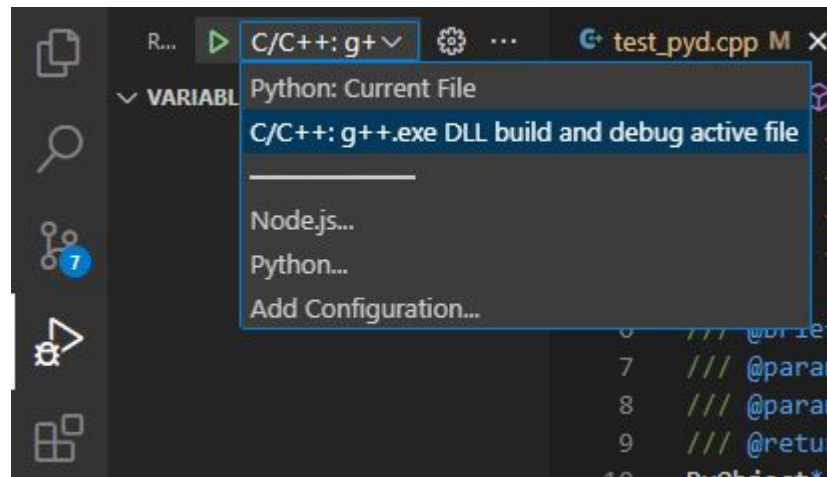
preLaunchTask - az előző dián a label

amivel fordítunk

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Python: Current File",
      "type": "python",
      "request": "launch",
      "program": "${file}",
      "console": "integratedTerminal",
      "justMyCode": true
    },
    {
      "name": "C/C++: g++.exe DLL build and debug active file",
      "type": "cppdbg",
      "request": "launch",
      "program": "c:\\python38\\python.exe",
      "args": ["${fileDirname}\\test_pyd_cpp.py"],
      "stopAtEntry": false,
      "cwd": "${fileDirname}",
      "environment": [{"name": "PYTHONHOME", "value": "c:\\python38"}],
      "externalConsole": false,
      "MIMode": "gdb",
      "miDebuggerPath": "C:\\Program Files\\mingw64\\bin\\gdb.exe",
      "setupCommands": [
        {
          "description": "Enable pretty-printing for gdb",
          "text": "-enable-pretty-printing",
          "ignoreFailures": true
        },
        {
          "description": "Set Disassembly Flavor to Intel",
          "text": "-gdb-set disassembly-flavor intel",
          "ignoreFailures": true
        }
      ],
      "preLaunchTask": "C/C++: g++.exe build DLL file"
    }
  ]
}
```

Hibakeresés C/C++ program esetén

a cpp fájl legyen kiválasztva a szerkesztőben
a zöld nyíl mellett nyissuk le a választót
és válasszuk az előző dián megadott labelt
C/C++: g++.exe build DLL file

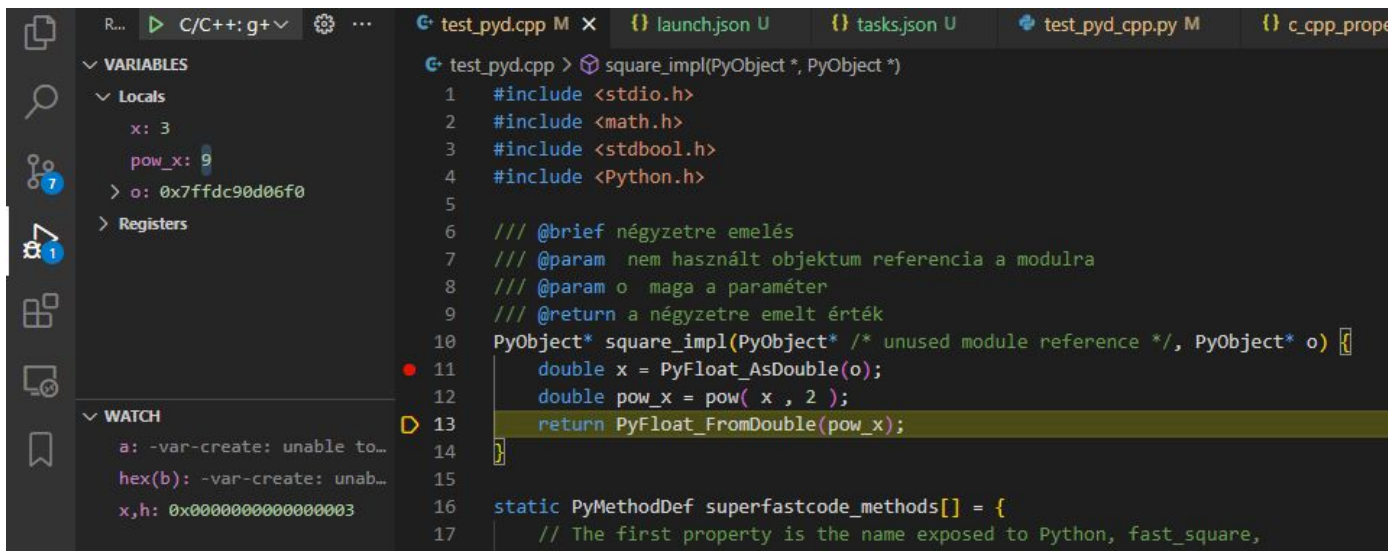


A zöld nyíllra kattintva
lefut a programunk,
alul a Panel sávban a
Terminal fülön
megjelenik a program
kimenete

```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS C:\munkak\egyeb\py_c_compile\presentation> & 'c:\Users\nyuszis\.vscode\extensions\ms-vscode
v' '--pid=Microsoft-MIEngine-Pid-vffvcntq.k5a' '--dbgExe=C:\Program Files\mingw64\bin\gdb.exe
19.0
9.0
16.0
25.0
PS C:\munkak\egyeb\py_c_compile\presentation>
```

Hibakeresés C/C++ program esetén

Hozzunk létre töréspontot és az előzőek szerint futtatva a kód megáll a töréspontnál, ahol ellenőrizhetjük a működést. C esetén a WATCH megjelenítőben ,h -val jeleníthetjük hexában a változó értékét



The screenshot shows a debugger interface with the following components:

- Left Panel:**
 - VARIABLES:**
 - Locals:
 - x: 3
 - pow_x: 9
 - o: 0x7ffdc90d06f0
 - Registers:
 - 1
 - WATCH:**
 - a: -var-create: unable to...
 - hex(b): -var-create: unab...
 - x,h: 0x0000000000000003
- Right Panel (Code Editor):**

```
test_pyd.cpp > square_impl(PyObject *, PyObject *)
1  #include <stdio.h>
2  #include <math.h>
3  #include <stdbool.h>
4  #include <Python.h>
5
6  /// @brief négyzetre emelés
7  /// @param nem használt objektum referencia a modulra
8  /// @param o maga a paraméter
9  /// @return a négyzetre emelt érték
10 PyObject* square_impl(PyObject* /* unused module reference */, PyObject* o)
11     double x = PyFloat_AsDouble(o);
12     double pow_x = pow( x , 2 );
13     return PyFloat_FromDouble(pow_x);
14
15
16 static PyMethodDef superfastcode_methods[] = {
17     // The first property is the name exposed to Python, fast_square,
```

A példakód fájljai

A példakód egy c/c++ python modult tartalmaz. Bemutatásra kerül, hogyan kommunikál a python a C-ben írt modullal. A c-ben megírt modulok pyd kiterjesztéssel fordított külső könyvtárak. Windows esetén DLL, linux esetén SO

- `test_pyd.cpp` - c python modul példakód
- `test_pyd_cpp.py` - python kód ami a péda modult hívja
- `test_pyd.pyd` - befordított modul, python 3.8 esetén működik, egyéb esetben újra kell fordítani
- `.vscode/c_cpp_properties.json` - a c linter beállításaira szolgáló fájl
- `.vscode/tasks.json` - a c kód fordítását beállító fájl
- `.vscode/launch.json` - a hibakerső futtatását beállító fájl. python 3.8-hoz lett beállítva, más verzió esetén módosítást igényel

A példakód c-ben írt python modul kódja

```
#include <math.h>
#include <Python.h>
/// @brief négyzetre emelés
/// @param nem használt objektum referencia a modulra
/// @param o maga a paraméter
/// @return a négyzetre emelt érték
PyObject* square_impl(PyObject* /* unused module reference */, PyObject* o)
{
    double x = PyFloat_AsDouble(o);
    double pow_x = pow( x , 2 );
    return PyFloat_FromDouble(pow_x);
}
PyObject* double_impl(PyObject* /* unused module reference */, PyObject* o) {
    double x = PyFloat_AsDouble(o);
    double doubled_x = x * 2;
return PyFloat_FromDouble(doubled_x);
}
PyObject* multiply_impl(PyObject* /* unused module reference */, PyObject* args) {
    double result = 1.0;
    int arg_count = PyTuple_Size(args);
    for (int i = 0; i < arg_count; ++i) {
        PyObject* o = PyTuple_GetItem(args, i);
        double x = PyFloat_AsDouble(o); result *= x; }
    return PyFloat_FromDouble(result);
}
}
```

A példakód c-ben írt python modul kódja

```
static PyMethodDef superfastcode_methods[] = {
    { "fast_square", (PyCFunction)square_impl, METH_O, nullptr },
    { "fast_double", (PyCFunction)double_impl, METH_O, nullptr },
    { "fast_multiply", (PyCFunction)multiply_impl, METH_VARARGS, nullptr },
    { nullptr, nullptr, 0, nullptr }
};

static PyModuleDef test_pyd_module = {
PyModuleDef_HEAD_INIT, "superfastcode", "Provides some functions, but faster", 0, superfastcode_methods
};

PyMODINIT_FUNC PyInit_test_pyd() {
    return PyModule_Create(&test_pyd_module);
}
```


A példakód python fájlja ami a C modult hívja

```
from test_pyd import fast_square, fast_double, fast_multiply
```

```
def test_fast_square():
```

```
    assert fast_square(3) == 9.0
```

```
    assert fast_square(4) == 16.0
```

```
    assert fast_square(5) == 25.0
```

```
def test_fast_double():
```

```
    assert fast_double(3) == 6.0
```

```
    assert fast_double(4) == 8.0
```

```
    assert fast_double(5) == 10.0
```

```
def test_fast_multiply():
```

```
    assert fast_multiply(3, 4) == 12.0
```

```
    assert fast_multiply(2, 3, 4) == 24.0
```

```
    assert fast_multiply(5, 6, 7, 8) == 1680.0
```

```
def test_fast_square_error():
```

```
    assert fast_square(3) == 8
```

```
if __name__ == '__main__':
```

```
    print(fast_square(3))
```

```
    print(fast_square(4))
```

```
    print(fast_square(5))
```